Trustworthy and Efficient Knowledge Sharing across Tasks in Deep Neural Networks

A thesis submitted in fulfillment of the requirements
for the degree of

Doctor of Philosophy

by

Haiyan Zhao

Under the supervision of

A/Prof. Guodong Long
Dist/Prof. Chengqi Zhang
A/Prof. Jing Jiang

to

Australian Artificial Intelligence Institute
Faculty of Engineering and Information Technology
University of Technology Sydney
NSW - 2007, Australia

June 2023

© 2023 by Haiyan Zhao
All Rights Reserved

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Haiyan Zhao, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science, Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 8th June, 2023

PLACE: Sydney, Australia

Abstract

Deep neural networks (DNNs) have revolutionized various fields with their impressive performance on particular tasks, but their increasing complexity raises concerns about trustworthiness and efficiency. In real-world applications, tasks are often interrelated, raising the necessity for DNNs to share and transfer knowledge reliably and effectively between tasks.

This thesis delves into the pivotal topic of trustworthy and efficient knowledge sharing across tasks in DNNs. Firstly, we develop a human-understandable NeuroChain from the pre-trained model for each task, aiming to illuminate the reasoning process of the model for the given task and to foster trust in the shared knowledge. Next, based on the interpretable sparse network of each task, we enhance the efficiency of DNNs by leveraging the shared knowledge of existing tasks, thereby enabling the rapid and reliable creation of well-performing sparse models for new tasks. Finally, we further investigate the knowledge sharing of parameters in DNNs when tackling sequential tasks. We propose a novel approach to mitigate forgetting and enhance the fairness of DNNs across tasks. We conduct extensive experiments to demonstrate the efficacy of our proposed methods on convolutional neural networks and vision transformers across multiple datasets. Our proposed methods consistently outperform popular baseline methods in terms of accuracy and efficiency.

In summary, this thesis offers valuable insights into the trustworthy and efficient sharing of knowledge across tasks in DNNs. Our proposed methods hold the potential

to enhance the reliability and effectiveness of these networks across a diverse range of real-world applications.

Dissertation directed by A/Prof. Guodong Long and A/Prof. Jing Jiang

Australian Artificial Intelligence Institute

Faculty of Engineering and IT

University of Technology Sydney

ACKNOWLEDGMENTS

During my doctoral studies, I have experienced many challenges, both in my academic pursuits and personal life. I am grateful to have had the support and encouragement of many people, without whom I would not have been able to complete this journey.

First and foremost, I would like to express my deepest gratitude to my supervisors, Guodong Long and Jing Jiang, for their unwavering support throughout my doctoral studies. Their expertise, encouragement, and patience have been instrumental in shaping my research and helping me achieve my academic goals. I am also deeply grateful to Tianyi Zhou for his invaluable guidance, which has been critical to the success of my research. Their immense knowledge and plentiful experience have encouraged me throughout my academic research and daily life. I would like to thank my friends, Shuang Ao and Zhuowei Wang, for their kind help and camaraderie. Their friendship and support have made the journey more enjoyable and less stressful.

Last but not least, I would like to extend my heartfelt gratitude to my family for their encouragement and love. Their sacrifices and understanding have enabled me to pursue my academic goals and achieve my dreams.

Haiyan Zhao

Sydney, Australia, 2023.

List of Publications

Conference Papers

C-1. **Haiyan Zhao**, Tianyi Zhou, Jing Jiang, Guodong Long, Chengqi Zhang. "Does Continual Learning Equally Forget All Parameters?," accepted by International Conference on Machine Learning, 2023.

C-2. **Haiyan Zhao**, Tianyi Zhou, Jing Jiang, Guodong Long, Chengqi Zhang. "Vote for Nearest Neighbors Meta-Pruning of Self-Supervised Networks," accepted by European Conference on Machine Learning, 2023.

C-3. **Haiyan Zhao**, Guodong Long. "One-Shot Pruning for Fast-adapting Pre-trained Models on Devices," submitted to AAAI Conference on Artificial Intelligence, 2024. (Under review)

Journal Papers

J-1. **Haiyan Zhao**, Tianyi Zhou, Jing Jiang, Guodong Long, Chengqi Zhang. "Extracting Local Reasoning Chains of Deep Neural Networks", accepted by Transactions on Machine Learning Research, 2022.

TABLE OF CONTENTS

List of Publications vii

List of Figures xiii

List of Tables xvii

1 Introduction 1

1.1 Background . . . 1

1.1.1 Knowledge Sharing of Interpretable Pre-trained Models . . . 3

1.1.2 Improve DNN's Efficiency by Knowledge Sharing . . . 6

1.1.3 Sharing Knowledge towards Fairness across Sequential Tasks . . . 8

1.2 Research Problems . . . 10

1.2.1 Interpretable Inference Process of Tasks in the Pre-trained Model 12

1.2.2 Reliable and Efficient Model Pruning by Knowledge Sharing . . . 13

1.2.3 Efficient Bias Mitigation in Learning Sequential Tasks . . . 15

1.3 Thesis Organization . . . 16

2 Literature Review 19

2.1 Knowledge Sharing between Tasks . . . 19

2.1.1 Transfer Learning . . . 19

2.1.2 Multi-task Learning . . . 23

2.2 Model Interpretability . . . 27

2.3 Model Pruning . . . 29

2.4 Continual Learning . . . 32

3 Interpretable Reasoning Process of Various Tasks 37

3.1 Introduction . . . 37

3.2 NeuroChains . . . 39

3.2.1 Problem: Extract Sub-networks . . . 39

3.2.2 Algorithm . . . 42
3.3 Experiments . . . 43
3.3.1 Implementation Details . . . 44
3.3.2 Quantitative Analyses . . . 45
3.3.3 Case Studies . . . 49
3.3.4 Detailed Analysis of Case Studies . . . 53
3.3.5 NeuroChains of Vision Transformer . . . 57
3.4 Ablation Studies . . . 59
3.5 Conclusion . . . 62

4 Reliable Model Pruning by Knowledge Sharing between Tasks 65
4.1 Introduction . . . 65
4.2 Empirical Study: Pruning a Pre-trained Model for Different Tasks . . . 67
4.2.1 A Dataset of Pruned Models . . . 67
4.2.2 Relationship between Pruned Models of Different Tasks . . . 70
4.3 Meta-Vote Pruning (MVP) . . . 73
4.3.1 Implementation Details . . . 73
4.4 Experiments . . . 75
4.4.1 Baseline Methods . . . 76
4.4.2 Main Results . . . 76
4.4.3 Performance on Unseen Dataset . . . 78
4.4.4 Results of MVP on Sub-tasks of Different Sizes . . . 80
4.5 Ablation Studies . . . 81
4.6 Conclusion . . . 84

5 Debiased Knowledge Sharing across Sequential Tasks 85
5.1 Introduction . . . 85
5.2 Problem Setup . . . 87
5.3 Forgetting of Different Parameters: An Empirical study . . . 88
5.3.1 Measuring Forgetting via Training Dynamics . . . 89
5.3.2 Forgetting of Different Parameters During CL . . . 89
5.4 Forgetting Prioritized Finetuning Methods . . . 94
5.4.1 Selection of sensitive parameters for FPF and k-FPF . . . 97
5.5 Experiments . . . 98
5.5.1 Main Results . . . 99
5.5.2 Finetuning Different Parameters in FPF and k-FPF . . . 101

5.5.3 Analysis of FPF and k-FPF in Different Scenarios 104
5.6 Ablation Studies 107
5.7 Conclusion 109

6 Conclusions and Future Work **111**
6.1 Conclusions 111
6.2 Future Work 112

Bibliography **115**

List of Figures

Figure **Page**

1.1 The development of popular DNNs in the past few years. Each circle represents a deep neural network. The x-axis indicates the publication date, while the y-axis refers to the training FLOPs (floating-point operations per second). The size of each circle corresponds to the number of parameters in the network. 2

1.2 Three critical problems studied in this thesis. 11

3.1 Size and fidelity (how well the sub-networks preserve the original pre-trained DNN's outputs) of 1,500 sub-networks extracted by NeuroChains for VGG-19 (left) and ResNet-50 (right) in different case studies (tasks). The x-axis in the top plot refers to the number of retained filters, while the x-axis in the bottom plot is the decrease of probability on the original-DNN predicted class. It shows that NeuroChains can usually find very small sub-networks and, meanwhile, preserve the original pre-trained DNN's outputs. 45

3.2 Barplot of the faithfulness (filter score vs. fidelity degeneration caused by removing the filter) of 1,500 sub-networks, each extracted by NeuroChains on 20 uniform samples randomly drawn from two classes for VGG-19 (left) and ResNet-50 (right). The x-axis refers to the interval of filter scores, while the y-axis denotes the decrease of the sub-network's probability on the original DNN predicted class after removing a filter from the sub-network. It shows that the sub-networks suffer from more degeneration if a filter with a higher score is removed. Hence, the scores faithfully reflect the importance of filters in explaining the original pre-trained DNNs. 46

3.3 Left: Histogram of the quotient metric in Eq. (3.9) computed over all the 1,500×20 samples. Right: Faithfulness of NeuroChains in terms of the quotient's sign. 47

3.4 Comparison of different pruning methods on the capability of preserving the original network's output distribution (smaller KL divergence means better preservation) over 1,500×20 uniform samples. 49

3.5 Inference chain by NeuroChains for ResNet-50 (pre-trained on ImageNet) when applied to 20 test images of "strawberry" and "dalmatian". The sub-network retains only 13/67 layers and 77/26,560 filters of the ResNet-50. Refer to Section 3.3.4 for a detailed analysis of extracted chains. 50

3.6 Inference chain by NeuroChains for VGG-19 when applied to images of "kangaroo" and "banana". The sub-network retains only 8/16 layers and 116/4,480 filters of the VGG-19. The scores for selected filters are represented using the colormap on the top-right. Refer to Section 3.3.4 for a detailed analysis of extracted chains. 52

3.7 Inference chain by NeuroChains for VGG-19 when applied to images of "Castle" and "Stone Wall" which confuse DNN models and may be misclassified. The sub-network retains only 10/16 layers and 121/4,480 filters of the VGG-19. Refer to Section 3.3.4 for a detailed analysis of extracted chains. 54

3.8 Inference chain by NeuroChains for VGG-19 when applied to images of 'orangutan", "tram" and "broccoli". Its sub-network retains only 8/16 layers and 158/4,480 filters of the VGG-19. 56

3.9 A case study of applying NeuroChains to vision transformer (ViT). Top: the retained patches (red squares) in each block. Middle: attribution of the class token to the original input patches. Bottom: attribution of the patch token (retained in the last block) to the original input patches. 59

3.10 Comparison between the number of filters in sub-networks extracted with or without layer pruning on VGG-19 (left) and ResNet-50 (right). 59

3.11 Inference chain by NeuroChains for VGG-19 when applied to tell apart images of "eagle" and other randomly selected samples. The sub-network retains only 7/16 layers and 110/4,480 filters from the original VGG-19. 61

3.12 The validation-set accuracy of sub-networks vs. the number of training samples per class in each sub-task. 62

4.1 IoU of layers in ResNet between tasks with different similarities. 71

4.2 IoU of blocks in ViT between tasks with different similarities. 72

4.3 The example of majority voting in MVP. Each similar neighbour task of the target task votes for filters that are reserved by its pruned model. Then, softmax is applied to the votes of all filters in layer-ℓ and filters with more votes have a higher probability to be selected by the target task. 74
4.4 **(a)** Comparison between MVP and conventional pruning methods with different pruning iterations on different architectures. For both ResNet-18 and ViT, MVP converges much faster in a small number of iterations than conventional pruning methods. **(b)** Comparison between LEEP score and Wordnet similarity for MVP with different pruning iterations. From similarity groups 1 to 5, the similarities between tasks decrease. For both similarity metrics, more similar tasks get better performance. LEEP score has a better ability to measure similarities between tasks than Wordnet similarity. 80
4.5 **(a)** IoU of layers in ResNet-18 between tasks whose pruned models are extracted by IFP (Taylor Pruning) and more similar tasks also share more filters, especially in deeper layers. **(b)** Results of applying MVP to pruned models from Activation Pruning and Taylor Pruning over different numbers of neighbours. MVP(neighbour number ≥ 2) can improve transfer learning performance (neighbour number = 1) by a large margin when applied to pruned models extracted by different pruning methods. 82
4.6 Comparison of IoU between MVP and random pruning. 83

5.1 (a-e) The training dynamics of different metrics for different groups of parameters when applying SGD in CL to train three types of deep neural networks; The training dynamics of other scenarios in ResNet-18: (e) on a non-standard dataset; (g) using a different CL method with a different buffer size; (h) in the domain-IL setting. Note the y-axis is of logarithmic scale. 90
5.2 Comparison between the distribution of filters' training dynamics within a task and that cross tasks in different convolutional layers of VGG-11. When tasks shift, for all layers, only a small part of the filters in each layer experience more changes. 93
5.3 Comparison of SGD, replay-based method, FPF and k-FPF. SGD trains tasks sequentially without replay. Replay-based methods train models on buffered and current data simultaneously. FPF finetunes the most sensitive parameters for a few iterations using buffered data at the end of arbitrary CL methods. k-FPF periodically (regardless of task boundaries) applies FPF for k times during training. 94

5.4 Comparison of FLOPs and accuracy between FPF, k-FPF and SOTA methods. **FPF improves all CL methods by a large margin without notably extra computation. k-FPF consumes much less computation but achieves comparable performance as FPF**. 101

5.5 Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show **both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs**. 102

5.6 Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show both FPF and k-FPF only require finetuning a small portion of task-specific parameters. k-FPF halves FPF's FLOPs. **Different from the results of k-FPF in class-IL, in Seq-PACS, since the output classes for different tasks are always the same, the last FC layer will not have a large bias on particular classes. Only finetuning BN or CONV1 layers for k-FPF can get comparable performance with ER.** Similar to class-IL, since experience replay is not allowed during the training of CL method SGD, a few more parameters are required to be finetuned by k-FPF to get comparable performance with FPF (about 24.92% of all parameters). 103

5.7 **Top**: Trade-off between FLOPs and accuracy for k-FPF with different k and τ (the SGD steps between two consecutive FPF)(a). By increasing the finetuning steps per FPF, the accuracy quickly saturates. The best trade-off is highlighted at the top-left corner when $k = 9(\tau = 1,500)$. **Bottom**: Comparison between ER and FPF+ER finetuning different parameters with different buffer sizes(b) and numbers of epochs per task(c). In all scenarios, FPF can significantly improve ER by only finetuning BN+FC. 105

List of Tables

Table **Page**

3.1 Information of pre-trained CNNs in this work. 43
3.2 Information of pre-trained ViT in this work. 44

4.1 Comparison between MVP and baseline methods on CNN. The '-SSL' behind each method means applying this method to pruned models extracted from self-supervised pre-trained models. **Bold** and **Bold gray** mark the best and second best accuracy, respectively. 77
4.2 Comparison between MVP and baseline methods on ViT. **Bold** and **Bold gray** mark the best and second best accuracy 78
4.3 Accuracy of MVP on unseen tasks. 79
4.4 Accuracy of MVP on fine-grained tasks. 79
4.5 Results on sub-tasks of different sizes for CIFAR-100. 80

5.1 Test accuracy (%) of baselines, FPF and k-FPF. "-" indicates that the algorithm is not applicable to the setting. k-FPF-KD applies an additional knowledge distillation loss to the objective of k-FPF-CE. **Bold** and **Bold gray** mark the best and second best accuracy. 100
5.2 The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PathMNIST. **Bold** and **Bold gray** mark the best and second best accuracy. 106
5.3 The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PACS. **Bold** and **Bold gray** mark the best and second best accuracy. 106
5.4 Comparison of accuracy and FLOPs between FPF and FPF-ALL(finetuning all parameters). **Bold** and **Bold gray** mark the best and second best accuracy. 107
5.5 Classification results for CL benchmarks and FPF on MLP and VGG-11. **Bold** and **Bold gray** mark the best and second best accuracy. 108

5.6 Comparison of accuracy and FLOPs of k-FPF with the original results in [1] which has a large number of training epochs for each task. 108

CHAPTER 1

INTRODUCTION

1.1 Background

Deep neural networks (DNNs) have brought unprecedented progress and advancements in many fields. These models, trained on specific datasets, leverage their ability to learn complex patterns from vast amounts of data. However, these networks continue to grow in complexity and are computationally demanding, which results in the challenges of trustworthiness and efficiency for deep models.

While DNNs have demonstrated remarkable accuracy in various domains, ensuring their trustworthiness remains a critical concern. Trustworthiness refers to the interpretability, reliability, and fairness of deep models. The lack of transparency and interpretability in DNNs poses a significant challenge in understanding their reasoning processes, which raises questions about the reliability and fairness of models, especially in critical domains like healthcare, finance, and criminal justice. Efficiency is another crucial aspect that demands attention in the deployment of DNNs. In Fig. 1.1, we illustrate the development of training computation and model size for well-known DNNs [2]. In recent years, the training FLOPs of prominent DNNs have experienced exponential growth.

This is primarily due to the increasing sizes of the neural net architecture and training datasets. It is now common for large-scale models to contain millions or billions of parameters, rendering them impractical for many real-world applications. Reducing the computational cost of deep models is crucial for their deployment on resource-constrained devices and speeds up their training and inference processes.

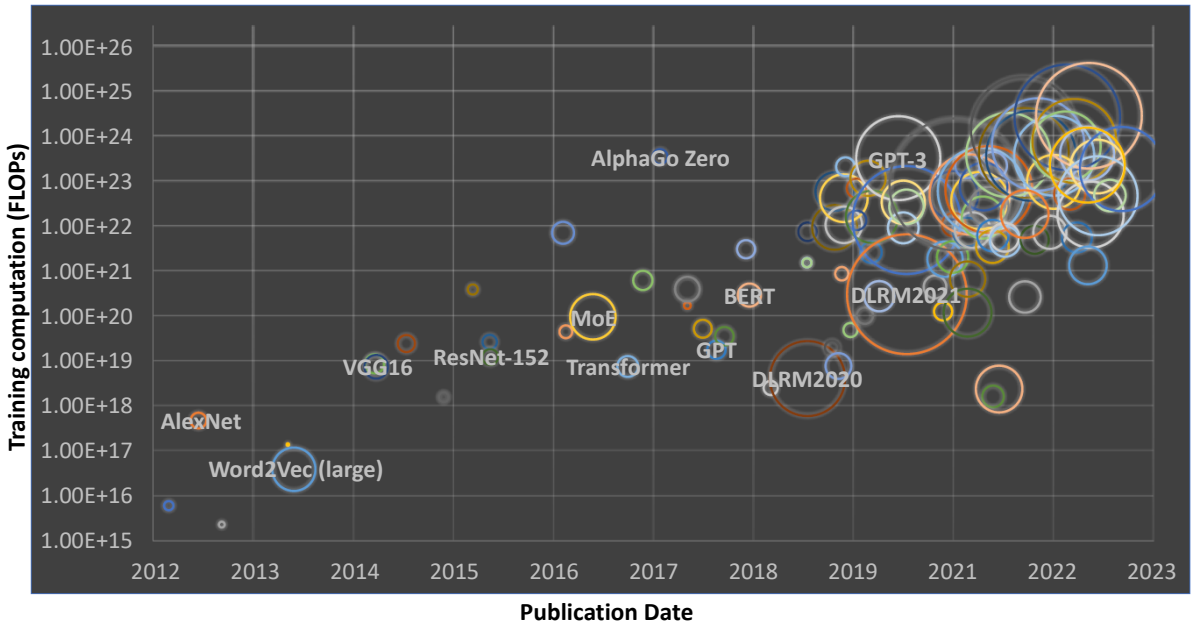


Figure 1.1: The development of popular DNNs in the past few years. Each circle represents a deep neural network. The x-axis indicates the publication date, while the y-axis refers to the training FLOPs (floating-point operations per second). The size of each circle corresponds to the number of parameters in the network.

In real-world scenarios, tasks often exhibit high correlations and contain a wealth of shared information. For example, in computer vision, datasets of different tasks may feature similar objects and patterns within image samples; in natural language processing (NLP), tasks like text summarization and question answering both necessitate a profound comprehension of the underlying meaning and context of the text. There is a lot of similar knowledge in the training data for these tasks.

Knowledge sharing between these tasks offers multiple advantages. Firstly, it promotes trustworthiness by reducing task-specific biases and improving the fairness and

reliability of the model's predictions across different domains and scenarios. Secondly, it enhances efficiency by leveraging the learned representations from similar tasks, reducing the need for extensive training from scratch. Lastly, it improves performance across tasks by harnessing the diverse knowledge embedded within different domains. Therefore, knowledge sharing has been widely studied to improve the effectiveness of DNNs.

1.1.1 Knowledge Sharing of Interpretable Pre-trained Models

Pre-trained models, which are deep neural network architectures trained on large-scale datasets to learn generalizable features and representations, have revolutionized deep learning by enabling knowledge sharing across various tasks. Improving the interpretability of pre-trained models can further facilitate knowledge sharing and make the model trustworthy. When models are more interpretable, the internal decision-making processes become more transparent, enabling a deeper understanding of the behavior of the model and helping to share knowledge with downstream tasks.

Transfer learning is a prominent approach that leverages pre-trained models to facilitate knowledge sharing, which has proven to be highly effective, especially for tasks with scarce data. It is based on the idea that knowledge learned from one task can be applied to the learning of a different but related task [3, 4]. This is also the underlying principle behind many transfer learning methods. For instance, finetuning [5, 6] refers to the method in which the pre-trained model of a previous task is finetuned on a new task by updating the model parameters with a smaller learning rate while another method [7] uses a pre-trained model as a feature extractor, where the output of one of the layers is used as input for another model trained on the new task. These approaches are much faster, require significantly fewer training samples, and consume fewer computing resources than training the new task from scratch. But, it can be challenging to interpret the acquired features and representations when employing DNNs on various tasks.

This can make it difficult to comprehend how the model makes its predictions and sometimes result in a negative transfer, in which knowledge acquired for one task hinders performance on another.

There are several factors that can affect the success of knowledge transfer from pre-trained models to downstream tasks. 1) The choice of data representation in the neural network plays a crucial role. Low-level features in the pre-trained model are generally applicable across a wide range of tasks, making them suitable for knowledge sharing, while high-level features represent complex and abstract concepts, which are often more task-specific. 2) The data distribution in the pre-training and downstream tasks is also important. A pre-trained model trained on a dataset of natural images is more likely to transfer its knowledge effectively to a downstream task involving similar natural images. However, the effectiveness of the knowledge transfer to the task involving a different type of data might be reduced due to the differences in data distribution.

Model interpretability can indeed help us understand these factors and facilitate knowledge sharing between tasks. For instance, by interpreting the model, we can understand the features and representations each filter, layer, and block has acquired and identify commonalities between the pre-training and downstream tasks. This helps us evaluate the suitability of the pre-trained model for the target task. Furthermore, understanding the reasoning process of pre-trained models can help us identify critical parameters for the given task. This knowledge aids in accelerating the training and inference processes of the model, leading to improved performance of this target task.

Numerous approaches have been proposed for interpreting pre-trained models and understanding the behavior of DNNs when applied in transfer learning. A line of work visualizes activations, weights, or learned features of DNNs. DeconvNet [8] maps featuremaps back to the input pixel space, identifying input patterns that activate individual filters at each layer in the network. Activation Maximization [9] visualize each learned

feature by generating an optimized input sample that maximizes the activation of the corresponding filter. Arefeen et al. [10] propose a method to identify the most significant layers from a pre-trained model for transfer learning while Kim et al. [11] define explainable features in the deep model and use them to train a model for a target task. This kind of method can help researchers understand what pattern each filter or layer in the pre-trained model has learned.

The attribution methods attribute network output to each input pixel. Layer-wise Relevance Propagation (LRP) [12] assigns the relevance score to each neuron in the output layer and then propagates these scores back through the network in a layer-wise manner until reaching the input layer. Simonyan et al. [13] compute the gradient of the output class score with respect to the input pixels and highlight the input patterns with the highest absolute gradient values. Class Activation Map [14], which is widely applied in transfer learning [15–17], involves the linear summation of visual patterns across various spatial locations within a network. These methods can identify the most critical input patterns for the final prediction of the network. Some other methods train a simpler, more interpretable model to approximate the behavior of the complex deep learning model. LIME [18] explains individual predictions of any machine learning model by approximating it with a locally linear model. The behavior and the reasoning process of these surrogate models are used to explain the original model. For instance, LIME is used by Ahsan et al. [19] to explain the prediction of transferred disease diagnosis models.

By implementing these methods, we can discern the patterns and features represented by each filter or layer in the pre-trained model and transferred model, and pinpoint the regions in the input samples that contribute to the network's final predictions. Nevertheless, to better share knowledge with downstream tasks, we need further gain a deeper understanding of the relationship among different filters or layers. The identification of critical parameters for given tasks is also crucial for effective knowledge transfer.

1.1.2 Improve DNN's Efficiency by Knowledge Sharing

Enhancing the efficiency of DNNs is important for their widespread adoption and practical deployment across various scenarios. Decreasing training time is one strategy for increasing the efficiency of DNNs. Rather than random initialization, transfer learning usually employs a pre-trained model learned from a previous related task. This initialized starting point is helpful to the new task, as it already includes valuable features and representations acquired from the previous task. Consequently, the model converges faster to a solution for the new task, leading to reduced training time [6, 20].

Using fewer parameters is an alternative approach to strengthen the efficiency of DNNs. Multi-task learning (MTL), where multiple tasks are learned concurrently using a single model [21], is another method to share knowledge between tasks. MTL can further reduce the number of parameters for training multiple tasks. When using MTL, task-specific heads are implemented for each task, which enables the model to learn task-specific features while still taking advantage of the shared lower-level features to improve the model's generalization performance. Typically, this architecture usually requires fewer parameters compared to training independent models for each task [22].

While the aforementioned methods have indeed increased neural network efficiency through knowledge sharing between tasks, during the testing phase, both transfer learning and MTL usually use the entire model to make predictions for each task, demanding substantial computational resources and time during model inference, especially when dealing with large models or a high number of tasks. This presents a significant challenge when deploying these models in real-world applications, especially on devices with limited computational resources.

Model pruning is another method to reduce the number of parameters of DNNs. It can reduce the model size and computational requirements by removing redundant and unimportant parameters in neural networks while maintaining its performance on a

given task. Combining model pruning and knowledge sharing, DNNs can achieve higher efficiency through reduced computational requirements while maintaining trustworthiness through improved generalization and reliability.

Pruning methods can be broadly categorized into structured and unstructured pruning based on the granularity of pruning. Structured pruning [23–25] targets coarser-grained units, such as filters, layers, or blocks in the neural network, reaching a more regular structure. These metrics can be well-suited for deep learning frameworks like PyTorch and TensorFlow to speed up the inference of the network. In MTL, Chen et al. [26] propose multi-task pruning to remove redundant filters in the model for both classification and segmentation tasks. Given the models of different tasks, Cheng et al. [27] remove unimportant filters from multiple models of different tasks, merge similar filters, and ultimately generate a single model. This method enables knowledge sharing across tasks and reduces the overall model size.

In contrast, unstructured pruning [28–30] focuses on pruning individual weights or connections within the network. Gordon et al. [31] study the effects of weight pruning on transfer learning. The findings reveal that low levels of pruning with the pruning ratio of $30-40\%$ will not affect pre-training loss or transfer to downstream tasks at all. This research proves that unstructured pruning allows for a higher pruning ratio and often leads to better performance in terms of model compression. However, unstructured pruning results in irregular structures that pose challenges to efficient storage and inference.

Model pruning can also be classified into manual and automatic pruning based on the pruning strategy employed. Manual pruning [24, 32, 33] typically involves establishing a criterion to evaluate the importance of each parameter/filter in the model and subsequently removing the less important ones. When transferring the model to new tasks, Gordon et al.[31] use the L1 norm of parameters to measure their importance, while Sanh et al.[34] calculate the product of weights and corresponding gradients to assess the

importance of weights. The latter approach often achieves better performance, indicating that a good importance criterion requires expert knowledge and a deep understanding of the model's structure.

Automatic pruning [32, 35, 36] assigns a trainable score to each prunable unit in the model. These scores are always optimized through the training process with a regularization term. Ultimately, parameters/filters with higher scores are retained, while those with lower scores are removed. Automatic pruning is widely used when combined with knowledge sharing [37–39] since it is difficult to establish a criterion to measure the importance of parameters for multiple tasks or unseen tasks. Automatic pruning has the advantage of being more adaptive as the pruning process is guided by the optimization of the model itself.

Model pruning is extensively utilized in practical applications to decrease the computational expense of training deep models and accelerate inference. When integrated with knowledge sharing, existing methods identify critical parameters to share among different tasks, resulting in a pruned model suitable for a few tasks. However, when new tasks come, it becomes necessary to start the pruning process from the beginning. To address this, we can leverage knowledge sharing to speed up the pruning process of new tasks by reusing both task-agnostic and task-specific knowledge related to the new task in existing pruned models.

1.1.3 Sharing Knowledge towards Fairness across Sequential Tasks

In real-world scenarios, tasks often arise sequentially rather than concurrently, and the data distribution can shift across tasks. For instance, in the healthcare domain, new diseases may emerge over time, requiring diagnostic models to learn from new medical cases and integrate this knowledge into their existing understanding. Similarly, the

driving environment in autonomous vehicles is constantly changing, and the vehicle needs to learn and adapt in real-time to ensure safe navigation.

In this scenario, traditional transfer learning and MTL methods may not be optimal. DNNs may forget previously learned knowledge and exhibit significant bias towards recent tasks, compromising their overall performance and undermining their reliability. Continual learning (CL) is proposed to address these issues and promote fairness across sequential tasks. Various CL methods utilize knowledge sharing to adapt the model to new tasks without forgetting the knowledge of previous tasks, which fosters a more balanced and trustworthy learning process.

Replay-based methods [40, 41] interleave samples from previous tasks with new data during training to share the knowledge across tasks so that the model will not forget the previous tasks while adapting to new ones. Knowledge distillation [42, 43] is also applied to remember previous knowledge. Specifically, the previously learned model serves as the teacher, while the new model learning the current task acts as the student. The distillation loss prompts the student model to generate similar outputs to the teacher model for the replay data, effectively imparting the knowledge of previous tasks to the new model.

Regularization-based methods [44–46] in CL address the problem of strong bias by imposing constraints on the model's parameters during the learning process. These methods first assess the importance of parameters during the learning of each task and subsequently employ a regularization term to penalize changes to the model parameters that were critical for previously learned tasks. These methods allow the model to preserve the knowledge of previous tasks, thereby facilitating knowledge sharing.

To avoid bias and make deep models trustworthy, another kind of CL method involves expanding or adapting the model's architecture as new tasks emerge. Some approaches [47, 48] introduce new parameters or modules for each new task and fine-

tune them together alongside existing parameters. These methods reuse the previously learned parameters to share prior knowledge, while the new parameters for each task learn the task-specific knowledge. Alternatively, some other methods [49, 50] selectively activate certain parts of the network for each new task without adding individual parameters. These approaches introduce task-specific masks or pathways in the model for each task. For different tasks, the masks or pathways may share certain parameters, allowing the knowledge of different tasks to be shared among them. By freezing the parameters of previous tasks during the training of new tasks, the final CL model avoids bias and forgetting, ensuring a more reliable and fair deep model.

Knowledge sharing has become a crucial approach to CL. By transferring information between tasks, the model not only retains a comprehensive memory of previous tasks but also leverages the knowledge from prior tasks to accelerate convergence for new tasks, which enhances the model's generalization capabilities and leads to fairness across sequential tasks.

1.2 Research Problems

As previously discussed, knowledge sharing between tasks can drive progress in diverse machine learning domains, and existing methods have demonstrated promising results. Nevertheless, these approaches still have certain limitations in terms of the trustworthiness and efficiency of DNNs, leaving ample room for improvement and refinement.

In this thesis, we point out three critical problems in DNNs as illustrated in Fig. 1.2 and propose to solve them by combining knowledge sharing with parameter analysis. To deepen our understanding of the relationships between various tasks, we initially construct a pool of pruned models tailored to different downstream tasks. Each pruned model can pinpoint crucial parameters in the pre-trained model relevant to its respective task. Subsequently, we enhance the trustworthiness of DNNs from three perspectives:

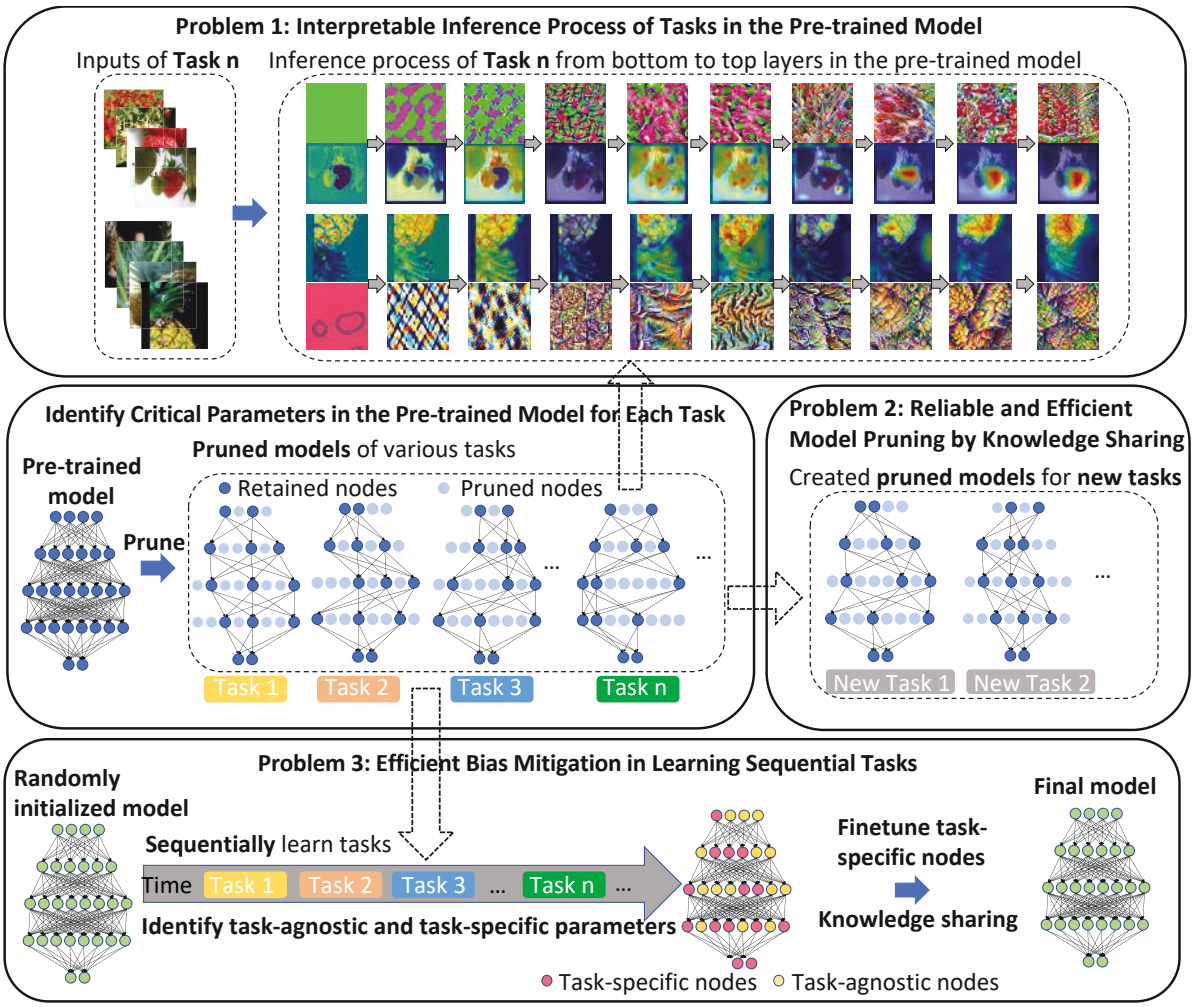


Figure 1.2: Three critical problems studied in this thesis.

interpretability, reliability, and fairness. Through parameter analysis, we distinguish between task-specific and task-agnostic parameters in the model, unveiling the inference process of the deep model for the given task, as shown in Problem 1, thereby improving its interpretability. With a clearer interpretation of each task, we gain better insight into the differences and similarities among tasks, enabling us to further share knowledge across these tasks. In Problem 2, we establish a reliable deep model by sharing knowledge from similar tasks. Sharing task-agnostic parameters and fine-tuning task-specific parameters in deep models, as discussed in Problem 3, can enhance the fairness of DNNs across various tasks. Beyond trustworthiness, our approach significantly boosts the efficiency of

training DNNs. Through knowledge sharing, we initiate training from a well-initialized network, rather than starting from scratch, as in Problem 2, resulting in substantial computational savings and improved model generalization. Additionally, as studied in Problem 3, frequent data replay from previous tasks is no longer necessary, greatly reducing the computational load of continual learning.

In our research, to study different problems and evaluate our proposed methods, we mainly focus on various n-classification tasks randomly sampled from different datasets like CIFAR and ImageNet, and we encourage DNNs to understand and share learned representations across various tasks.

1.2.1 Interpretable Inference Process of Tasks in the Pre-trained Model

Model interpretability is a critical component of building trustworthy DNNs, as it facilitates the development of deep models that exhibit transparency, reliability, and fairness. Pre-trained models have become pervasive in real-world applications, as they can be finetuned or adapted to share knowledge with downstream tasks. Improving the interpretability of pre-trained models can contribute to a deeper understanding of the acquired knowledge and its relationship to downstream tasks, as well as facilitate more effective knowledge sharing.

However, existing interpretation methods mainly explain neural networks from the perspectives of individual input samples, filters/nodes, or activations, while neglecting the specific explanation of the pre-trained model in relation to different tasks. To share the knowledge between tasks, we need to identify task-specific parameters in the pre-trained model for each downstream task and understand how each task obtains its final prediction step by step from the input to the output in the DNN. Only by solving these problems, we can further improve the performance of downstream tasks based on the

pre-trained model.

In this thesis, we propose a human-readable model interpretability method that unveils the inner mechanism of the pre-trained model and illustrates its inference process for a given task, layer by layer. To ensure that the explanation is human-readable, the reasoning process of each task must be precisely localized, and the number of critical parameters should be small enough. We employ model pruning to extract a sub-network from the pre-trained model for a given task. To minimize the size of the sub-network, not only the filters but also the irrelevant layers and blocks within the pre-trained model are removed. The parameters in the pre-trained model are fixed during pruning to preserve their original meaning. After generating the sub-network, we visualize the filters and stitch them layer by layer to demonstrate the model's reasoning process.

The proposed method initially identifies essential task-specific parameters in the pre-trained model and then generates a high-performing sparse model for each task. The retained sub-network provides a human-readable NeuroChain for each task to explain its inference process in a layer-wise manner, which deepens our understanding of the inner mechanisms of DNNs. Furthermore, as the functions of filters and layers within the model are unveiled, these representations can be efficiently adapted to other related tasks to accelerate their training process and improve their performance.

1.2.2 Reliable and Efficient Model Pruning by Knowledge Sharing

As models grow increasingly large and datasets continue to expand in real-world applications, improving the training and inference efficiency of DNNs has become an unavoidable challenge. However, many existing pruning methods require pruning the pre-trained model from scratch for each individual task, which is computationally demanding and time-consuming, especially when dealing with a large number of downstream tasks and

devices. Additionally, when pruning an existing model for a new task, it is challenging to ensure the trustworthiness of the pruned model without a clear understanding of the relationship between the source and target tasks.

In this thesis, to solve the problem of reliability and efficiency, we propose to combine model pruning with knowledge sharing. As discussed in the previous section, model pruning can not only serve as an effective method for reducing the complexity and size of deep models, but it can also identify critical parameters in the model and extract an interpretable pruned model for a given task. We can share the knowledge from the pruned models of existing tasks to efficiently create reliable high-performing pruned models for related new tasks.

Specifically, we first create a pool of pruned models derived from pre-trained DNNs on various tasks as shown in Fig. 1.2. Each pruned model in the pool consists of critical parameters related to its respective task as identified in Problem 1. When a new task arises, we calculate the semantic similarities between the new task and existing tasks in the model pool. The pruned models for tasks with high similarities are selected to share their knowledge with the new task and collaboratively contribute to the creation of a reliable well-initialized pruned model, which can quickly adapt to the new task by finetuning for a few numbers of iterations.

Given that the selected tasks are all semantically similar to the new task, and the retained filters in the pruned model of similar tasks are specific and critical to their class labels, the initialized model created by the shared knowledge of similar tasks already contains a wealth of information about the new task. This knowledge can accelerate the convergence of the initialized model on the new task, and significantly shorten the process of model pruning. In addition, the diverse knowledge from various similar tasks improves the model's reliability and generalization ability, thereby enhancing its performance on the new task. The combination of model pruning and knowledge sharing

boosts the performance of DNNs in terms of accuracy and efficiency.

1.2.3 Efficient Bias Mitigation in Learning Sequential Tasks

In MTL, a trustworthy deep model always exhibits fair behavior across multiple tasks. However, when training sequential tasks, DNNs often struggle with a phenomenon known as catastrophic forgetting. This means they tend to lose previously learned information when adapting to new tasks, resulting in a strong bias towards the most recent tasks. In contrast, humans naturally exhibit the ability to learn continually and adapt to new information without forgetting prior knowledge. To address this issue, CL has been proposed to develop models capable of acquiring new knowledge while retaining previously learned information, thereby reducing bias and promoting fairness across tasks.

As discussed in Sec. 1.1.3, knowledge sharing has played a crucial role in various CL methods, as it helps mitigate catastrophic forgetting and enables a more efficient transfer of knowledge between tasks. However, existing CL methods have their own drawbacks and limitations. High-performing replay-based methods often necessitate considerable computational resources to learn replay data in each training iteration. Owing to the constraint of parameters in the network, regularization-based methods struggle to learn new tasks effectively and often demonstrate poor performance. Moreover, these methods, along with replay-based methods, tend to exhibit significant bias towards recent tasks. Methods with dynamic architectures consistently require additional storage for individual modules associated with each task, leading to an increase in memory demand, which can result in less efficient outcomes.

As shown in Fig. 1.2, each task has its own critical pathway in the neural network to perform inference. These pathways may have some common parameters that contain task-agnostic knowledge, which can be shared across various tasks. The remaining

parameters are task-specific, and any changes to them could result in the forgetting of the corresponding task. In this thesis, we first conduct an empirical study to identify task-specific parameters in the model, which are sensitive to task shifts and result in bias across tasks We then finetune these task-specific parameters on the replay data to reduce bias and share the knowledge of previous tasks. Furthermore, we propose a novel method that replaces the every-step replay with periodical finetuning of sensitive parameters. This approach facilitates knowledge sharing of previous tasks while simultaneously significantly reducing the computational cost associated with replay-based methods.

Our proposed method identifies parameters sensitive to task shifts in various model architectures and demonstrates that merely finetuning these parameters on the replay data significantly reduces the bias and promotes fairness among sequential tasks. Our method eliminates the need for every-step replay in CL without sacrificing accuracy, saving at least half of the training computation cost.

1.3 Thesis Organization

This thesis conducts a thorough investigation into knowledge sharing in DNNs, organized into six chapters. Chapters 1 and 2 establish the foundation by setting up the research's groundwork and conducting an extensive literature review. Building on this, Chapters 3, 4, and 5 introduce methodologies addressing various research problems depicted in Fig. 1.2, delving into specific technical aspects. The thesis concludes with Chapter 6, offering a summary of the research and suggesting potential areas for further exploration. Following are detailed explanations for each chapter.

- Chapter 1: This chapter lays the foundation by introducing the background of knowledge sharing in various domains of DNNs, emphasizing its significance in improving the trustworthiness and efficiency of DNNs, and presenting three

research problems to be addressed in this study.

- Chapter 2: This chapter provides a comprehensive literature review of knowledge sharing methods and related work of proposed methodologies involved in the research.

- Chapter 3: This chapter proposes a novel model interpretability method to solve problem 1 in Fig. 1.2. It first applies model pruning to identify critical parameters in the pre-trained model for each task and then proposes an approach to reveal the reasoning process of the pre-trained model for each task. The layer-by-layer inference process of the model is visualized to explain its behavior.

- Chapter 4: This chapter aims to build a method to accelerate the process of model pruning and enhance the reliability of the pruned model, as depicted in problem 2 shown in Fig. 1.2. We empirically investigate the correlation between the pruned models of various tasks in the pre-trained model, and leverage the knowledge of similar tasks to rapidly construct a reliable, sparse, and high-performing model for any new task.

- Chapter 5: To address problem 3 in Fig. 1.2 and improve the fairness of DNNs, this chapter explores the stability and sensitivity of parameters in deep models when addressing sequential tasks. To mitigate the bias across tasks and enhance efficiency, the identified task-specific parameters in the model are finetuned on replay data during the training course of CL.

- Chapter 6: This chapter concludes the thesis, and points out the direction of future research in the field of knowledge sharing between tasks in DNNs.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we provide a comprehensive review of the existing literature related to knowledge sharing between tasks, including transfer learning and multi-task learning (MTL). We will also carefully summarize related works concerning model interpretability, model pruning, and continual learning, which are central to our research problems.

2.1 Knowledge Sharing between Tasks

2.1.1 Transfer Learning

Transfer learning aims to improve the efficacy of DNNs on a target task by leveraging knowledge from a related task [3, 51]. The hierarchical feature representations learned by DNNs exhibit generalizability across diverse tasks, thus contributing to the success of transfer learning [52]. In transfer learning, various methods can facilitate the sharing of different knowledge between tasks. In this section, we introduce transfer learning methods that share features and parameters, as well as recent work involving knowledge sharing in Vision Transformers (ViT).

Feature-based methods focus on transforming the original features of the source and target domains to create a new feature representation that aims to minimize the discrepancy between source and target domain feature distributions. These methods are often used to address the domain shift problem and perform well when the number of labeled data for the target domain is limited.

Maximum Mean Discrepancy (MMD)[53] is widely used in transfer learning to measure the difference between source and target domain feature distributions, which are usually represented by activations in a neural network's hidden layers. Many works propose methods[54–56] to reduce the MMD value during training to close the gap between domains. This process enables the acquisition of domain-invariant feature representations that are effective for both source and target domains. DaNN [55] first introduces MMD loss in deep networks to calculate the distance between the source and target domains. However, due to the shallow network employed, it is unable to effectively address the domain adaptation problem.

DDC [57] uses a pre-trained AlexNet and adds an adaptation layer before the classifier in the network, whose outputs are used to calculate the MMD value. The purpose of the adaptation layer is to evaluate the network's ability to differentiate between the source and target domains. If the adaptation layer's discrimination capability is poor, the authors infer that the features learned by the network are inadequate to distinguish data from the two domains, thereby facilitating the learning of domain-invariant feature representations.

Building on DDC, DAN [58] addresses two problems with DDC. First, they add adaptation layers to all three fully-connected layers of AlexNet because the features learned by these layers are more task-specific, and their representations can be transferred [55]. Second, the authors replace MMD with Multi-kernel MMD (MK-MMD)[59]. MK-MMD leverages multiple kernels instead of a single kernel, better capturing the differences be-

tween distributions by considering multiple feature spaces. Additionally, other methods to measure the distance between source and target domains have been proposed. Sun et al.[60] introduce a CORAL loss, which aligns the second-order statistics of the source and target distributions with a linear transformation.

Due to the success of generative adversarial networks (GAN), the idea of adversarial training has also been incorporated into transfer learning. Recent works [61–63] add a domain discriminator to the network to compete with the feature extractor. The feature extractor is trained to confuse the domain discriminator, while the domain discriminator is trained to distinguish between source and target domain features. During this process, the feature extractor is encouraged to learn domain-invariant features that are useful for both source and target domains.

Parameter-based methods [6, 64–67] are a category of transfer learning methods that leverage the learned parameters or weights of a pre-trained model to facilitate knowledge transfer between tasks.

In 2014, the transferability of individual layers in the DNNs is studied [6]. Yosinski et al. first train a deep neural work on the source task and then finetune each layer in the model on the target task to quantify the degree of generality or specificity of features acquired in that layer. The study reveals that features learned on the bottom layers (close to the input) tend to be general and applicable to multiple datasets and tasks.

Finetuning serves as an effective method to transfer knowledge across tasks, but the choice of finetuned layers varies depending on the task [65]. To solve this problem, several methods [65–67] are proposed to selectively finetune the parameters of different layers. Rozantsev et al. [65] a two-stream architecture, with one stream operating in the source domain and the other in the target domain. Some layers in the networks are shared by the source and target domains to transfer knowledge while weights of the non-shared layers will be finetuned. These non-shared weights are also encouraged to

remain close to being linear transformations of each other by introducing an additional regularization term. To select the layers for finetuning, the authors compare the MMD values between the learned source and target representations for different configurations, and the configuration that yields the smallest MMD value will be selected.

It is possible to employ an exhaustive search to select layers for finetuning in small-scale neural networks. However, as the scale of the models increases, this method becomes impractical. Guo et al. [67] introduce a lightweight policy network to determine whether each layer in the network should be finetuned. Since the policy network,Äòs output is discrete, the Gumbel Softmax sampling method is employed to train the policy network. Layers sampled and selected within the network will be finetuned, while those not chosen will be directly reused by the target task.

Transfer learning on ViT The previously discussed methods primarily focus on transferring knowledge across tasks in CNNs. However, when applying transfer learning to ViT, different strategies are employed. Chen et al. [68] develop a pre-trained image processing model based on the Transformer architecture, known as the Image Processing Transformer (IPT). This model can be transferred to low-level computer vision tasks such as denoising, super-resolution, and deraining. IPT model consists of heads, a transformer encoder, a transformer decoder, and tails. The transformer encoder and decoder in the model are shared by all tasks, while each pair of head and tail are carefully designed for each specific task. With only finetuning, IPT can achieve state-of-the-art performance on various low-level computer vision tasks. BEiT [69] and MAE [70] are pre-trained vision transformers that are trained by reconstructing the missing parts of input images. After pre-training the model, the encoder in the model can be shared with downstream tasks. Owing to the sufficient semantic region knowledge in the pre-trained model, the performance of finetuning achieves significant improvement.

In the context of unsupervised domain adaptation (UDA) problems, previous methods

have primarily applied CNNs to learn domain-invariant representations. Recently, a few studies [71, 72] have investigated the transferability of ViT for addressing UDA problems. Specifically, Xu et al.[71] leverage the cross-attention mechanism in transformers and employ a source-target branch to align the distribution of source and target domains. While Yang et al.[72] develop a transferability adaptation module that encourages ViT to simultaneously learn transferable and discriminative features for effective knowledge transfer. These studies highlight the potential of ViT in solving UDA problems.

2.1.2 Multi-task Learning

Multi-task learning (MTL) [21] is a widely-used approach for sharing knowledge between tasks. Unlike transfer learning, which primarily focuses on improving a target task using knowledge from a source task, MTL simultaneously learns multiple related tasks and aims to optimize the performance of all tasks jointly. Knowledge sharing methods in multi-task learning can generally be classified into two main approaches: learning shareable features and analyzing inherent relationships between tasks.

Shareable features This kind of method seeks to uncover similarities across tasks and learn shared representations. One of the earliest multi-task models is the multi-layer feedforward neural network [21], which shares the input layer and the hidden layer across all tasks while assigning each task its own output unit. The outputs of the hidden layer can be viewed as the common feature learned for all tasks. Owing to the powerful ability of DNNs to learn nonlinear features, this concept has been incorporated into deep learning for solving problems across diverse research fields [73–77].

Zhang et al. [73] apply MLT to facial landmark detection, investigating whether optimizing facial landmark detection along with related tasks could enhance detection performance. In their model, all convolutional layers and one fully-connected layer are shared, and the output from the fully-connected layer serves as the shared representa-

tion for all tasks. Subsequently, each task has its own loss function to optimize these shared parameters. Liu et al.[77] introduced a novel multi-task deep neural network model termed MT-DNN. The MT-DNN model learns four types of natural language understanding (NLU) tasks jointly based on a pre-trained BERT model[78]. During training, the lower layers, comprising the lexicon encoder and transformer encoder, are shared across all tasks, while additional task-specific layers generating task-specific representations are appended for each task. By harnessing the advantages of both BERT and MTL, MT-DNN attains state-of-the-art results on ten NLU tasks. The outcomes demonstrate that MTL can effectively leverage supervised data from numerous related tasks, benefiting from regularization effects and consequently rendering the learned representations generalizable across tasks.

Inspired by generative adversarial networks, adversarial learning has also been applied in MTL to develop general representations for all tasks [79, 80]. In the models of these methods, beyond the task-agnostic feature extraction network shared by all tasks and the task-specific classifier module for training each task, an additional domain discriminator is introduced to identify the domain of each data sample. Throughout the training process, the feature extraction network is optimized to deceive the domain discriminator, leading to increasingly general learned representations. Apart from sharing the same representations across all tasks, an alternative approach suggests that each task learns different yet related representations. Misra et al. [81] proposed a cross-stitch network that incorporates a linear combination of task-specific activations, enabling the model to optimally learn how to combine and share knowledge from various tasks. This method facilitates a more nuanced exchange of knowledge between tasks, leading to enhanced performance.

Task Relationships Analyzing the inherent relationships between tasks is an essential aspect of MTL, as it enables the model to more effectively utilize the knowledge

from various tasks.

Task clustering is a kind of method [82–84] that groups similar tasks together based on their similarities, allowing the model to learn shared representations for each cluster. Thrun et al. [82] introduced a task-clustering algorithm, which initially learns a distance metric for each task to measure the distance between any two samples in the corresponding task. The data of $Task_j$ is then applied to the distance metric of $Task_i$ to calculate the generalization accuracy of $Task_j$ on $Task_i$. The generalization accuracy between any two tasks is stored in a transfer matrix. Based on this transfer matrix, tasks are grouped into different clusters. When a new learning task arises, the proposed method first identifies the most related task cluster and selectively exploits knowledge from that task cluster only. In [85], the author proposed a method called flexible clustered multi-task learning, which learns the cluster structure by identifying the representative task in each cluster. Each task is described by multiple representative tasks with varying probabilities. The model parameters of each task and its representative task are encouraged to be similar. By leveraging these probabilities, tasks are not required to share knowledge to the same extent, allowing for more adaptable knowledge sharing among tasks.

In some methods, the correlation between tasks is regarded as prior knowledge. Based on the principle that more similar tasks have closer model parameters, research such as [86–88] leverages task similarities to design regularization terms, guiding the learning of multiple tasks. In most real-world applications, manually defining the relatedness between tasks can be challenging. As a result, some methods, like those proposed in [89–91], aim to learn task relations automatically. Long et al. [92] propose MRN which introduces the tensor normal distribution as the prior distribution over network parameters of fully connected layers to learn find-grained multilinear relationships of tasks.

Ma et al. [93] introduce the Multi-gate Mixture-of-Experts (MMoE) to learn model task relationships from data by sharing expert networks across all tasks. This approach

differs from the model in [21], where only one bottom is shared by all tasks. MMoE includes a group of bottom networks, each referred to as an expert. The authors employ a gating network for each task to assemble the experts with different softmax probabilities. The results of all experts are then fused and sent to task-specific networks. The gating networks for various tasks can learn different mixture patterns of experts, thereby capturing task relationships. Liu et al. [94] introduce attention mechanism into the MTL to study the relationship between tasks. The proposed framework comprises a main network designed to generate shared features for all tasks. Based on this, each task acquires useful features for itself from the main network through the task-specific soft-attention modules. In this manner, the network can simultaneously learn both the task-agnostic and task-specific representations.

Knowledge sharing in federated learning In addition to transfer learning and multi-task learning, knowledge sharing is a key component of federated learning (FL) [95, 96]. FL involves training a machine learning algorithm using multiple local datasets stored on local nodes without explicit data exchange. FL finds extensive application in sensitive domains such as economics [97], healthcare [98, 99], government [100, 101] and personal privacy [102], due to its ability to quickly learn client data while ensuring robust data privacy.

Personalized federated learning (PFL) jointly trains diverse local models through balancing between knowledge sharing across clients and model personalization per client [103, 104]. Chen et al. [105] enhance the knowledge-sharing process in PFL by utilizing graph-based structural information among clients. Clustered federated learning (CFL)[106, 107] groups similar clients into clusters and subsequently learns a shared model for all clients within a cluster. This approach better captures the heterogeneity of data distributions across clients and enhances knowledge sharing among similar clients. To enhance the robustness of FL algorithms against the diversity of data distributions,

Tan et al.[108] facilitate knowledge sharing by having clients and servers communicate abstract class prototypes instead of gradients.

2.2 Model Interpretability

Interpretable deep neural network methods can be primarily classified into two categories: those that assess the importance of each input feature for a single sample and those that explain the behavior of networks.

Interpretability of input sample Approaches in the first category typically rely on a certain back-propagation from the DNN's output to derive an importance score for each input feature or hidden node. Deconvolution is the first method developed to visualize the learned features across different layers of networks, which performs a three-step operation of unpooling, rectification, and filtering to map each featuremap to the original input sample.

Some earlier works are based on back-propagated gradients, like back-propagation [13], and guided back-propagation [109]. Sundararajan et al. [110] propose an (approximate) calculation of the integral of the gradients along a path between a baseline point and the input sample, ensuring sensitivity and implementation invariance that some previous methods lacked. In [111], the authors find that adding some noise to the original image can reduce the noise in sensitivity maps. They take an image as input and sample similar images by adding different noises to the image, then average the resulting sensitivity maps for each sampled image. This method is also complementary to existing gradient-based methods.

More recent methods propose novel back-propagation rules to directly derive the attribution scores of neurons from output to input, e.g., DeepLIFT [112], deep Taylor decomposition [113], and layer-wise relevance propagation (LRP) [12]. Among these, LRP is the representative method. LRP redistributes the relevance of a neuron in a given

layer to the neurons in the previous layer according to specific propagation rules. These rules intended to ensure that the relevance is maintained throughout the network, i.e., that the sum of the relevance scores at the input layer equals the relevance score at the output layer. In LRP, each pixel's relevance score indicates its contribution to the output. A negative relevance score indicates a negative contribution.

Interpretability of behavior Methods in the second class treat DNNs as black boxes and seek simple models to explain how the DNN's output changes in a local region. For example, perturbation-based methods [114] add masks to different parts of the input to evaluate how the perturbations change the output, reflecting the importance of different features in the input sample. Zeiler et al. [8] covered different parts of the input with a gray square, which results in varying prediction probabilities for the true class of the output, showing the importance of each part to a certain class. In contrast, Zintgraf et al. [115] replaced each patch in the input image with the surrounding patch due to the observation that a pixel in an image has a strong dependence on its neighborhood. The effect of each patch on the final output will be calculated by averaging the results of using various surrounding patches. Additionally, the authors extend this method to study the relationship between any two nodes in the DNN.

Some methods provide insights into the behavior of a machine learning model for specific instances or a small region within the input space. In LIME, Ribeiro et al. [18] trained a sparse linear model on noisy input-output pairs as a local surrogate approximating the original pre-trained DNNs. The sparse weights in this linear model help explain the importance of input features for a specific instance, providing an interpretable explanation of the model's behavior. Zhang et al. [116] propose an approach that trains a decision tree to quantitatively interpret the logic of a convolutional neural network (CNN) at the semantic level. In this method, each node in the decision tree represents a specific decision pattern, and the tree organizes all these patterns in a hierarchical,

coarse-to-fine manner. Given an input sample, its inference path through the decision tree can provide insights into the model's behavior. By analyzing the nodes traversed during the inference, researchers can identify which parts of the object or filter are used for prediction and assess the contribution these parts make to the final prediction.

While the previously mentioned methods focus on explaining the behavior of DNNs for individual input samples, they do not fully capture the reasoning process of the model. In Chapter. 3, we propose NeuroChain, which addresses this limitation by providing a more comprehensive explanation of DNNs for a specific (sub)task and its data. NeuroChain differs from the other methods in that it explains not only the model's behavior for a single input but also how the model integrates information from important filters or neurons in a step-by-step manner. This approach offers a more complete understanding of the model's reasoning process, enabling researchers to better interpret and analyze the inner workings of DNNs across different tasks and datasets.

2.3 Model Pruning

In this section, we will first provide an overview of representative model pruning methods for CNNs and ViT. Then, we will discuss several methods that incorporate meta-learning techniques to prune models more effectively.

Model pruning on CNNs Structural pruning methods remove whole layers, channels, filters, or neurons in CNNs based on a certain criterion that measures the importance of the prunable units. Li et al.[24] prune filters by calculating the smallest sum of absolute values of their parameters, while Lin et al.[32] use the second-order Taylor expansion of the loss to determine which filters to prune. Methods like HRank [33] and APoZ [117] evaluate the importance of each filter based on their corresponding activation maps. Luo et al. [118] develop an approach that applies pruning to two adjacent convolution layers at each time to take the dependency between the two layers into account. Yu

et al. [119] evaluate filter importance by considering the impact of each filter on the final output.

Another group of methods [32, 35] maintains a mask for filters or nodes in the model to automatically eliminate redundant parameters. To encourage sparsity in the pruned network, different regularization techniques have been used, such as L_0[120], L_1[121], and L_2[28] regularization. Polarization regularization[122] is another approach that shrinks some nodes towards zero while simultaneously strengthening others, thus preserving important nodes and promoting a more efficient pruned network.

Unstructured pruning removes individual weights or connections rather than the entire structures, such as layers or filters, in DNNs. Unstructured pruning works by assigning an importance score to each weight in the network, often based on the magnitude or another criterion. The lottery ticket hypothesis [123] is a representative method in weight pruning, which posits that there exist sub-networks (i.e., winning tickets) that can achieve comparable generalization performance to the original pre-trained DNN when re-trained. DLTH [124] is a variant of LTH and also starts from a well-designed sub-network. It claims that randomly extracted subnetworks from a randomly initialized dense network can be transformed into a well-performing sub-network that can achieve admirable performance compared to LTH. To provide theoretical support for the lottery ticket hypothesis, a number of recent works [125–128] have attempted to prove its validity.

Model pruning on ViT ViT has been widely used in computer vision and achieved state-of-the-art (SOTA) performance in numerous tasks. Recent works [36, 129, 130] automatically prune the unimportant heads, nodes, and blocks in ViT by introducing a learnable parameter to each printable unit to reduce the computational complexity of the network. Zhu et al.[129] suggest pruning ViT in a manner similar to CNNs[121]. They train ViT with sparsity regularization, prune unimportant dimensions in the linear

projection module, and then finetune the model to recover the performance. Yu et al. [36] propose a method called WDPruning, which targets both the width and depth dimensions of ViT. To reduce the width, the authors use a learnable saliency score to prune the multi-head self-attention module and linear projection module. To reduce depth, they introduce several shallow classifiers that utilize intermediate information from the transformer block. The images can be classified by shallow classifiers and all subsequent transformer blocks will be removed.

In a Vision Transformer (ViT), an image is divided into non-overlapping patches. Removing the less important patches in the input can lower the overall computational complexity of the model by decreasing the number of tokens fed into the transformer layers. Goyal et al. [131] propose using attention scores between different tokens in each layer to measure their importance, as tokens with small attention scores usually have little relevance to others. They then introduce a word-vector extraction layer to dynamically prune tokens with low-importance scores. Initially proposed for pruning BERT, this method can be effectively adapted to ViT. Tang et al. [132] evaluate the importance of each patch in maintaining the original final results and discard less useful patches using a top-down paradigm. Patches retained by the top layers are also kept by the bottom layers. HVT [133] is a CNN-like method that shortens the patch sequence by max-pooling, which compresses information of multiple patches into a small number of patches.

Meta-pruning Several recent studies aim to learn meta(prior) knowledge that can improve model pruning. MetaPruning [134] trains a weight-generation meta-network to prune a network for the same task with different pruning ratios, allowing the pruned model to be used on devices with varying memory limitations. DHP [135] addresses the same problem without relying on reinforcement learning or evolutionary algorithms, as it makes the pruning procedure differentiable. Meta-learning has also been studied for

finding better weight initialization for pruning on different tasks. For example, Tian et al. [136] proposed a meta method that trains a well-initialized pruned meta-model to quickly adapt to different few-shot tasks, reducing overfitting. However, this meta-model is the same for all tasks and cannot generalize to devices with varying memory constraints. He et al. [137] apply a meta-learning method to select the best pruning criterion for different tasks. In [138], a shared sparse backbone network is trained for multi-task learning, but it is difficult to adapt to new tasks.

Conventional pruning methods perform well on single-task pruning but are more expensive for multiple tasks, as their cost linearly increases due to: (1) the need to train a large model for each task, and (2) the requirement of pruning each large pre-trained model from scratch for every task. For both CNN and ViT, these pruning methods are time-consuming when building pruned models for each unseen target task from a large pre-trained model. To the best of our knowledge, the non-parametric meta-pruning problem, i.e., how to prune a model for a target task using the pruned models of other tasks, has not been specifically studied in previous work. In Chapter 4, we propose a meta-pruning method that leverages the knowledge of pruned models from existing tasks to speed up pruning for new tasks. Our approach differs from existing methods in two main aspects: (1) we do not train a parametric meta-learner; instead, we use majority voting from similar tasks, and (2) our meta-voting generates a pruned small sub-network to initialize target task training, significantly reducing the pruning cost.

2.4 Continual Learning

The goal of continual learning (CL) is to make models capable of learning and adapting to new tasks or data over time without losing any of their prior learning. In this section, we will introduce popular methods employed in CL to alleviate forgetting. Furthermore, we will discuss some works that investigate the parameters in DNNs.

Replay-based Methods A line of methods has been developed to store samples of previous tasks to combat the forgetting of prior knowledge. ER [139] employs reservoir sampling [140] to maintain a memory buffer containing uniform samples from all previously encountered tasks. During training, each mini-batch is randomly sampled from both the current task and the buffered data to retain knowledge from previous tasks. DER and DER++ [1] store buffered data along with their output logits, allowing them to utilize knowledge distillation to recover previous knowledge during the learning of new tasks. Instead of random sampling, some methods focus on selecting critical samples from each task to mitigate forgetting: MIR [141] introduces a strategy for selecting memory samples that suffer the largest loss increase caused by the incoming mini-batch. This approach ensures that samples at the forgetting boundary are selected as buffered data. iCaRL [42] selects samples closest to the representation mean of each class and trains a nearest-mean-of-exemplars classifier to preserve class information of samples.

Due to data privacy concerns and memory limitations, some methods [142–144] suggest generating synthetic data for previous tasks and training them alongside current data to mitigate forgetting. Shin et al. [145] propose using a generative adversarial network (GAN) to create replay samples in CL. When a new task comes, the authors first update the generator to produce samples for each task. Then the classification model will be optimized for all tasks using synthetic data and current real data. He et al. [142] build upon this method by preserving additional real data for each task. They also develop a separate generator for each task to create samples specific to that task. Several methods [146–148] also integrate dataset distillation techniques into CL. Dataset distillation methods can compress a large dataset into a smaller set of informative synthetic samples, thereby reducing the memory demands of replay-based approaches in CL.

Model-based methods Another research direction involves imposing regularization

on model parameters critical to previous tasks to retain prior knowledge. EWC [44] first introduces the parameter regularization metric. They use the Fisher Information Matrix[149] to gauge the importance of each parameter on previous tasks within the network at the end of each task. During the training of a new task, an additional regularization term is incorporated into the loss function to constrain parameter changes based on their importance, allowing crucial parameters to retain the knowledge from previous tasks without forgetting. Various approaches to calculating parameter importance have also been proposed. MAS [46] determines the importance of parameters by examining the impact of altering each parameter on the model's outputs when new tasks come. SI [45] estimates parameter importance by computing the sensitivity of the loss function concerning each parameter. They calculate this importance value in an online manner, enabling them to integrate changes to parameters that were essential for all previous tasks.

To minimize interference between tasks and enhance the model's ability to retain previous knowledge, many methods allocate a specific set of parameters to each task or introduce new modules to the model. PNN [47] adds a new backbone for each new task, avoiding interference with previous tasks. When learning a new task, the backbones of previous tasks are employed to supply previously acquired knowledge to the new task. PackNet [150] combines CL with model pruning without adding new modules. The authors train a model on $Task_1$ and prune it to identify critical parameters for $Task_1$ in the network. The pruned parameters are then reinitialized to train subsequent tasks while the critical parameters of previous tasks remain fixed. This approach ensures the model does not forget prior knowledge, and the knowledge from previous tasks can be shared by previous tasks. Pham et al.[151] and Elahe et al.[152] introduce additional backbones to strengthen the memory of previous tasks. In [151], they propose DualNet, which comprises two components: fast and slow learning systems. The slow learner

is responsible for learning general, task-independent representations, while the fast learner uses sequential labeled data to quickly capture new knowledge. The general representations learned by the slow learner can be effectively adapted to the fast learner for acquiring a new task.

Analysis of different modules in neural networks In Chapter 5, our proposed method analyzes the sensitivity of different modules in CL. We will compare it with existing works and discuss their differences. Quang et al.[153] and Lesort et al.[154] investigate the impact of various normalization layers and classifiers on CL under a given setting. In contrast, our method examines the sensitivity of all parameters across different network architectures and scenarios. Wu et al. [155] explore the forgetting of various blocks in pre-trained language models by evaluating their representation ability. We provide a more fine-grained analysis of each module's forgetting based on their training dynamics. Our findings show that parameters of different types of modules have distinct sensitivities to forgetting. Ramasesh et al. [156] demonstrate that freezing earlier layers after training the first task has minimal impact on the performance of the second task. This is because their unfrozen portion includes the last fully connected (FC) layer and numerous batch normalization (BN) parameters, which are the most sensitive/critical according to our empirical study. Furthermore, they do not consider that earlier layers have significantly fewer parameters and capacity than the top layers. Zhang et al. [157] discover that, in different architectures, the parameters in the bottom layers (close to input) are more critical, and perturbing them results in poor performance. Our empirical study aligns with their findings, indicating that the earlier convolutional layer is sensitive to task drift and the induced biases on them lead to catastrophic forgetting.

In CL, replay-based methods consistently achieve SOTA performance, but replaying at every step is time-consuming. Regularization-based methods attempt to identify

critical parameters for different tasks during CL and limit the update of these parameters. However, this approach can also hinder task-agnostic parameters from learning shared knowledge across tasks. In Chapter 5, we analyze the training dynamics of CL to identify parameters sensitive to distribution shift. Our methods can serve as complementary techniques to all replay-based approaches. They can further enhance the accuracy and efficiency by finetuning a small portion of task-specific parameters on buffered data once (FPF) or occasionally (k-FPF).

CHAPTER 3

INTERPRETABLE REASONING PROCESS OF VARIOUS TASKS

3.1 Introduction

Deep Neural Networks (DNNs) have significantly revolutionized a multitude of tasks, especially with the emergence and evolution of pre-trained models. These models are initially trained on large datasets and are able to acquire useful generalized representations. To adapt to a variety of downstream tasks, the pre-trained models are subsequently finetuned or used as general-purpose feature extractors.

To effectively share the knowledge of pre-trained models, gaining a deep understanding of their behavior is essential. Model interpretability has attracted growing interest in a variety of areas. Existing methods essentially reveal the neuron-to-neuron correlations [8, 12, 13, 113, 158] (e.g., input to output, intermediate layer/neuron to output, etc), but they cannot provide an overview of the whole inference process occurring inside the complicated structure of DNNs for a given task. To facilitate knowledge sharing, it

is imperative to understand how the model learns, reasons, and makes decisions on a given task and identify the relationship between different tasks [159].

In this chapter, we study a more challenging problem: Can we unveil the major hidden steps of inference in DNNs for a given task and present them in a succinct and human-readable form? We develop an efficient tool called "neural net inference chains (NeuroChains)" to extract the underlying inference chains of a DNN only for a sub-task. A sub-task of a classification task can be defined as classification on a small subset of classes, which in this work refers to a few data drawn from a subset of $1,000$ classes in ImageNet [160]. NeuroChains aims to extract a much smaller sub-network composed of a subset of neurons/filters exactly copied from the original pre-trained DNN and the output for data from the sub-task is consistent with that of the original pre-trained DNN. While the selected filters explain the key information captured by the original pre-trained DNN when applied to data from the same task/class, the architecture of the sub-network stitches this information sequentially, i.e., step by step and layer by layer, and recovers the major steps of inference that lead to the final outputs. We extract chains from the sub-network and visualize their filters and featuremaps of the original network by existing methods [8, 9].

NeuroChains is a novel technique specifically designed for interpreting the inference chains of DNNs for a given task. As aforementioned, it potentially provides an efficient tool to study other problems, such as knowledge sharing in related tasks. NeuroChains is fundamentally different from existing network pruning and interpretation tasks, so methods developed for those two problems can't be used to solve our problem. Compared to network pruning: (1) finetuning is not allowed in NeuroChains; (2) it targets a much larger pruning rate for succinct visualization, e.g., $\geq 95\%$ for VGG-19 [161], $\geq 99\%$ for ResNet-50 [162] and $\geq 92\%$ for ViT [163, 164] on ImageNet with ≤ 200 filters or patches remained; (3) it is for a few samples drawn from a task instead of the whole data dis-

tribution. Compared to mainstream interpretation tasks: (1) NeuroChains produces an interpretation of the entire inference process for a specified task (e.g., a subset of classes) rather than of one neuron/filter or output on/around a single sample; (2) NeuroChains provide complementary information to the importance of individual neurons/filters for a task.

3.2 NeuroChains

3.2.1 Problem: Extract Sub-networks

Although the DNNs widely used nowadays are usually composed of hundreds of layers and millions to billions of hidden nodes, *when applied to samples from a sub-task (e.g., composed of two classes), it is plausible that its inference process mainly relies on a small subset of layers and filters.* In this work, we prove this conjecture by developing an efficient and practical algorithm, called NeuroChains, to extract the subset and its underlying architecture as a sub-network whose filters are selected and exactly copied from the original pre-trained DNN while its outputs for a given subset of data or classes stay the same as the ones produced by the original pre-trained DNN. Although DNNs are usually non-smooth in definition if using a non-smooth piece-wise activation such as ReLU, when trained with the commonly used techniques, e.g., data augmentation, mix-up, and dropout, the resulted DNNs are relatively smooth in a sufficiently small local region.

In order to preserve the original inference chains, we do not allow any finetuning or re-training on any filter or the weight vector corresponding to any neuron: they can only be exactly copied from the original pre-trained DNN. Let $F(\cdot;\{W^\ell\}_{\ell=1:L})$ (a mapping from input to output) denote the original pre-trained DNN, W^ℓ represents the set of filters/weight vectors in layer-ℓ, and $W^\ell[i]$ represents the ith filter/weight vector in layer-ℓ. Any sub-network fulfilling our above requirement can be defined and parameterized by an indicator vector M^ℓ per layer, whose each entry is a $\{0,1\}$ value indicating whether

retaining the associated filter/neuron in W^ℓ. We further define operator $\circ$ as

$$(W^\ell \circ M^\ell)[i] \triangleq \begin{cases} W^\ell[i], & M^\ell[i] = 1; \\ \vec{0}, & M^\ell[i] = 0. \end{cases} \tag{3.1}$$

Thereby, $\{M^\ell\}_{\ell=1:L}$ defines a qualified sub-network for inference chain and its weights are $\{W^\ell \circ M^\ell\}_{\ell=1:L}$, where we extend the operator $\circ$ to make $W \circ M = \{W^\ell \circ M^\ell\}_{\ell=1:L}$ given the original pre-trained DNN's weights $W = \{W^\ell\}_{\ell=1:L}$. Given a set of samples $\mathscr{X}$ drawn from a specific sub-task, we can formulate the problem of finding an inference chain as the following combinatorial optimization, which aims to find the most sparse indicator M (i.e., the sub-network with the fewest filters retained) that does not change the outputs of the original pre-trained DNN for $\forall x \in \mathscr{X}$, i.e.,

$$\min_{\{M^\ell\}_{\ell=1:L}} \sum_{\ell=1}^{L} \|M^\ell\|_1 \; s.t. \; F(x;W) = F(x;W \circ M), \; \forall x \in \mathscr{X}. \tag{3.2}$$

However, it is impractical to directly solve this combinatorial optimization since the possible choices for M^ℓ are of exponential number. We relax the 0-1 indicator vector M^ℓ to a nonnegative-valued score vector S^ℓ of the same size. We define an operator $\odot$ applied to W^ℓ and its associated scores S^ℓ as

$$(W^\ell \odot S^\ell)[i] \triangleq S^\ell[i] \cdot W^\ell[i]. \tag{3.3}$$

Note that we limit entries in S^ℓ within $[0,1]$ due to the possible redundancy among filters in the original pre-trained DNN, i.e., there might be filters of similar functionality for the given samples, and a preferred pruning should be able to only preserve one of them and multiply it by the number of those redundant filters in the sub-network. In addition, fewer constraints are easier to handle in optimization, and it is helpful to find sub-networks whose outputs are closer to those of the original pre-trained DNN, since the class of sub-networks with parameters $W \odot S$ includes all the sub-networks with parameters $W \circ M$. Hence, we relax the challenging combinatorial optimization to the following optimization, i.e.,

$$\min_{\{S^\ell\}_{\ell=1:L}} \frac{1}{|\mathscr{X}|} \sum_{x \in \mathscr{X}} l(F(x;W), F(x;W \odot S)) + \lambda \textstyle\sum_{\ell=1}^{L} \|S^\ell\|_1, \tag{3.4}$$

where $l(\cdot,\cdot)$ is a loss function aiming to minimize the distance between the original pre-trained DNN's output $F(x;W)$ and the sub-network's output $F(x;W\odot S)$. In our experiments, for classification, we use KL-divergence between the output distributions over classes, where the two output distributions are computed by applying softmax to $F(x;W)$ and $F(x;W\odot S)$ respectively, i.e.,

$$l(F(x;W),F(x;W\odot S)) = D_{KL}(softmax(F(x;W))||softmax(F(x;W\odot S))). \tag{3.5}$$

In addition, empirical evidence [165, 166] shows that for most samples, there are some layers that can be entirely removed without changing the final prediction. Hence, only a few hard and confusing samples need more delicate features, while most other samples can be correctly classified based on simple patterns from shallower layers. Therefore, in NeuroChains, we apply a sigmoid function with input score α^ℓ as a gate G^ℓ determining whether to remove the entire layer-ℓ during pruning, i.e.,

$$G^\ell = 1/\Big[1+\exp(-\alpha^\ell/T\Big)], \tag{3.6}$$

where T is a temperature parameter. With a gate G^ℓ applied after each layer-ℓ whose input and output have the same size (which is common in many DNNs), we can recursively define the input $H^{\ell+1}(\cdot)$ to the next layer-$(\ell+1)$, i.e., $H^{\ell+1}(x;\{W^{\ell'}\odot S^{\ell'},\alpha^{\ell'}\}_{\ell'=1:\ell}) =$

$$\begin{cases} G^\ell\cdot F^\ell(H^\ell;W^\ell\odot S^\ell)+(1-G^\ell)\cdot H^\ell(x; & \\ \{W^{\ell'}\odot S^{\ell'},\alpha^{\ell'}\}_{\ell'=1:\ell-1}) & \text{if input size = output size} \\ F^\ell(H^\ell;W^\ell\odot S^\ell) & \text{otherwise} \end{cases} \tag{3.7}$$

where $F^\ell(H^\ell;W^\ell\odot S^\ell)$ denotes the output of layer-ℓ. The reason to use a gate here is that we expect to either remove the whole layer or retain it without adding an extra shortcut (which will change the original pre-trained DNN's architecture). Since we prefer to remove non-informative layers, in the objective, we add another regularization α^ℓ to encourage the removal of entire layers (because decreasing α reduces G^ℓ and thus increases the chance of layer removal). Therefore, the final optimization for NeuroChains

is

$$\min_{\{S^\ell,\alpha^\ell\}_{\ell=1:L}} \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} l(F(x;W),H^{L+1}(x; \{W^{\ell'}\odot S^{\ell'},\alpha^{\ell'}\}_{\ell'=1:L})) + \lambda\sum_{\ell=1}^{L}\|S^\ell\|_1 + \lambda_g\sum_{\ell=1}^{L}\alpha^\ell, \tag{3.8}$$

Our objective above is similar to the one used in Network Slimming [121] but we optimize it for a sub-task (so we can consider removing layers) and we do not allow finetuning on weights W.

Algorithm 1 NEUROCHAINS

Input : Pre-trained network $F(\cdot;\{W^\ell\}_{\ell=1:L})$, sub-task T and training set D_T, pruning iterations N, threshold τ
Output : An interpretable sub-network for sub-task T
Initialize : Filter scores $\{S^\ell\}_{\ell=1:L}=\vec{1}$, gate score $\{\alpha^\ell\}_{\ell=1:L}=0$

```
Fix {W^ℓ}_{ℓ=1:L};
for n ← 1 to N do
    Apply one optimization step on D_T to fine-tune filter scores S according to Eq.3.8,
     and l(·,·) in Eq.3.8 refers to KL-divergence loss;
end
for ℓ ← 1 to L − 1 do
    if α^ℓ = 0 then
        Prune layer ℓ;
    else
        for i ← 1 to ‖S^ℓ‖_1 do
            if S^ℓ[i] < τ then
                Prune filter i in layer ℓ;
            end
        end
    end
end
```

3.2.2 Algorithm

Our algorithm is simply a standard back-propagation for the optimization problem in Eq. (3.8), which produces sparse scores for filters and gate values for layers. The detailed procedure of NeuroChains is described in Alg. 1. Note that the weights in W are fixed and

the backpropagation only updates S. We initialize the filter scores $S = \vec{1}$ so $W \odot S = W$ at the beginning of optimization. We initialize the gate score $\alpha^\ell = 0$ for all $\ell = 1 : L$ so $G^\ell = 0.5$ at the beginning, i.e., the probabilities to remove or to retain a layer are equal. For classification, we set loss $l(\cdot, \cdot)$ to be the KL-divergence between the output distributions of the original pre-trained DNN and the sub-network. After convergence of the optimization, we then apply simple thresholding to these scores to further remove more filters and layers: (1) we remove the filters with scores under a threshold τ; (2) we remove layer-ℓ if $G^\ell < 0.5$. This yields a sufficiently small sub-network architecture. Given a sub-network produced by NeuroChains, we then visualize its architecture and scores as the structure of the inference chains. Moreover, for samples from the same region, we visualize their inference pathway from the original pre-trained DNN by their activation patterns and featuremaps, respectively, using existing interpretation methods [8, 9].

3.3 Experiments

Statistics	ResNet-50	VGG-19
Top-1 test accuracy	76.5%	72.9%
Test images/sub-networks	10,000/1,688	10,000/1,746
Convolutional filters	26,560	4,480
Parameters of Conv-layers	23,454,912	20,018,880
Parameters of FC-layers	2,048,000	123,633,664

Table 3.1: Information of pre-trained CNNs in this work.

In experiments, we apply NeuroChains to extract the inference chains of the widely-adopted VGG-19, ResNet-50, and ViT, which are all pre-trained on ImageNet. We provide the basic information on the three DNNs in Table 3.1 and Table 3.2. In the following, we will present two quantitative analyses over hundreds of case studies, which show that (1) NeuroChains is capable of producing sub-networks retaining only $< 5\%$ of filters and

Statistics	ViT(DeiT-small)
Top-1 test accuracy	79.9%
Test images/sub-networks	10,000/1,500
Patches across blocks	2,364
Parameters of MHA-blocks	7,096,320
Parameters of FFN-blocks	14,178,816

Table 3.2: Information of pre-trained ViT in this work.

meanwhile preserving the outputs of the original pre-trained DNN in most cases; (2) the filters selected by NeuroChains with high scores are important to preserving the outputs since removing one will lead to a considerable drop in performance. We also compare the capability of preserving the original neural network's outputs between NeuroChains, magnitude-based pruning, and random pruning methods. We will then provide several detailed and insightful case studies and visualizations of extracted sub-networks for different sub-tasks.

3.3.1 Implementation Details

We implement NeuroChains by PyTorch [167]. In every case study, we first randomly sample 2 classes in ImageNet and then randomly sample 10 images from each class's images. Note that the sampled images may have been wrongly classified into other classes by the original DNN. We apply inference to those 20 images, and their outputs are used in solving the optimization of Eq (3.8) in order to extract the local inference chain in the form of a sub-network. For models with shortcuts, e.g., ResNet-50, the sigmoid gate is applied to prune a bottleneck block rather than a layer. A layer inside a block will be removed if the scores of all filters in the layer are nearly 0.

We use the Adam optimizer for the optimization of Eq (3.8) for filter/layer scores. We use a fixed learning rate of 0.005. We set the temperature $T = 0.2$ in the sigmoid gate (Eq. (3.6)) to encourage the value G^{ℓ} close to either 0 or 1, and the threshold τ to goal

scores is set to 0.1 so that the outputs of sub-networks are consistent. We only tried a limited number of choices on tens of experiments, chose the best combination balancing fidelity and sub-network size, and then applied it to all other experiments without further tuning. In particular, we tried $\tau \in \{0.01, 0.1, 0.5\}$, $\lambda \in \{0.001, 0.005, 0.01, 0.1\}$, and $\lambda_g \in \{1, 2, 5\}$. For different models, the weights of two penalties in Eq. (3.8) are different. For VGG-19, we use $\lambda = 0.005$ and $\lambda_g = 2$. While we choose $\lambda = 0.005$ and $\lambda_g = 1$ for ResNet-50. This choice performs consistently well and robustly in all other experiments. The iteration steps of training are 300 and we stop training when the loss difference is quite small, i.e., less than 0.05. It costs only ~ 90s for VGG-19 and ~ 55s for ResNet-50 to extract a sub-network on a single RTX 6000 GPU since we only optimize a few scores.

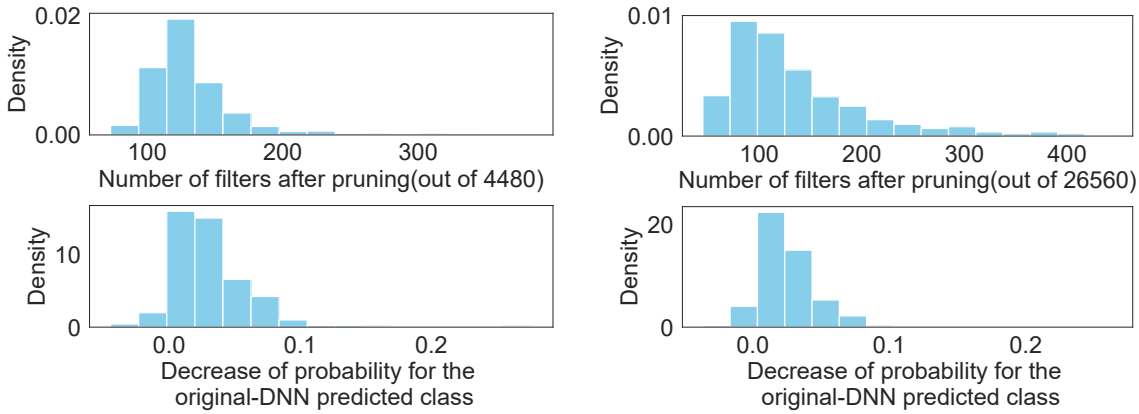


Figure 3.1: Size and fidelity (how well the sub-networks preserve the original pre-trained DNN's outputs) of 1,500 sub-networks extracted by NeuroChains for VGG-19 (left) and ResNet-50 (right) in different case studies (tasks). The x-axis in the top plot refers to the number of retained filters, while the x-axis in the bottom plot is the decrease of probability on the original-DNN predicted class. It shows that NeuroChains can usually find very small sub-networks and, meanwhile, preserve the original pre-trained DNN's outputs.

3.3.2 Quantitative Analyses

Melis et al. [168] propose some criteria to evaluate the interpretation methods for DNNs. In this work, we extend some of their notations and present two quantitative analyses of

NeuroChains over 1,500 case studies for different sub-tasks, i.e., (1) Fidelity: does the sub-network preserve the original pre-trained DNN's outputs on the given samples? how does it change for sub-networks of different sizes? (2) Faithfulness: how well does the importance score of a filter reflect the degeneration in fidelity caused by removing the filter from the sub-network? In this work, we evaluate the fidelity and faithfulness by the decreasing amount of probability on the original pre-trained DNN's predicted class when using the sub-network for inference. All the above metrics are averaged over 1,500 sub-networks and across all the images used to extract each sub-network.

Fig. 3.1 shows the statistics of the fidelity for sub-networks of different sizes (measured by the number of filters) that are extracted by NeuroChains. Most sub-networks only retain $\leq 1\%(\leq 5\%)$ of ResNet-50(VGG-19) for succinct visualization, but they preserve the outputs of ResNet-50(VGG-19) with high fidelity.

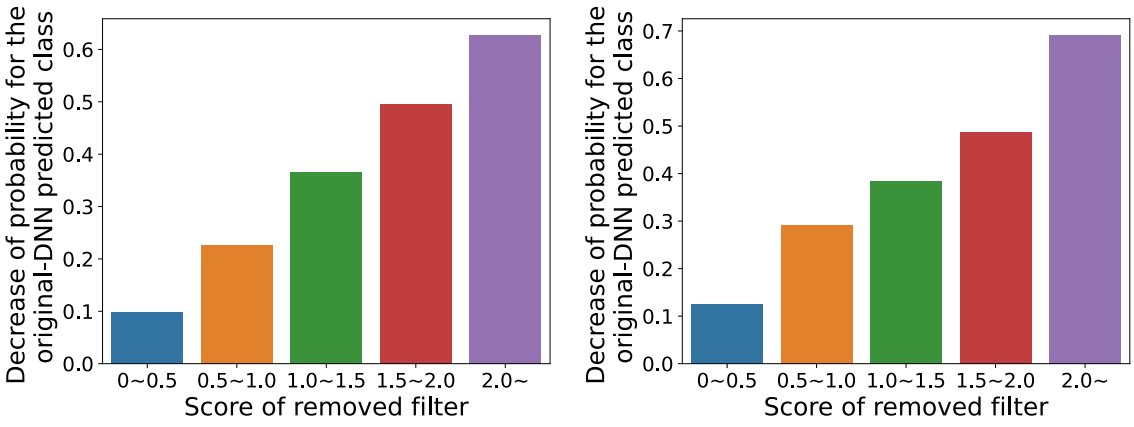


Figure 3.2: Barplot of the faithfulness (filter score vs. fidelity degeneration caused by removing the filter) of 1,500 sub-networks, each extracted by NeuroChains on 20 uniform samples randomly drawn from two classes for VGG-19 (left) and ResNet-50 (right). The x-axis refers to the interval of filter scores, while the y-axis denotes the decrease of the sub-network's probability on the original DNN predicted class after removing a filter from the sub-network. It shows that the sub-networks suffer from more degeneration if a filter with a higher score is removed. Hence, the scores faithfully reflect the importance of filters in explaining the original pre-trained DNNs.

Fig. 3.2 reports the faithfulness of extracted sub-networks, i.e., how a sub-network's performance in preserving the original pre-trained DNN's output degrades if removing

one filter from it, and what is the relationship between this degeneration and the score of the removed filter. The statistics on 1,500 sub-networks in Fig. 3.2 show that as the scores of filters increase, the fidelity degrades more. The degeneration and the score are strongly and positively correlated, indicating that our optimized scores faithfully reflect the importance of filters in explaining the original pre-trained DNN. Moreover, removing even one highly scored filter from the sub-network can significantly degrade the explanation's performance. Hence, NeuroChains usually find the smallest sub-networks without redundancy among retained filters/layers, i.e., every critical inference step is retained.

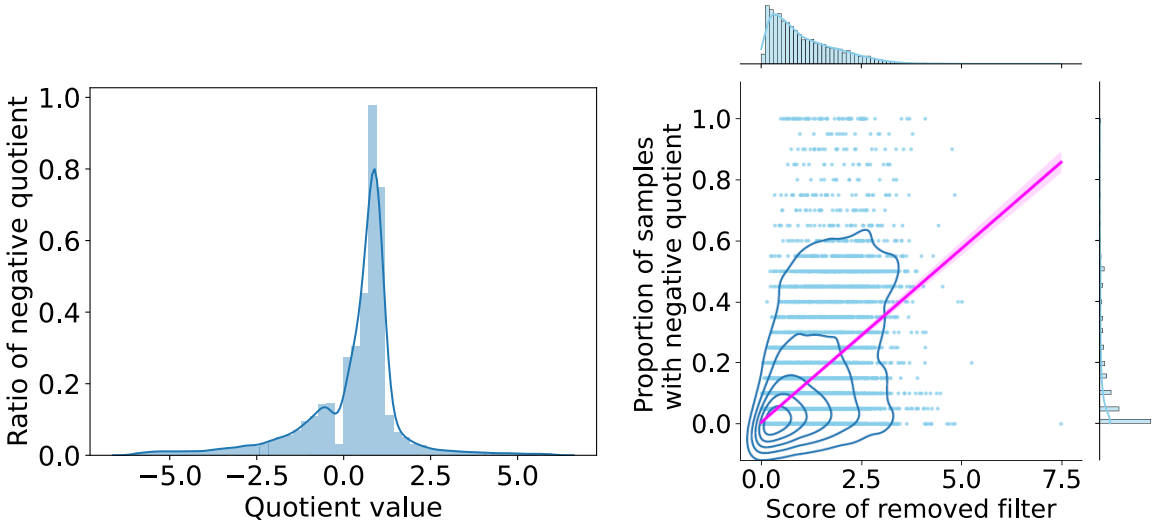


Figure 3.3: Left: Histogram of the quotient metric in Eq. (3.9) computed over all the 1,500×20 samples. Right: Faithfulness of NeuroChains in terms of the quotient's sign.

We also present another faithfulness study based on a quotient metric defined below. Let $p, q \in \Delta^c$ (Δ^c is the probability simplex for c classes) be the output probability vectors of the original neural net and the extracted sub-network, respectively, for the same input. We define a quotient metric to measure the change of class prediction between p and q, i.e.,

$$Q(p,q) = \frac{q[y] - \max_{z\in[c], z\neq y} q[z]}{p[y] - \max_{z\in[c], z\neq y} p[z]}, \quad y \in \arg\max_{z\in[c]} p[z], \tag{3.9}$$

where y is the predicted class by the original neural net, and $Q(p,q)$ is the quotient of two probability differences computed respectively on the original neural net and the sub-network. In particular, it computes the difference of probabilities for class y and the highest-rated other class. The sign of $Q(p,q)$ indicates whether the predicted class changes (e.g., it changes if $Q(p,q) < 0$) while the magnitude of $Q(p,q)$ measures the change in prediction confidence.

The result is consistent with our above observations and is given in Fig. 3.3. The left plot is the histogram of the quotient computed over all the 1,500×20 samples. The histogram shows that most samples keep the original predicted label after pruning, i.e., NeuroChains can preserve the original pre-trained DNN's outputs in most cases. Moreover, the number of filters preserved in these sub-networks is $157(mean) \pm 43(std)$, which is small enough to explain. The right plot reports the faithfulness of NeuroChains in terms of the quotient's sign. We remove each filter from each sub-network and report how many samples' predicted labels are changed after the removal, i.e., the quotient is negative. Each point in the scatter plot corresponds to a sub-network, the x-axis is the score of the removed filter given by NeuroChains, and the y-axis is the proportion of samples with negative quotients. The plot shows a strong linear correlation between the score of the removed filter and the degradation of faithfulness. Since removing filters with high scores results in more samples with predicted class changes after pruning, the score given by NeuroChains measures the importance of filters in DNN inference.

We compare the capability of preserving the original neural network's outputs between NeuroChains and magnitude-based pruning (removing the filters whose output featuremaps' average magnitude (L2 norm) over all considered samples is small) and random pruning ("by chance") in Fig. 3.4. In particular, under the same setting of each experiment in the work, we prune the original VGG-19 and retain the filters with the largest featuremap magnitude in each layer or randomly, 180 in total (more than

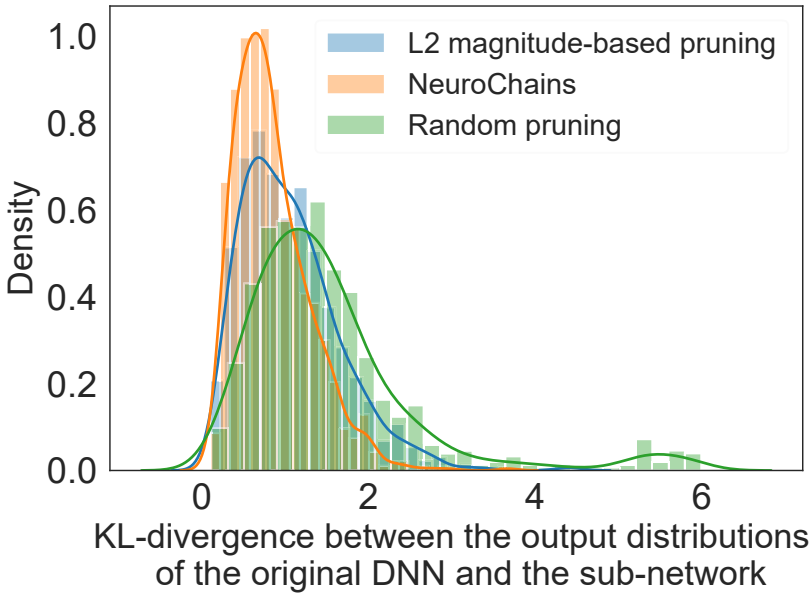


Figure 3.4: Comparison of different pruning methods on the capability of preserving the original network's output distribution (smaller KL divergence means better preservation) over 1,500×20 uniform samples.

$157(mean) \pm 43(std)$ filters for sub-networks extracted by NeuroChains), and we then finetune the filters' scores. For Fig. 3.4, the percentage of cases with KL-divergence≤ 0.8 for NueroChians, magnitude-based pruning and random pruning are 54.3%, 38.8%, 21.1% respectively, and 9.3%, 19.4%, 38.8% cases have KL-divergence≥ 1.5. Fig. 3.4 shows the histogram of the KL divergence between the original output class distribution and the one produced by the sub-networks. For sub-networks generated by NeuroChains, the KL-divergence in most cases stays close to 0, while the output preserving capability of simple pruning is much worse.

3.3.3 Case Studies

We present three case studies of the sub-networks extracted by NeuroChains. For Fig. 3.5 and Fig. 3.6, the data points are from classes that are easy to tell apart, while in Fig. 3.7 images are sometimes mis-classified. The case study of the multi-class classi-

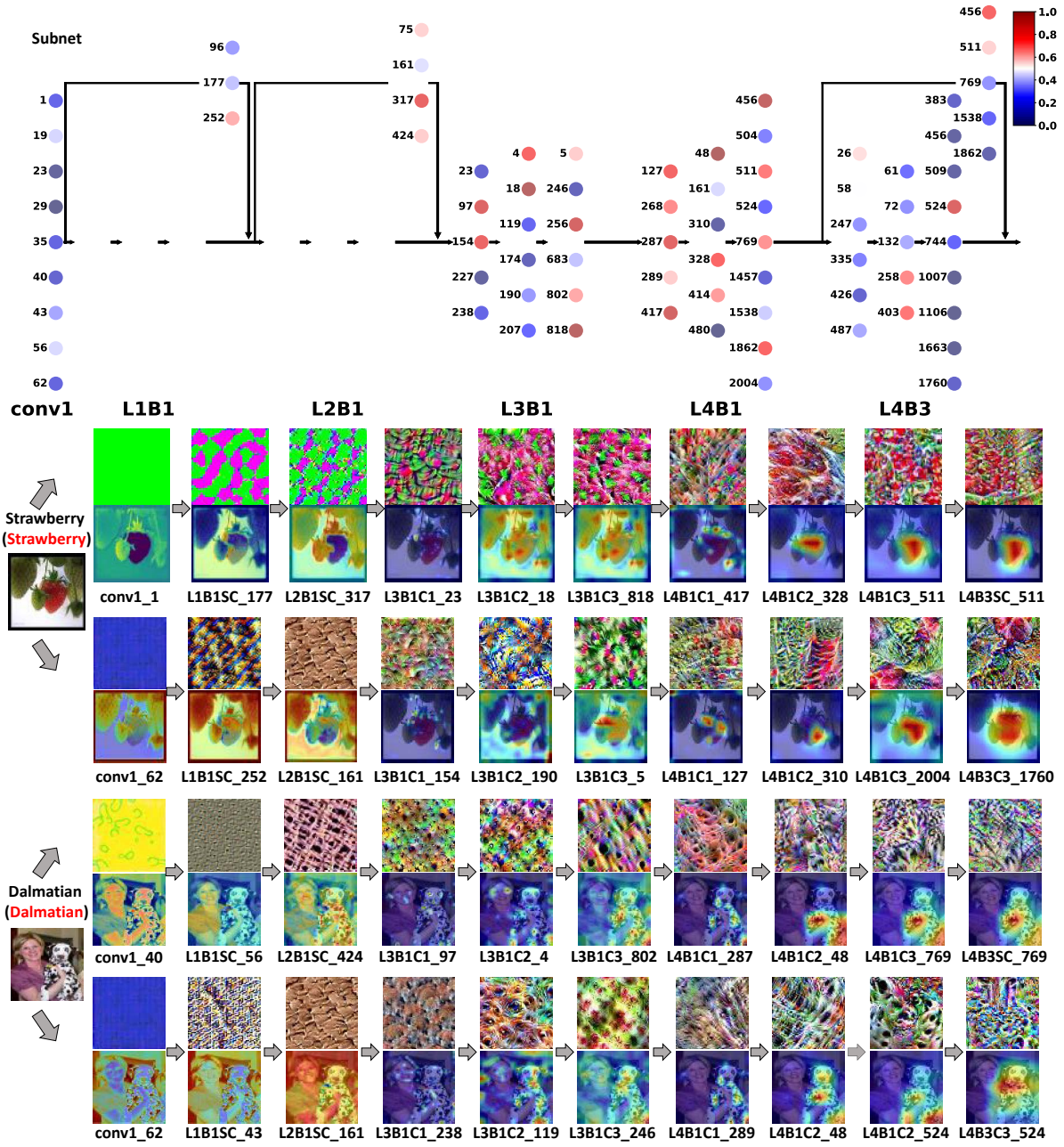


Figure 3.5: Inference chain by NeuroChains for ResNet-50 (pre-trained on ImageNet) when applied to 20 test images of “strawberry” and ”dalmatian”. The sub-network retains only 13/67 layers and 77/26,560 filters of the ResNet-50. Refer to Section 3.3.4 for a detailed analysis of extracted chains.

fication task is shown in Fig. 3.8. The visualization in each case study is composed of two parts: (1) the sub-network's architecture and filter scores; (2) the original images from each class and the visualization of the image's NeuroChains extracted from the original pre-trained model. The true class and predicted class of the sample is shown above the image. They show that NeuroChains considerably enrich the explanation details of DNN's inference process. By connecting the important filters from different layers, the extracted sub-network highlights the main steps leading to the output prediction.

On the sub-network's architecture, we use "L0" to denote the corresponding convolution layer in VGG-19 and "L0_1" to denote the first filter from this layer. For ResNet-50, we further use "L1B1" to denote the first sub-block in the first bottleneck block, "SC" for the shortcut connection, and "C1" for the first convolution layer in the sub-block. The redder the node in the sub-network, the larger the scaling score; conversely, the bluer the node, the lower the score.

NeuroChains are stitched by filters step by step, and the visualization of each filter and the corresponding featuremap generated by the original pre-trained model are displayed in each step. The visualization of each selected filter is achieved by maximizing its activation relative to the input. Afterward, we show the patterns that the filter aims to detect, which are independent of the input image.

To obtain the heatmaps, each filter applied in the sub-network produces a featuremap for an input image. By applying an activation function to the featuremap, only a few pixels are activated, and they highlight the regions in the featuremap that represent the features/patterns of the input sample that are detected by the filter. Since the featuremap size is always smaller than the input image, we apply bilinear interpolation to upsample the featuremap up to the input image size. Finally, we overlap the resized featuremap with the input image, which results in a heatmap that highlights the regions of the input image represented by the corresponding filter.

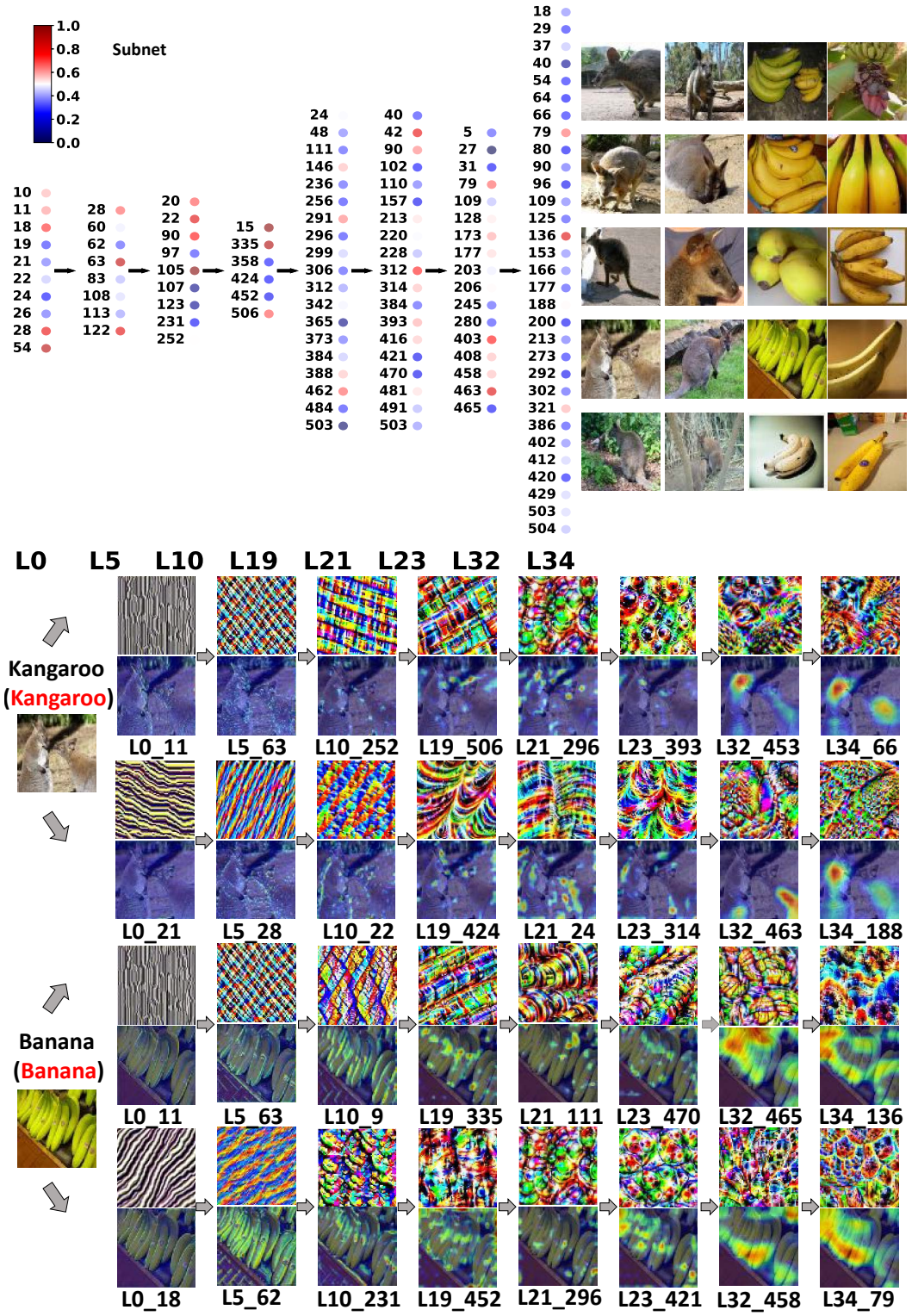


Figure 3.6: Inference chain by NeuroChains for VGG-19 when applied to images of “kangaroo” and ”banana”. The sub-network retains only 8/16 layers and 116/4,480 filters of the VGG-19. The scores for selected filters are represented using the colormap on the top-right. Refer to Section 3.3.4 for a detailed analysis of extracted chains.

3.3.4 Detailed Analysis of Case Studies

In Fig. 3.5 and Fig. 3.6, the case studies of VGG-19 and Resnet-50 are shown. For each sample from the sub-task, two NeuroChains which present the inference process of the original pre-trained model, are displayed.

In Fig. 3.5, features extracted by the chains of the strawberry gradually evolve from low-level to high-level, e.g., in the first chain, L1B1SC_177, L2B1SC_317, and L3B1C1_23 extract the low-level patterns of red-green color and texture, L3B1C2_18, L3B1C3_818, and L4B1C1_417 look like the abstract pattern of strawberry, and L4B1C2_328, L4B1C3_511 and L4B1SC_511 capture the pattern of the skin of strawberry. While the first chain of the strawberry focuses on the skin, the second chain extracts the pattern of the leaves, i.e., L3B1C1_154 and L3B1C3_5. The leaves in the featuremaps of L4B1C1_127 and L4B1C3_2004 are highlighted. L4B3C3_1760 extracts the global patterns of the strawberry. Both the two chains of the dalmatian extract the global patterns of the dalmatian's black and white fur. In shallow layers, L3B1C1_97, L3B1C1_238, L3B1C2_4, and L3B1C2_119 capture a more local black spot pattern of the dalmatian, while L4B1C1_289, L4B1C2_48, L4B1C3_769 and L4B3C3_524 reveal the global patterns.

In Fig. 3.6, the featuremaps show that the first chain of the kangaroo extracts the pattern of eyes and noses, while the second chain pays more attention to the fur of kangaroos. L21_296, L23_393, L32_453, and L34_66 turn from a round pattern to an eye and nose pattern. L21_24, L32_463, and L34_188 look like the fur patterns of animals. In the first chain of bananas, L19_335, L21_111, L23_470, and L32_465 turn from the curved rectangle pattern to a hand of bananas. The L32_458 and L34_79 in the second chain of bananas seem like the patterns of the toes and roots of bananas. The corresponding areas in the featuremaps are also highlighted. The two chains focus on bananas' global and local patterns, respectively.

In Fig. 3.7, we first show the NeuroChains of the correctly predicted castle and stone

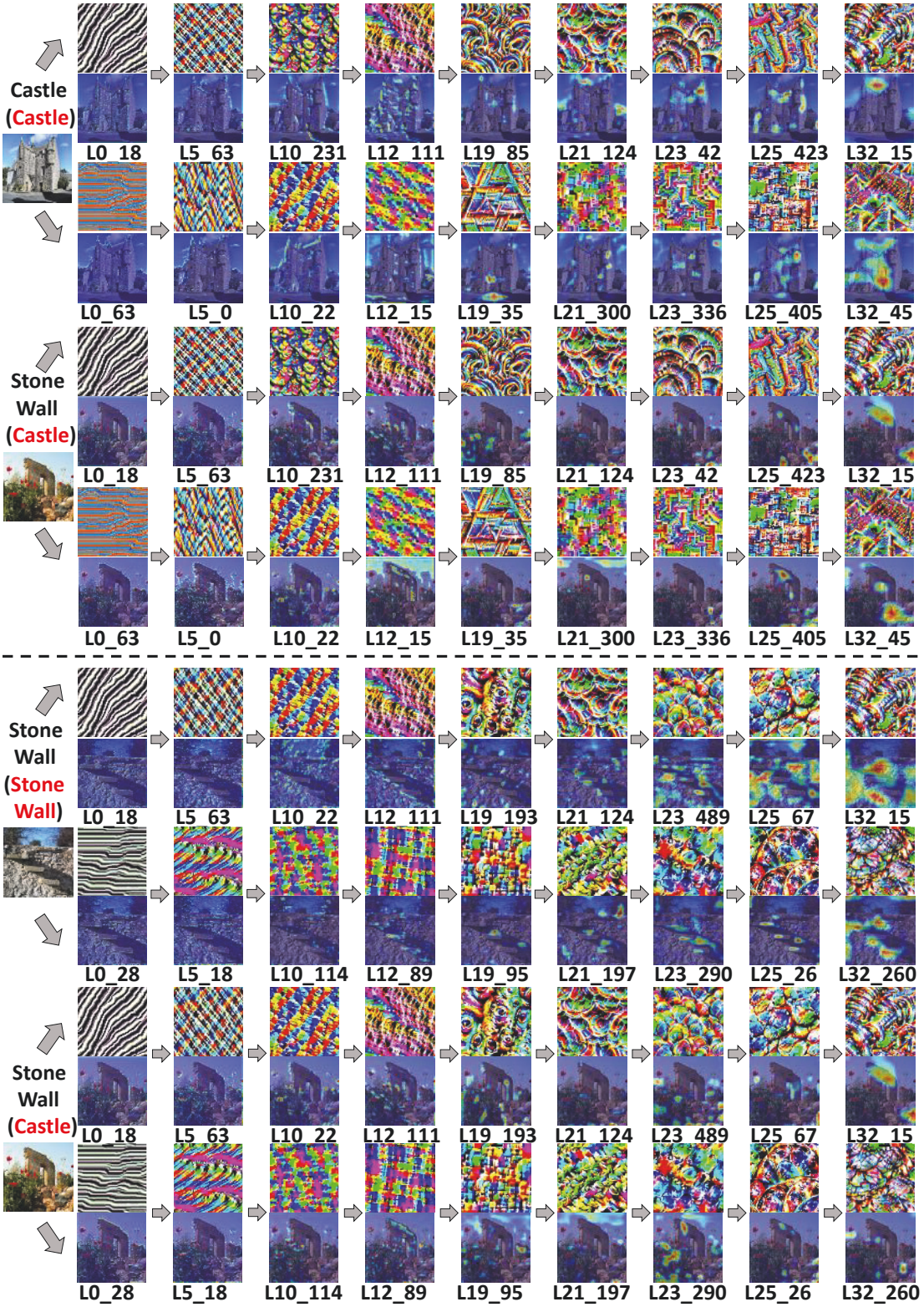


Figure 3.7: Inference chain by NeuroChains for VGG-19 when applied to images of "Castle" and "Stone Wall" which confuse DNN models and may be misclassified. The sub-network retains only 10/16 layers and 121/4,480 filters of the VGG-19. Refer to Section 3.3.4 for a detailed analysis of extracted chains.

wall. Then, using the same filters from the extracted chains, we draw the featuremaps of a stone wall, which is wrongly predicted as the castle by the original pre-trained model, to uncover the inner mechanism of the false prediction. The featuremaps of the wrongly predicted stone wall from filters activated by the castle show that the featuremaps in the mid-part of the chains are rarely activated, like L21_300 and L23_336, which is in line with our expectations. But in the deep layers, L32_15 and L32_45, which activate the roof and body of the castle, also highlight a wide area in the wrongly predicted stone wall image. This indicates that the wrongly predicted stone wall sample contains similar features as the castle. The featuremaps of the wrongly predicted stone wall from filters activated by the stone wall (correctly predicted) also show the anomaly. L21_197, L25_26, and L32_260, which are critical to the correctly predicted stone wall, extract little information from the wrongly predicted stone wall. We find that L21_124 and L32_15 are both important to identify the castle and stone wall, which implies that the castle and stone wall have similar patterns and are easily confused.

NeuroChains can also be used to extract the sub-networks for tasks with multiple classes, and the extracted sub-network is still small enough to be analyzed and explained. In Fig. 3.8, the NeuroChains of a 3-classes sub-task are presented.

The chains of orangutans extract features from different parts of the orangutan. The top chain of the orangutan extracts the pattern in the face. L23_82 and L25_27 show the round and black node patterns. Then in L30_42 and L32_453, these nodes evolve into the pattern of eyes and noses. In L34_112, these features are integrated into the entire face of the orangutan. The second chain of the orangutan mainly focuses on the fur and shape of its arms. In L25_220, L30_439 and L32_376, the elbow part in the featuremap is highlighted, and the visualization of filters also shows the pattern of the curved shape of the joint part. In the first chain of the tram, the pattern of the arranged windows is extracted in L25_202, L30_329 and L32_491 and the windows in the featuremap

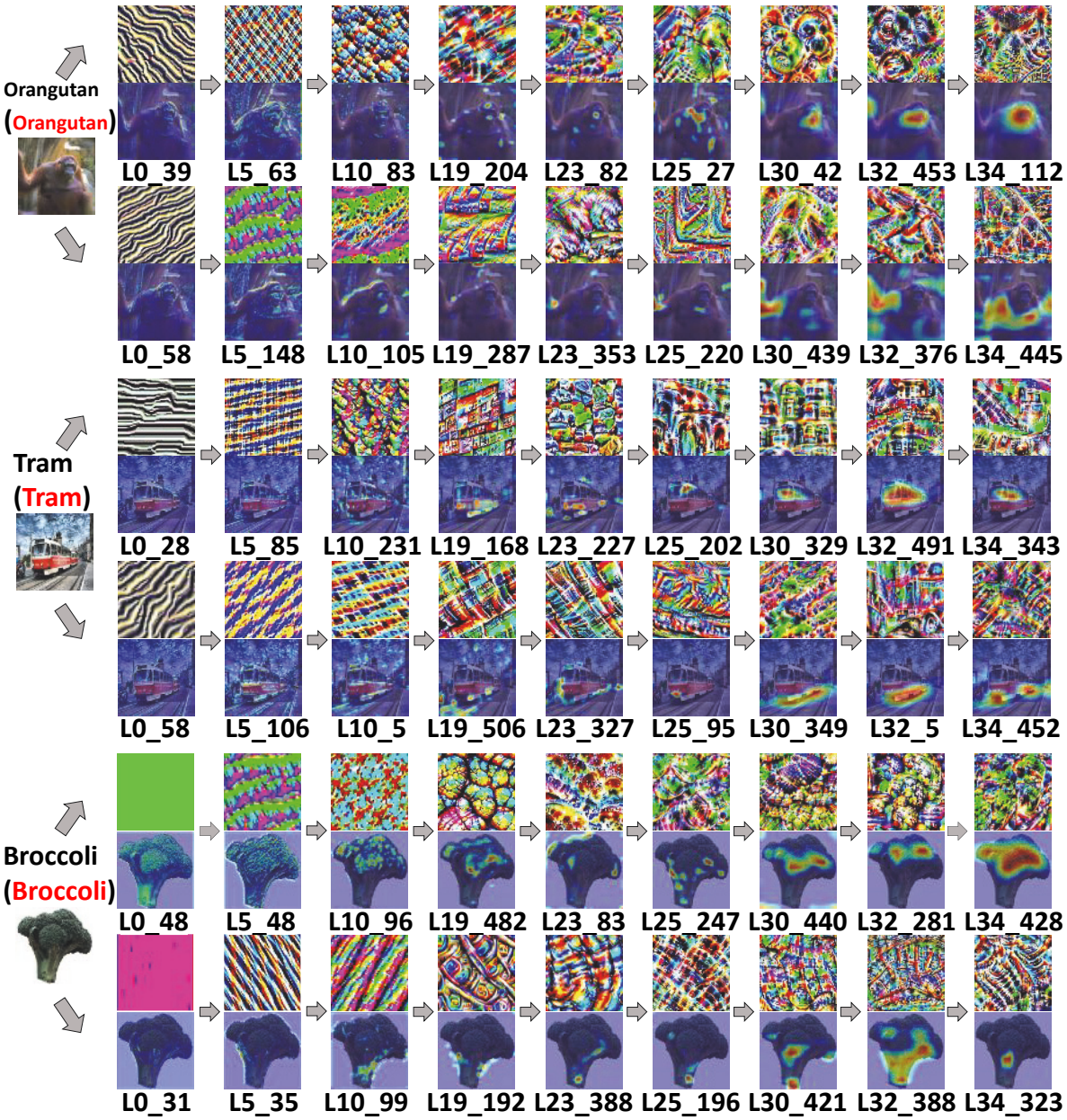


Figure 3.8: Inference chain by NeuroChains for VGG-19 when applied to images of ‘orangutan”, ”tram” and ”broccoli”. Its sub-network retains only 8/16 layers and 158/4,480 filters of the VGG-19.

are highlighted. From the featuremap of the second chain, the bottom part of the tram is highlighted, and the pattern in L32_5 seems like wheels, but the visualization of other filters is not readable, which indicates that this visualization method still has limitations. In the chain of broccoli, the head part is extracted by the first chain while

the stem part is captured by the second chain. From layer 10 to layer 34 of the first chain, L19_482, L23_83 and L25_247 show the pattern of a dense semisphere, and L30_440, L32_281 and L34_428 look like the pattern of the head of broccoli. In the second chain, the pattern of closely arranged columns is shown in L30_421, L32_388 and L34_323, which is consistent with the shape of the stem of the broccoli. From these analyses, we can find that despite the small number of filters in the sub-network, for samples from different classes, more than one critical pattern is captured to identify the samples. It shows the strong expression ability of neural networks and the interpretation ability of NeuroChains.

3.3.5 NeuroChains of Vision Transformer

To show that NeuroChains can be applied to different types of neural networks for comparing their reasoning chains, we additionally apply NeuroChains to the vision transformer (ViT). Since recent work [132] discovered the effectiveness and advantages of patch pruning for ViT, our method produces a score for each patch in different blocks of ViT. In particular, NeuroChains freezes all the parameters in ViT and only optimizes the scores for each patch. At the end of patch pruning, patches with small scores are removed. The patch scores are not shared among training samples, i.e., each input sample has its own patch scores, so the critical patch for each sample can be preserved. In Fig. 3.9, we provide a case study of NeuroChains on ViT. The top figure shows the patches (red squares) retained in each ViT block. In block-1, the whole image is taken as input. As the blocks get deeper, more patches are removed, and only a few patches are retained in the last few blocks. The pruning results imply that the patch preserved in the last block contains the most critical information to classify the input sample.

To study the inference process of ViT on the given sample, we generated the attribution heatmaps for both the class token and patch token in different blocks. The

attribution scores are computed as follows:

For a ViT with L blocks and N patches, the attention score between patches in block-ℓ can be calculated by $\mathbf{Attention}_\ell = softmax(\frac{\mathbf{Q}_\ell \mathbf{K}_\ell^\mathbf{T}}{\sqrt{\mathbf{d}_{\mathbf{k}_\ell}}})\mathbf{V}_\ell$, where the Q_ℓ,K_ℓ and V_ℓ are queries, keys, and values in the multi-head attention module of block-ℓ. The attribution of each patch in block-ℓ to the original patches in the input sample is computed as

$$\mathbf{Attribution}_\ell = \mathbf{Attention}_\ell \times \mathbf{Attribution}_{\ell-\mathbf{1}} + \mathbf{Attribution}_{\ell-\mathbf{1}}, \tag{3.10}$$

$\mathbf{Attribution}_\mathbf{0} = I_N$, indicating that in the input all patches only attribute to themselves. Since $\mathbf{Attention}_\ell$ is calculated based on the output patches of the previous block, in Eq. 3.10, we multiply the $\mathbf{Attention}_\ell$ by $\mathbf{Attribution}_{\ell-\mathbf{1}}$ to include the attribution to patches in the current block. We further add $\mathbf{Attribution}_{\ell-\mathbf{1}}$ to reflect the shortcut operation in ViT blocks: the output patches of block-$(\ell-1)$, which encode the attribution of patches in block-$(\ell-1)$, are added directly to the output of block-ℓ.

In the middle row of Fig. 3.9, we show the attribution heatmaps of the class token in different blocks of the pruned ViT. In the shallow blocks close to the input, it attends all patches with similar attribution scores. As the blocks get deeper, the class token gradually focuses on the critical patches essential to classify the input image.

In this case study, only one patch token is retained in the last block of the pruned ViT. The attribution heatmaps of this patch are shown in the bottom row of Fig. 3.9. The patch token pays more attention to itself in the first few blocks but it starts to attend to more important patches in later blocks that are more relevant to the predicted class.

The results of applying NeuroChains to ViT in Fig. 3.9 show that the inference processes of ViT and CNN are very different. In CNN, from the shallow layers to the deep ones, the model first extracts the low-level local patterns of the input image and then aggregates them to get high-level global patterns, step by step, towards classifying the input image. In ViT, the model can capture both the local and global patterns of the input since the very bottom/early blocks/layers. During inference, the model gradually

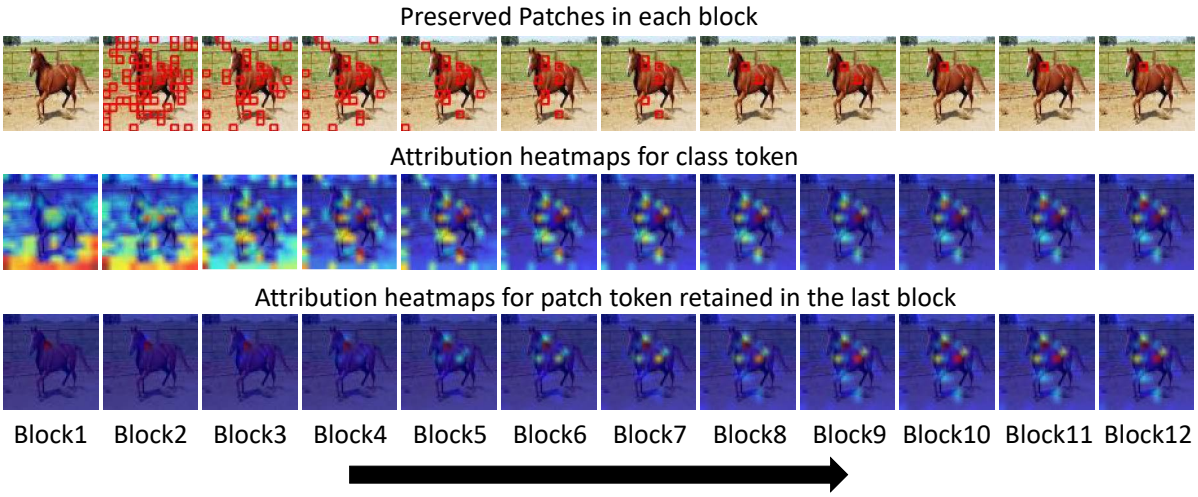


Figure 3.9: A case study of applying NeuroChains to vision transformer (ViT). Top: the retained patches (red squares) in each block. Middle: attribution of the class token to the original input patches. Bottom: attribution of the patch token (retained in the last block) to the original input patches.

identifies patches important to the classification task and mainly extracts patterns relevant to the class token.

3.4 Ablation Studies

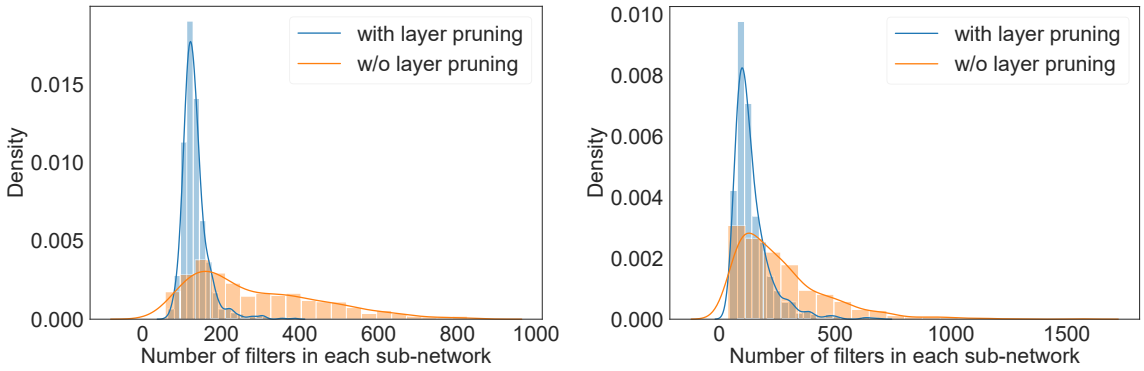


Figure 3.10: Comparison between the number of filters in sub-networks extracted with or without layer pruning on VGG-19 (left) and ResNet-50 (right).

Effect of Layer Pruning In Fig. 3.10, we compare the number of filters in sub-networks extracted with or without layer pruning. When layer pruning is applied,

for both VGG-19 and ResNet-50, most sub-network sizes lie in a small range of 0 to 200. Without layer pruning, many more redundant filters will be preserved in the sub-networks. Hence, layer pruning is essential to building small, human-interpretable, and effective sub-networks.

Effect of the Number of Classes In Fig. 3.8, we show a case study of applying NeuroChains to a 3-classes classification sub-task, demonstrating the capability and effectiveness of NeuroChains on multi-class problems. However, as classes per sub-task increase, the number of preserved filters in the sub-networks also increases, which may weaken the interpretability. We test NeuroChains on the 10-classes classification sub-tasks. The average number of filters in the extracted sub-networks is 795(1206) for VGG-19(ResNet-50). Though much smaller than the original network, the extracted sub-networks are too large to explain.

That being said, NeuroChains can explain the correlation among multiple classes by identifying the chains important to distinguishing a class from the others. We conduct a binary-classification experiment shown in Fig. 3.11. In this sub-task, samples in one class are all eagles, while samples in another class are randomly selected from different classes, as shown in the top right of the figure.

From the figure, the sub-network extracts two characteristic patterns for eagles. The first chain detects the pattern of feathers, while the second chain extracts the pattern of eyes and heads. These two patterns are enough to recognize eagles. For images from other classes (a horse image in this case study), the key pattern of the object will be identified. In the figure, the first chain of the horse locates the elbow part. Filter $L30_286$ contains both the patterns of eyes and elbows. However, the second chain of the eagle only activates the eyes; in the chain of the horse, the elbow of the horse is highlighted. The unique patterns in other objects will be extracted in order to distinguish them from eagles. In the second chain of the horse image, we use the filters in the first chain of

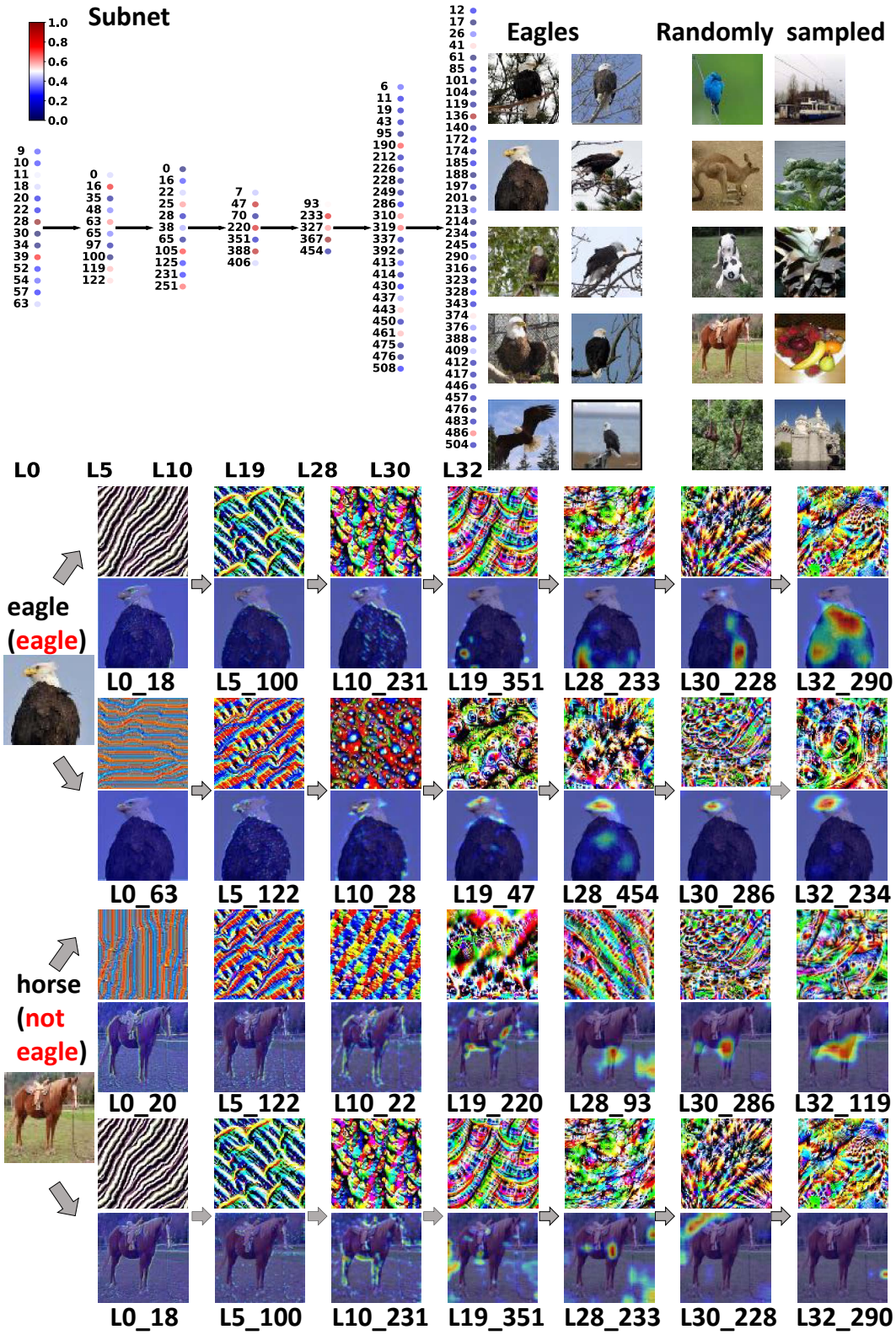


Figure 3.11: Inference chain by NeuroChains for VGG-19 when applied to tell apart images of "eagle" and other randomly selected samples. The sub-network retains only 7/16 layers and 110/4,480 filters from the original VGG-19.

the eagle to extract patterns for the horse image. The featuremaps show that the eagle filters are not activated on the horse image, so it can be distinguished from eagles.

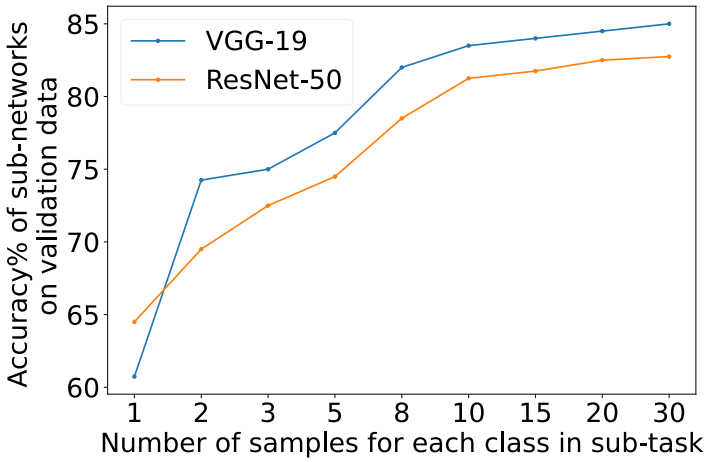


Figure 3.12: The validation-set accuracy of sub-networks vs. the number of training samples per class in each sub-task.

Effect of the Number of Training Data Fig. 3.12 shows how the sub-networks' validation-set accuracy changes when increasing the training samples per class in each sub-task. For both pre-trained VGG-19 and ResNet-50, when the number of samples per class is too small (≤ 5), the sub-network tends to be overfitting and cannot generalize well to unseen validation data. However, when the number of samples per class increases to 10, the sub-network starts to achieve promising validation-set accuracy. If further increasing the number of samples per class, the validation-set accuracy quickly saturates. Therefore, in the experiments, 10 samples per class turn out to be the sweet spot to extract sub-networks with sufficiently good generalization performance so they can be interpreted as the inference chains of the pre-trained model on given sub-tasks.

3.5 Conclusion

We propose an efficient approach called NeuroChains to extract the main inference chains/steps in a DNN when applied to a sub-task. We learn a sparse scoring of filter-

s/layers to extract a sub-network, retaining a subset of filters/layers from the original pre-trained DNN. We provide case studies and quantitative analyses to demonstrate that NeuroChains produces a more informative and reliable explanation of the inner inference process of DNNs for a given task. This approach enhances interpretability and fosters trustworthiness in DNNs. By explaining the filters and layers throughout the model, we gain insights that enable us to share knowledge with similar tasks effectively.

CHAPTER 4

RELIABLE MODEL PRUNING BY KNOWLEDGE SHARING BETWEEN TASKS

4.1 Introduction

Model pruning [29, 135, 136, 169] has been widely studied to compress neural nets by removing redundant connections and nodes. Large-scale pre-trained models are widely used in various domains [170–172]. In a variety of practical applications, a large-scale pre-trained network like ResNet-50 [162] or ViT [163] usually needs to be pruned for a wide variety of devices and adapted to different downstream tasks. Running a pruning algorithm for every device or task from the same pre-trained network can create an enormous carbon footprint overload in our biosphere and waste a lot of computational power. On the other hand, pruning a model for a new task may not be reliable if the relationship between the source and target tasks is unclear. In real-world applications, a few pre-trained models have already created thousands of pruned models for different downstream tasks. We can share the knowledge of existing pruned models to improve the

reliability and efficiency of model pruning [173]. We call this problem "meta-pruning".

In this chapter, we propose to initialize a reliable sparse network for a target task based on the pruned models of similar tasks. Meta-pruning is non-parametric if no parametric model is trained to produce the initialization. It is analogous to MAML [174] in that the meta-objective optimizes the initialization of a network. It differs from MAML in that (1) both the sub-networks architecture and weights are initialized, and (2) the initialization is not universal but task-specific. Since meta-pruning aims to find reliable sub-network initialization for new tasks, we limit the iterations during meta-pruning to strengthen the impact of initialization on the final pruned model.

The primary contribution of this work is twofold. In the first part, we conduct a thorough empirical study of the relationship between the pruned models for hundreds of tasks. No meta-pruning is studied in this part, and its primary purpose is to (1) find the nearest tasks for a new task using different similarity metrics and (2) compare the pruned models for different but similar tasks. Statistics from different scenarios indicate that similar tasks with high similarity tend to share more nodes/filters/heads preserved in their pruned models.

Motivated by the empirical study, the second part of this work proposes a reliable meta-pruning method called "meta-vote pruning (MVP)". Given a pre-trained model, MVP creates a pruned model for a new task by selecting nodes/filters/heads through majority voting among its nearest similar tasks; e.g., a filter will be sampled with a higher chance if it is selected into more pruned models of similar tasks. Given that the new task and its similar tasks share semantic relationships and the filters preserved in the pruned models are specific to the class labels of each task (as demonstrated in Chapter. 3), we can establish a reliable sub-network initialization for each new task. It can significantly reduce the pruning cost and memory required by previous pruning approaches yet still produce pruned models with promising performance. We

demonstrate these via experiments over tasks from CIFAR-100 [175], ImageNet [160], Caltech-256 [176], and several fine-grained datasets. The pruned models extracted from an ImageNet pre-trained model can also vote for tasks drawn from unseen datasets with great performance, showing the reliability and generalization of MVP.

4.2 Empirical Study: Pruning a Pre-trained Model for Different Tasks

In this section, we conduct an empirical study that applies different methods to prune a CNN or ViT pre-trained model for hundreds of tasks. Our study focuses on the overlap between the pruned models for different tasks and whether/how it relates to their similarity. To this end, we introduce different task similarities and compare the overlap associated with different similarity groups. The results show that more similar tasks tend to share more nodes/filters/heads in their pruned models. And this holds across different pruning methods, datasets, and pre-trained models. No meta-pruning is used in the study.

4.2.1 A Dataset of Pruned Models

While the number of possible downstream tasks and users can be huge in practice, the current progress on foundation models shows that one or a few large-scale pre-trained models with light finetuning usually achieve state-of-the-art (SOTA) performance on most of them. To simulate this scenario on a standard dataset, our empirical study creates a dataset of pruned models for hundreds of tasks from the same pre-trained model. We choose CIFAR-100 and ImageNet for the study due to the many classes in them. We randomly draw $1,000$ classification tasks for each dataset, each defined on 5 classes sampled without replacement. We adopt ResNet-18 [162] pre-trained on

CIFAR-100, ResNet-50 and a small version of DeiT [164] pre-trained on ImageNet. For ResNet-18 and ResNet-50, we prune two types of pre-trained models, i.e., the supervised training following [177] and the self-supervised training following SimSiam [178] (only the encoder is used). For ViT, the training of its pre-trained model follows [163].

Algorithm 2 ITERATIVE FILTER PRUNING (IFP)

Input : Pre-trained network $F(\cdot;\theta)$, Task T and training set D_T, Hyperparameters J, h, r, p

Initialize : $\Omega_\ell \leftarrow [n_\ell]$, the set of filters preserved in layer-ℓ

```
for j ← 1 to J do
    if j%h = 0 and |Ω_ℓ| > (1 − r)n_ℓ then
        for ℓ ← 1 to L − 1 do
            Prune p% of filters in Ω_ℓ with the smallest importance score over D_T;
        end
    end
    Apply one SGD step on a mini-batch of D_T to finetune the remained filters {θ_{ℓ,i} : ℓ ∈
     [L − 1], i ∈ Ω_ℓ} and θ_L;
end
```

Iterative Pruning We apply iterative filter-pruning (IFP) to ResNet. Unlike magnitude based pruning [24] with a one-time selection of nodes/weights, iterative pruning alternates between network pruning and finetuning of model weights for multiple iterations, each of which prunes $p\%$ of the remaining nodes/weights so it progressively prunes a large network to the targeted size. It usually performs better than other pruning methods and has also been mainly studied in theoretical works about Lottery Ticket Hypothesis [125]. We take the activation values of filters averaged over all training samples to measure the importance of filters [179], referred to as activation pruning, in which filters with smaller activation values contain less information about input data.

The detailed procedure of IFP is described in Alg. 2. Given a pre-trained network $F(\cdot;\theta)$ of L layers (layer-L is fully-connected) with parameter $\theta = \{\theta_\ell\}_{\ell=1:L}$ and a training set D_T of a target task T, let $\theta_\ell = \{\theta_{\ell,i}\}_{i=1:n_\ell}$ denote all parameters in layer-ℓ composed of $\theta_{\ell,i}$ for every filter-i. IFP finetunes the model for total J iterations. It prunes $p\%$ of

the filters that remain in each layer every h iteration according to their activation values $f_{\ell,i}(x)$. It stops pruning layer-ℓ if it reaches the targeted pruning ratio r.

Algorithm 3 AUTOMATIC HEAD&NODE PRUNING (AHNP)

Input : Pre-trained network $F(\cdot;\theta)$, Task T and training set D_T, Hyperparameters J, K, r, τ

Initialize : $\Omega_\ell \leftarrow [n_\ell]$, the set of heads/nodes preserved in layer-ℓ. $S_{\ell,i} \leftarrow 1$, the score for each prunable head/node in layer-ℓ.

```
for j ← 1 to J do
    for ℓ ← 1 to L − 1 do
        for i ∈ Ω_ℓ do
            Prune the head/node if its score S_{ℓ,i} < τ;
        end
    end
    Stop pruning if reaching the target pruning ratio r.
    Apply one optimization step on a mini-batch of D_T to finetune the remained heads/nodes and scores {θ_{ℓ,i}, S_{ℓ,i} : ℓ ∈ [L − 1], i ∈ Ω_ℓ} and θ_L;
end
Remove S, and finetune the pruned model for K iterations on Di.
```

Automatic Pruning Inspired by the SOTA ViT structured pruning method [36], we prune ViT by automatic head&node pruning (AHNP) for a given task, which parameterizes the sub-network as the pre-trained model with a learnable score multiplied to each prunable head and node. To encourage sparsity, the differentiable scores of all prunable heads and nodes are optimized with an additional $L1$ regularization loss. After each optimization step, we apply simple thresholding to these scores to remove heads and nodes with small scores. The optimization stops if the pruned model reaches the targeted size, and the model will be finetuned for a few iterations.

The detailed procedure of AHNP is described in Alg. 3. Given a pre-trained network $F(\cdot;\theta)$ of L layers (layer-L is fully-connected) with parameter $\theta = \{\theta_\ell\}_{\ell=1:L}$ and a training set D_T of a target task T, let $\theta_\ell = \{\theta_{\ell,i}\}_{i=1:n_\ell}$ denote all parameters in layer-ℓ composed of $\theta_{\ell,i}$ for every head/node-i. $S_{\ell,i}$ denote the score for each prunable head/node-i in layer-ℓ. AHNP finetunes the model and scores for total J iterations. It prunes the heads/nodes if their scores are smaller than the threshold τ. It stops pruning layer-ℓ if reaching the

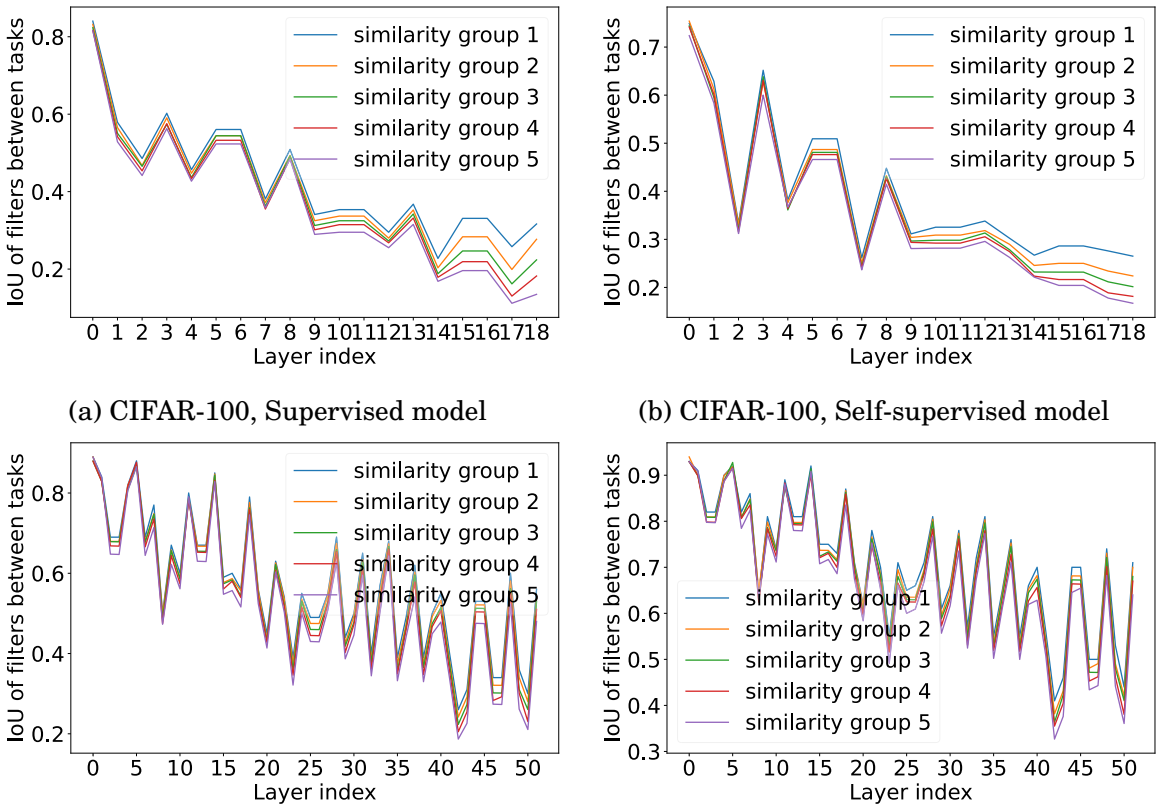
targeted pruning ratio r. Then, AHNP finetunes the pruned model for K iterations.

For tasks from CIFAR-100, we run IFP for all 1,000 tasks on ResNet-18. And we apply IFP and AHNP to tasks from ImageNet on ResNet-50 and ViT, respectively. Finally, we create a dataset of pruned models for thousands of tasks across different pre-trained models. For each task i, we record its labels C_i, the set of preserved nodes/filters/heads $\{\Omega_\ell\}_{\ell=1:L-1}$ and the pruned model θ_T. We use the same hyper-parameters for different tasks. For IFP on ResNet, we use a learning rate of 0.005, pruning iterations of 1,000 and batch-size of 128 for both the tasks of CIFAR-100 and ImageNet. When applying AHNP to ViT, we follow the ViT training in [164]. We reduce the pruning iterations to 1,000 and use a small learning rate of 0.00005 for parameters inherited from the pre-trained ViT (to preserve its knowledge) and a large learning rate of 0.05 for the learnable scores. The pruning ratio is 90% for all pruned models.

In the experimental setting, there is a one-time cost for preparing the model zoo. However, like training any meta-learning model, this can facilitate many future tasks by significantly reducing their computation and required samples. Moreover, we can keep adding MVP-pruned models of new tasks into this model zoo and improving it in a life-long learning manner with no extra cost. Meta-pruning moves required computation from new-task adaptation to pre-training (preparing the model zoo). In practice, this is an even more important advantage over single-task pruning because the meta-pruning cost is offline on the server side, so it is tolerable. Practitioners care more about the deployment cost of new tasks, e.g., on-edge devices with limited data and computation. Because of the model zoo, our method makes the practical deployment of model pruning more affordable.

4.2.2 Relationship between Pruned Models of Different Tasks

The representations learned for a task can be shared by similar tasks. But do similar tasks also share more structures in their pruned sub-networks? We apply two met-



(a) CIFAR-100, Supervised model

(b) CIFAR-100, Self-supervised model

(c) ImageNet, LEEP Similarity

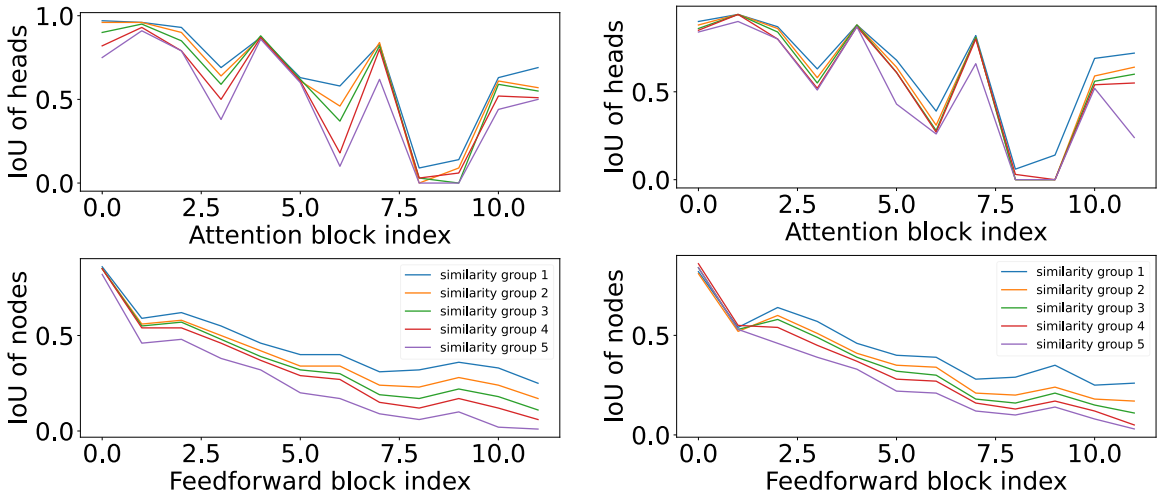
(d) ImageNet, Wordnet Similarity

Figure 4.1: IoU of layers in ResNet between tasks with different similarities.

rics to measure the similarity between classification tasks in our dataset and study whether/how the similarity relates to their shared nodes/filters/heads in different layers of their pruned models.

Similarity Metrics We apply two metrics to compute the similarity between tasks and find the nearest tasks, i.e., the Log Expected Empirical Prediction (LEEP) [180] and the Wordnet wup similarity [181, 182]. LEEP score is widely used in transfer learning to estimate the knowledge transferability from a source task to a target task. In our study, for each target task, we can rank the other tasks by their LEEP similarity score to the target one. Computing the LEEP score only requires a single forward pass of the pruned model on the target task's data. Wordnet wup similarity leverages the semantic labels of classes in each task. It is calculated based on the depths of their corresponding

synsets in the Wordnet [183] taxonomies. Since it does not rely on the pruned model, it is computationally efficient. Moreover, these similarity metrics contribute to the reliability of our proposed approaches as they capture the semantic relationships between tasks.



(a) ImageNet, LEEP Similarity

(b) ImageNet, Wordnet Similarity

Figure 4.2: IoU of blocks in ViT between tasks with different similarities.

Overlap Between Tasks Let Ω_ℓ^i and Ω_ℓ^j denote the sets of filters/nodes/heads remained in layer-ℓ after running IFP or AHNP for task i and j (when using the same pre-trained model), we measure the overlap of the two sets by intersection over union (IoU) ratio [184], i.e., $IoU = |\Omega_\ell^i \cap \Omega_\ell^j| / |\Omega_\ell^i \cup \Omega_\ell^j|$. Fig. 4.1 (ResNet) and Fig. 4.2 (ViT) report the IoU of each layer/block for pairs of tasks with different similarities. For each target task, the tasks in the dataset are partitioned into 5 similarity groups according to their LEEP scores or Wordnet similarities to the target task. The similarity decreases from group 1 to group 5. Specifically, for each test task, we compute its similarity scores with its neighbours in the model zoo. We sort these similarity scores and partition them into five groups with equal intervals. Neighbours whose similarity scores fall into a certain interval will be assigned to the corresponding group.

For all of the datasets and architectures, more similar tasks tend to share more filters/nodes/heads (larger IoU) between their pruned models. Therefore, for a new task,

the pruned models of its nearest tasks preserve many important filters for it, and combining them might result in a better and much smaller sub-network to initialize the new task. Moreover, for deeper layers/blocks in both ResNet and ViT, the gap between different similarity groups on the IoU increases because the features are more task-specific in deeper layers. Due to the same reason, for every similarity group, IoU decreases with depth in the overall trend (though fluctuating locally). Furthermore, Fig. 4.2 shows that the IoU gap between similarity groups defined by the LEEP score is larger than that obtained by Wordnet similarity. This indicates that the semantic similarity between class labels might not be as accurate as the LEEP score, which takes the pruned model and its learned representations into account.

Algorithm 4 META-VOTE PRUNING (MVP)

Input : Target task i and its training set D_i, pruning ratio r, J, N, a dataset of pruned models for different tasks
Output : A pruned model for target task-i
Initialize : $\Omega_\ell \leftarrow \emptyset$, the set of filters in layer-ℓ

```
Sample/find N similar tasks N^i to task i according to LEEP score or Wordnet similarity;
for ℓ ← 1 to L − 1 do
    Sample (1 − r)n_ℓ filters with probability p(k) (Eq. (4.1)) and add them to Ω_ℓ;
    for k ∈ Ω_ℓ do
        Initialize filter-k by averaging its parameters of tasks in {j ∈ N^i : k ∈ Ω_ℓ^j};
    end
end
finetune the pruned model for J iterations on D_i.
```

4.3 Meta-Vote Pruning (MVP)

4.3.1 Implementation Details

Inspired by the empirical study above, we propose a simple yet strong baseline "meta-vote pruning (MVP)" (Alg. 4) for non-parametric meta-pruning. The procedure of MVP majority voting is shown in Fig. 4.3. Given a target task i, MVP draws a sub-network of a

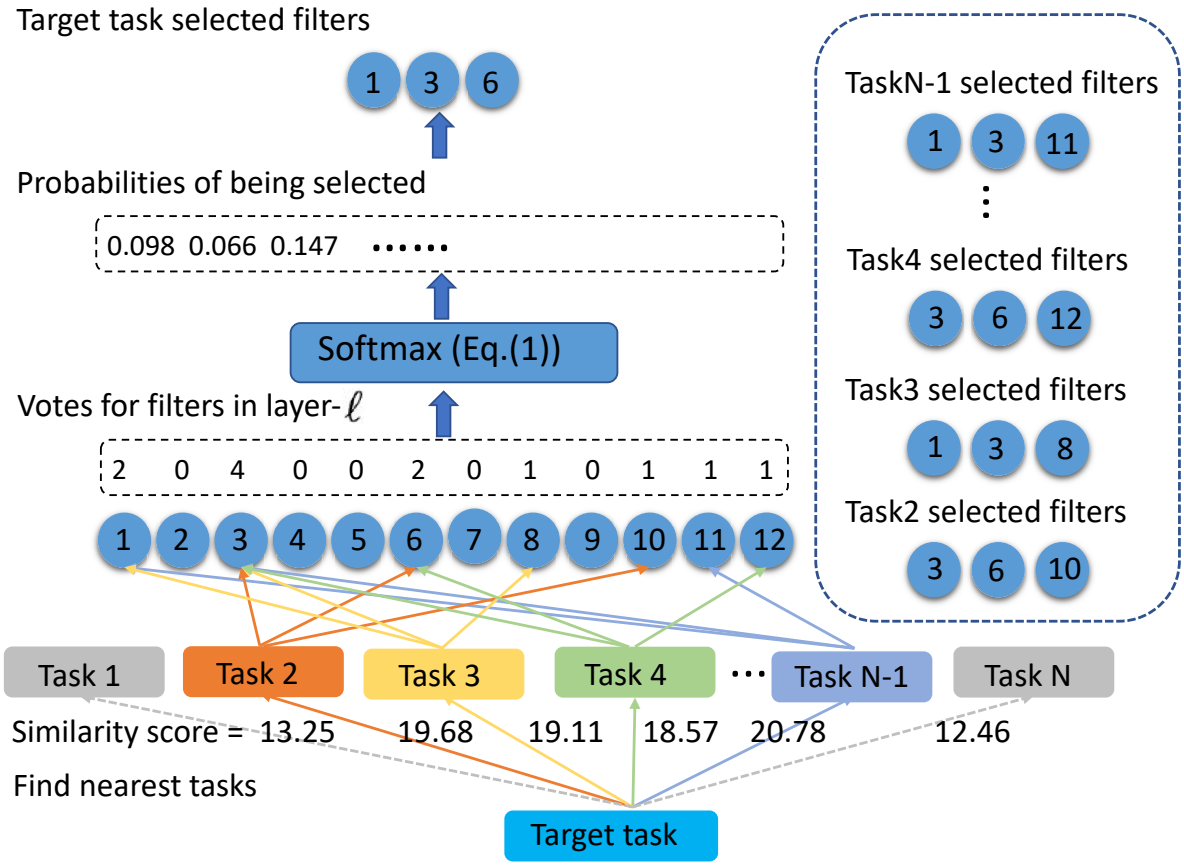


Figure 4.3: The example of majority voting in MVP. Each similar neighbour task of the target task votes for filters that are reserved by its pruned model. Then, softmax is applied to the votes of all filters in layer-ℓ and filters with more votes have a higher probability to be selected by the target task.

pre-trained network by sampling filters/nodes/heads in each layer using majority voting from its nearest tasks N^i and their pruned models. In particular, for each filter-$k \in [n_\ell]$ from layer-ℓ of the pre-trained model, we apply softmax (with temperature τ) to the times of each filter being selected by tasks in N^i, which yields a probability distribution over all the filters $[n_\ell]$, i.e., $\forall k \in [n_\ell]$,

$$p(k) = \frac{\exp(|\{j \in N^i : k \in \Omega_\ell^j\}|/\tau)}{\sum_{h \in [n_\ell]} \exp(|\{j \in N^i : h \in \Omega_\ell^j\}|/\tau)} \tag{4.1}$$

To initialize layer-ℓ of the sub-network, MVP samples filters from this distribution (without replacement) according to the targeted pruning ratio r. We further initialize the parameters of each filter-k by averaging its parameters in the pruned models of

similar tasks that preserve filter-k. MVP then finetunes the initialized sub-network for a few iterations on the training set of the target task since MVP targets keep the computational cost low.

4.4 Experiments

In this section, we conduct extensive experiments on CIFAR-100 and ImageNet over different pre-trained models, which evaluate MVP (Alg. 4) and compare it with SOTA pruning methods under different settings. We validate the strong generalization of MVP by applying it to unseen tasks from Caltech-256 and other fine-grained datasets. We further study the effect of different pruning iterations, neighbour numbers, task sizes, and similarity metrics on MVP. All the results show that MVP can outperform other methods with better performance and higher efficiency.

The experiments of MVP are mainly based on the tasks from the dataset introduced in Sec. 4.2.1. For each setting of experiments, we randomly draw 100 test tasks (i.e., the target task in Alg. 4) from the dataset and treat the rest tasks as training tasks. To evaluate MVP on CNN, we run MVP on the pruned models of ResNet-18 and ResNet-50 for CIFAR-100 and ImageNet respectively. For both of these two experiments, we use the meta-pruning iterations of 100, batch size of 128, the learning rate of 0.01 and an optimizer of SGD with the cosine-annealing learning rate schedule. For experiments with ViT, MVP is applied to the pruned models of ViT for ImageNet. The meta-pruning iterations and batch size are also set as 100 and 128 respectively. Following the setting of training ViT in [164], we apply a small learning rate of 0.0002 and the optimizer of AdamW with the cosine-annealing learning rate schedule. The small number of meta-pruning iterations demonstrates the efficiency of MVP. The target pruning ratio of MVP for all tasks is 90%. All the results of accuracy shown in this section are averaged over the 100 test tasks.

4.4.1 Baseline Methods

We compare MVP with several baselines and SOTA pruning methods. We first implement two baselines to show the advantages of MVP. (1) Conventional pruning. We apply a larger number of pruning iterations to extract pruned models for each target task by IFP or AHNP introduced in Sec. 4.2.1. This baseline can be regarded as the upper bound of performance. (2) Random pruning. To validate whether the initialization of MVP makes sense, for each target task, we initialize its sub-network by randomly sampling the same number of nodes/filters/heads as MVP from the pre-trained model. We take this baseline as the lower bound of performance.

We also include other SOTA pruning methods. For MVP on CNN, we compare MVP with IHT-based Reptile [136], a meta-pruning method that uses Reptile [185] and iterative pruning to find better weight-initialization for a pruned meta-model. Given a new task, it finetunes the pruned meta-model for a limited number of iterations to obtain the final pruned model. MEST [186] is the SOTA method in the sparse training community, which trains a model from a sparse sub-network so that less computation is required. DLTH [124] is based on a variant of the Lottery Ticket Hypothesis. It transforms random tickets into winning tickets. We compare MVP with UVC [36] and PoWER [131] on ViT pruning. Unlike AHNP, which prunes heads and nodes, UVC also skips the unimportant layers and blocks in ViT. Unlike parameter pruning, PoWER adopts a dynamic method for pruning the input patches of each block for each input sample. For a fair comparison, except for the upper bound baseline, the pruning iterations of all other methods and MVP are set to 100. And the pruning ratios of all methods are set to 90%.

4.4.2 Main Results

The results of applying MVP to tasks from CIFAR-100(ImageNet) on ResNet-18(ResNet-50) supervised and self-supervised pre-trained model, and the baseline methods are

Table 4.1: Comparison between MVP and baseline methods on CNN. The '-SSL' behind each method means applying this method to pruned models extracted from self-supervised pre-trained models. **Bold** and **Bold gray** mark the best and second best accuracy, respectively.

Methods	**Pruning Iterations**	**ResNet-18**		**ResNet-50**	
		Acc	FLOPs	Acc	FLOPs
IFP	1,000	**87.99±0.47**	14.88(T)	**91.16±0.68**	110.06(T)
IFP-SSL	1,000	85.22±0.52	14.88(T)	85.84±0.75	110.06(T)
Random Pruning	100	33.12±6.47	0.43(T)	22.42±3.92	3.16(T)
IHT-based Reptile[136]	100	75.23±0.87	0.43(T)	73.40±0.75	3.16(T)
MEST[186]	100	76.28±0.82	0.47(T)	66.25±2.33	3.48(T)
DLTH[124]	100	74.46±1.24	4.28(T)	69.33±1.56	31.64(T)
MVP(**ours**)	100	**88.98±0.38**	0.43(T)	**91.80±0.26**	3.16(T)
MVP-SSL(**ours**)	100	86.82±0.13	0.43(T)	85.92±0.26	3.16(T)

reported in Tab. 4.1. On both datasets and pre-trained models, MVP outperforms IFP which spends 10 × iterations of MVP. Hence, MVP can produce a higher-quality pruned model when using fewer iterations. The results demonstrate that MVP can work well on tasks from both supervised and self-supervised pre-trained models. The random pruning performs much poorer than MVP, which indicates the importance of majority voting from the nearest tasks in selecting filters.

We also compare MVP with SOTA pruning methods for CNN. IHT-based Reptile [136] trains a universal sparse sub-network for all target tasks by applying meta-learning to training tasks. MVP achieves higher accuracy than IHT-based Reptile under the same training iterations, implying that MVP can find an accurate sub-network for each target task as its initialization and improve its performance. MEST [186] can speed up pruning by starting training from a well-designed sub-network. As a variant of Lottery Ticket Hypothesis, DLTH [124] proposes a method to transform any random ticket into the winning ticket. MVP outperforms MEST and DLTH by a large margin because MVP is trained on a sub-network selected using meta-knowledge from similar tasks. In contrast, the initial sub-network for MEST or the winning ticket for DLTH does not leverage any prior knowledge about the target task.

Table 4.2: Comparison between MVP and baseline methods on ViT. **Bold** and **Bold gray** mark the best and second best accuracy

Methods	Pruning Iterations	ViT	
		Acc	FLOPs
AHNP	1,000	**89.48±0.62**	81.50(T)
Random Pruning	100	58.71±4.14	3.25(T)
UVC[36]	100	80.30±0.57	26.73(T)
PoWER[131]	100	77.76±1.18	20.86(T)
MVP(**ours**)	100	**89.23±0.49**	3.25(T)

Tab. 4.2 shows the comparison between MVP and baseline methods on ViT. Similar to the results on pruning CNN, the performance of MVP on ViT is comparable to AHNP which applies much more pruning iterations. The accuracy of random pruning is still much worse. MVP also outperforms SOTA pruning methods developed for ViT. Hence, on ViT, MVP can efficiently produce a small yet high-quality sub-network for each new task by exploiting the nearest tasks' models. The baselines are slower and require more iterations than MVP because they need to retrain the model to achieve a small loss when some parameters or patches are removed. Both UVC and PoWER cannot recover accuracy under this strong constraint. In contrast, the majority voting in MVP directly produces a small sub-network from similar tasks' models, so only a few iterations suffice to reach a downstream task performance comparable to AHNP with 10x iterations.

4.4.3 Performance on Unseen Dataset

In this section, to validate the reliability generalization of MVP, we apply MVP to produce pruned models for target tasks from the unseen dataset Caltech-256 [176] and fine-grained dataset CUB200-2011 [187], Oxford Flowers-102 [188] and Oxford-IIIT Pets [189], using the pruned models of tasks from ResNet-50 training on ImageNet. The data of these datasets is never seen by the pre-trained model or tasks in the pruned model dataset. Each target task is defined in 5 classes sampled without replacement

from each dataset.

Table 4.3: Accuracy of MVP on unseen tasks.

Methods	**Caltech-256**		
	Iters	Acc	FLOPs
IFP	800	80.28±1.64	93.06(T)
IFP	60	42.90±3.79	6.73(T)
MVP(**ours**)	60	**80.72±0.64**	1.90(T)

The performance of MVP on Caltech-256 is shown in Tab. 4.3, which is still comparable to the IFP using 10x pruning iterations. When the number of pruning iterations of IFP decreases, its performance becomes much worse. Besides Caltech-256, we also validate the effectiveness of MVP on more difficult fine-grained datasets, i.e., CUB200-2011, Oxford Flowers-102 and Oxford-IIIT Pets where images in different classes are from various species of birds, flowers, and animals, which are hard to distinguish. On target tasks from fine-grained datasets, MVP also works better than IFP which requires much higher computation costs. Results can be found in Tab. 4.4.

The results show that MVP can still produce a high-quality initialization for the task from unseen datasets by majority voting of similar tasks so that the pruned model can converge quickly with high accuracy. MVP's great performance on fine-grained datasets implies that MVP can learn from different objects to facilitate the classification of hard-to-distinguish target tasks. This experiment demonstrates that MVP is reliable to various datasets and generalizes well.

Table 4.4: Accuracy of MVP on fine-grained tasks.

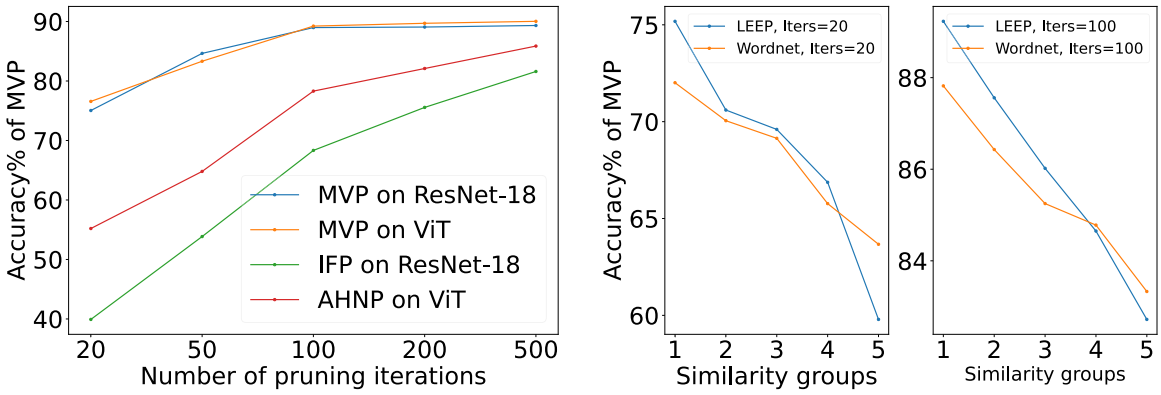
Methods	**CUB200-2011**			**Oxford Flowers-102**			**Oxford-IIIT Pets**		
	Iters	Acc	FLOPs	Iters	Acc	FLOPs	Iters	Acc	FLOPs
IFP	800	77.09±0.51	23.99(T)	800	95.20±1.39	47.98(T)	1,000	77.38±0.96	59.33(T)
IFP	80	52.85±2.95	2.17(T)	60	54.20±6.61	3.04(T)	100	55.76±3.86	4.57(T)
MVP(**ours**)	80	**79.54±0.88**	0.63(T)	60	**95.40±1.34**	0.95(T)	100	**78.29±0.77**	1.58(T)

4.4.4 Results of MVP on Sub-tasks of Different Sizes

To evaluate the reliability of MVP on sub-tasks of different sizes, we build a dataset of pruned models for 10-classification and 3-classification sub-tasks from CIFAR-100, of which the pruning ratio is set to 85% and 95% respectively. The results are shown in Tab.4.5. From the results, we can find that when changing the size of the sub-tasks, MVP can consistently achieve comparable or better performance than SoTA methods by spending much less computation. MVP is applicable and reliable to a variety of tasks of different sizes.

Table 4.5: Results on sub-tasks of different sizes for CIFAR-100.

Methods	10-**classification**			3-**classification**		
	Iters	Acc	FLOPs	Iters	Acc	FLOPs
IFP	1,500	84.29±0.26	25.48(T)	500	88.75±0.71	5.59(T)
MVP(**ours**)	190	83.53±0.34	1.21(T)	60	89.25±0.23	0.12(T)



(a) Effect of Pruning Iterations

(b) Effect of Similarities between Tasks

Figure 4.4: **(a)** Comparison between MVP and conventional pruning methods with different pruning iterations on different architectures. For both ResNet-18 and ViT, MVP converges much faster in a small number of iterations than conventional pruning methods. **(b)** Comparison between LEEP score and Wordnet similarity for MVP with different pruning iterations. From similarity groups 1 to 5, the similarities between tasks decrease. For both similarity metrics, more similar tasks get better performance. LEEP score has a better ability to measure similarities between tasks than Wordnet similarity.

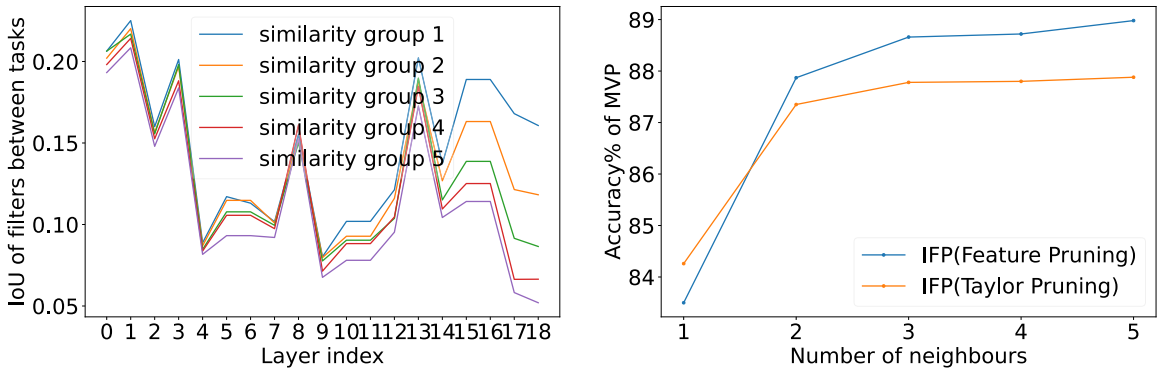
4.5 Ablation Studies

Effect of Iteration Numbers

Given a new target task and a pre-trained model, MVP can build a well-performing small model in a few iterations, demonstrating its capability to reduce adaptation costs. In plot (a) of Fig. 4.4, we compare MVP with conventional pruning methods using different numbers of iterations. MVP converges to high accuracy on different architectures of pre-trained models after nearly 100 iteration. On the contrary, the conventional pruning methods need much more iterations(> 500) to be comparable to MVP. MVP can reach a reasonable accuracy with only ≤ 50 pruning iterations, while conventional pruning methods perform poorly. These imply that the initialized sub-network obtained by majority voting already contains helpful knowledge from similar tasks to speed up the training of the pruned model.

Effect of Similarities between Tasks MVP consistently performs better when applied to the nearest tasks with the highest similarities. In plot (b) of Fig. 4.4, we compare the LEEP score with the Wordnet similarity and study the effect of applying MVP to neighbour tasks with different similarities. From similarity group 1 to group 5, the similarities between tasks decrease. We find that for both the two similarity metrics, the accuracy of MVP improves significantly when the similarities between tasks increase. When the pruning iterations are small($= 20$), where the initialization of the sub-network is more important, the accuracy of tasks from similarity group 1 leads to similarity group 5 by 15%. Despite the accuracy of similarity group 5 improving when the pruning iterations increase to 100, there is still a gap of 7%. This result indicates that neighbour tasks with high similarities share more knowledge with the target task. In this plot, we also find that tasks in different similarity groups classified by LEEP score show larger differences than Wordnet similarity, implying that LEEP score can better evaluate similarities between tasks. This result is consistent with our observation in the empirical

study. The performance of Wordnet similarity is also good and can still be an alternative when time and computational resources are limited.



(a) IoU of Layers for IFP(Taylor Pruning)

(b) Effect of Neighbour Numbers

Figure 4.5: **(a)** IoU of layers in ResNet-18 between tasks whose pruned models are extracted by IFP (Taylor Pruning) and more similar tasks also share more filters, especially in deeper layers. **(b)** Results of applying MVP to pruned models from Activation Pruning and Taylor Pruning over different numbers of neighbours. MVP(neighbour number ≥ 2) can improve transfer learning performance (neighbour number = 1) by a large margin when applied to pruned models extracted by different pruning methods.

Comparison between Pruned Models Extracted by Different Pruning Method

In this part, we apply MVP to pruned models extracted by Taylor Pruning [190] on ResNet-18 for CIFAR-100 tasks to prove that MVP works well on pruned models extracted by various pruning methods. Taylor Pruning measures the importance of each filter by the effect of removing it on the final loss. In plot (a) of Fig. 4.5, we show the IoU of each layer for pairs of tasks with different task similarities, of which the pruned models are extracted by Taylor Pruning. Consistent with our observation in the empirical study, pruned models with higher similarities share more filters.

Effect of Number of Neighbours In plot (b) of Fig. 4.5, we investigate the effect of the number of neighbours on MVP. When the number = 1, MVP reduces to transfer learning, which learns from the pruned model of a single selected similar task. In the plot, the performance improves sharply when the number of neighbours increases from 1 to 2. This result implies the effectiveness of meta-knowledge from different neighbours.

When the number of neighbours ≥ 3, for both Activation Pruning and Taylor Pruning, the accuracy improves little, which indicates that 3 neighbours are enough for MVP to produce a high-quality initialization.

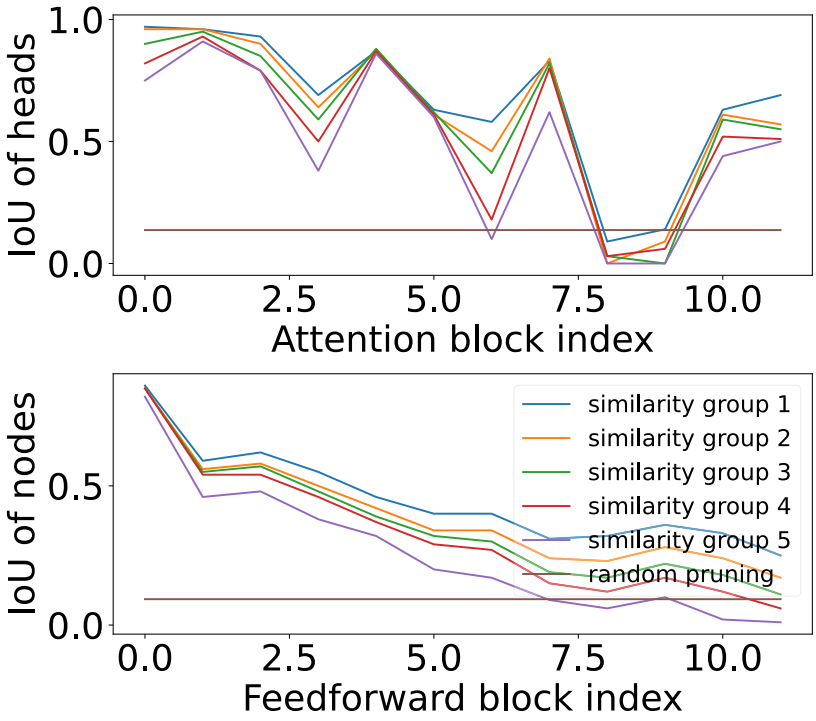


Figure 4.6: Comparison of IoU between MVP and random pruning.

Comparison of IoU between MVP and random pruning In Fig.4.6, besides MVP, we also show the IoU of pruned models extracted by random pruning. When the similarity between tasks is large, the IoU of MVP is much larger than random pruning, implying that these tasks contain lots of relevant information. When the similarity between tasks is small, in the last few blocks, the IoU of nodes is similar to that of random pruning, which indicates that tasks of low similarity share little high-level information with the target task.

4.6 Conclusion

In this chapter, we study the "non-parametric meta-pruning" problem that aims to reduce the memory and computational costs of single-task pruning by sharing the knowledge of similar tasks' pruned models. We conduct an empirical study investigating the relationship between the pruned models of different yet similar tasks and observe that more similar tasks share more parameters in the pre-trained model. The empirical study motivates a method called "meta-vote pruning (MVP)" (Alg. 4), which creates a well-performing initialized pruned for a target task by voting from the pruned models of similar tasks. By extensive experiments on multiple tasks drawn from several datasets under different training settings, we demonstrate the advantages of MVP over other SOTA pruning methods in efficiency and accuracy. Besides, MVP is reliable for tasks with unseen data and of different sizes.

CHAPTER 5

DEBIASED KNOWLEDGE SHARING ACROSS SEQUENTIAL TASKS

5.1 Introduction

DNNs have achieved unprecedented promising performance on challenging tasks under the i.i.d. offline setting where data is treated fairly. However, its reliability and performance degenerate drastically in continual learning (CL) where tasks are introduced sequentially, as the model quickly adapts to a new task and overwrites the previously learned weights. This leads to a severe bias toward more recent tasks and compromises fairness across sequential tasks.

A frequently employed strategy to mitigate forgetting is the use of experience replay (ER) [40, 41] and its variants [1, 139, 191]. These methods store a few data from previous tasks in a limited memory buffer and train the model using current and buffered data. However, this approach involves replaying buffered data at every step, which can be time-consuming and computationally expensive. Another approach [45, 192] constrains

the change in parameters that are critical to previous tasks. However, imposing such extra constraints may degrade the training performance and discourage task-agnostic modules from capturing shared knowledge across tasks. Despite these efforts, both approaches still exhibit significant bias across tasks.

In this chapter, we aim to mitigate the bias in CL by knowledge sharing to promote fairness in deep models, which is crucial to reach the balance between various tasks [193]. Additionally, we strive to enhance the efficiency of the training process of CL. we study a fundamental but open problem, i.e., are most parameters task-specific and sensitively changing with the distribution shift? Or is the bias mainly caused by the change in a few task-specific parameters? Additionally, how to balance task-specific and task-agnostic parameters so that the model can achieve promising performance on new tasks while facilitating knowledge sharing across tasks. Finally, whether the practice of the time-consuming, every-step replay is truly necessary.

To answer these questions, we investigate the training dynamics of model parameters during the course of CL by measuring their changes over time. For different CL methods training on different neural networks, we consistently observe that only a few parameters change more drastically than others between tasks. The results indicate that most parameters can be shared across tasks, and we only need to finetune a few task-specific parameters to encourage knowledge sharing and reduce bias.

The empirical studies immediately motivate an effective method, "forgetting prioritized finetuning (FPF)" which finetunes the task-specific parameters using buffered data at the end of CL methods. Surprisingly, on multiple datasets, FPF consistently improves several widely-studied CL methods and substantially outperforms various baselines. Moreover, we extend FPF to a more efficient replay-free CL method, "k-FPF", that entirely eliminates the cost of every-step replay by replacing such frequent replay with occasional FPF. k-FPF can achieve comparable performance with FPF+ state-of-the-art

(SOTA) CL methods and significantly reduce the computational cost. The exceptional performance of FPF throughout the training process of CL demonstrates its ability to balance sequential tasks and promote fairness in CL models.

5.2 Problem Setup

Notations We consider the CL setting, where the model is trained on a sequence of tasks indexed by $t \in \{1, 2, \ldots, T\}$. During each task t, the training samples (x, y) (with label y) are drawn from an i.i.d. distribution D_t. Given a neural network $f_\Theta(\cdot)$ of L layers with parameter $\Theta = \{\theta_\ell\}_{\ell=1:L}$, $\theta_\ell = \{\theta_{\ell,i}\}_{i=1:n_\ell}$ denote all parameters in layer-ℓ where $\theta_{\ell,i}$ denotes parameter-i. On each task, $f_\Theta(\cdot)$ is trained for N epochs. We denote all parameters and the layer-ℓ's parameters at the end of the n-th epoch of task t by Θ_n^t and $\theta_{\ell,n}^t$, $n \in \{1, \ldots, N\}$, respectively.

Settings In this work, we mainly focus on class-incremental learning (class-IL) and domain-incremental learning (domain-IL). In class-IL, D_t are drawn from a subset of classes C_t, and $\{C_t\}_{t=1}^T$ for different tasks are assumed to be disjoint. Class-IL is a more challenging setting of CL[194] than task-incremental learning (task-IL) [195]. Unlike task-IL, class-IL cannot access the task label during inference and has to distinguish among all classes from all tasks. In domain-IL, tasks to be learned remain the same, but the domain varies, i.e. the input data distribution D_t changes. The model is expected to adapt to the new domain without forgetting the old ones. The goal of the class-IL and domain-IL is: $\min_\Theta L(\Theta) \triangleq \sum_{t=1}^T \mathbb{E}_{(x,y)\sim D_t}[l(y, f_\Theta(x))]$, where l is the objective function.

Datasets We conduct class-IL experiments on Seq-MNIST, Seq-OrganAMNIST, Seq-PathMNIST, Seq-CIFAR-10, and Seq-TinyImageNet. Seq-OrganAMNIST and Seq-PathMnist are generated by splitting OrganAMNIST or PathMNIST from MedMNIST [196], a medical image classification benchmark. CL on medical images is essential in practice but also challenging since medical images always come as a stream with new

patients and new diseases. Moreover, medical images of different classes might only have subtle differences that are hard to distinguish. Both Seq-OrganAMNIST and Seq-PathMnist consist of 4 disjoint classification tasks. The number of classes per task in Seq-OrganAMNIST and Seq-PathMnist are [3, 3, 3, 2] and [3, 2, 2, 2], respectively. Seq-MNIST (Seq-CIFAR-10) are generated by splitting the 10 classes in MNIST [197] (CIFAR-10 [175]) into five binary classification tasks. Seq-TinyImageNet partitions the 200 classes of TinyImageNet [198] into 10 disjoint classification tasks with 20 classes per task. For domain-IL experiments, we use PACS dataset [199], which is widely used for domain generalization. It can present a more realistic domain-shift challenge than the toy-setting of PermuteMNIST [44]. Images in PACS come from seven classes and belong to four domains: Paintings, Photos, Cartoons, and Sketches. In Seq-PACS for CL, each task only focuses on one domain, and the sequence of tasks is Sketches → Cartoons → Paintings → Photos (increasing the level of realism over time) [200].

Models We follow the standard network architectures adopted in most previous CL works. For Seq-MNIST, following [139], we employ an MLP, i.e., a fully-connected (FC) network with two hidden layers, each composed of 100 ReLU units. Following [201, 202], we train ResNet-18 [162] on other five datasets. In addition, we also extend our empirical study to another architecture, i.e., VGG-11 [161] on Seq-CIFAR-10.

5.3 Forgetting of Different Parameters: An Empirical study

A fundamental and long-lasting question in CL is how the distribution shift impacts different model parameters and why it leads to harmful forgetting and bias. Its answer could unveil the plasticity-stability trade-off in CL, where some parameters are plastic and task-specific and thus have to be finetuned before deploying the model, while the

stable ones can be shared with and generalized to new tasks. In order to answer this question, we conduct a comprehensive empirical study that compares the training dynamics of different parameters in three widely studied neural networks.

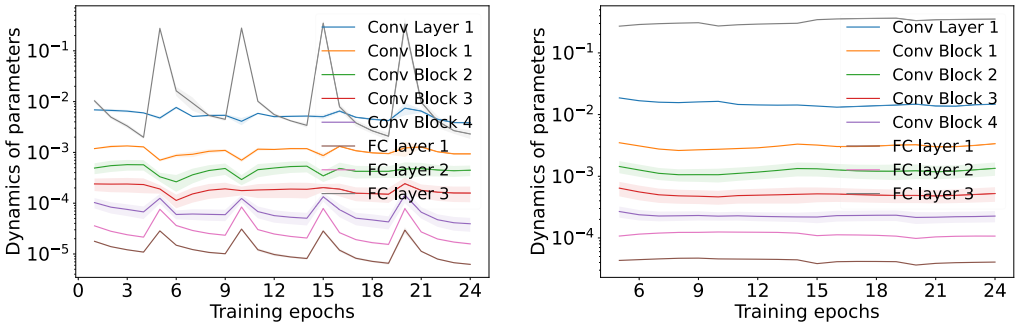
5.3.1 Measuring Forgetting via Training Dynamics

To measure and compare the forgetting effects of different parameters, we adopt two intuitive metrics to compute the change of parameters and investigate their dynamics over CL. The first metric calculates the difference between two consecutive epochs, e.g., for parameter θ_ℓ, it computes (1) $(1/|\theta_\ell|)\|\theta^t_{\ell,n} - \theta^t_{\ell,n-1}\|_1$ between epoch-$(n-1)$ and epoch-n within a task-t and (2) $(1/|\theta_\ell|)\|\theta^{t+1}_{\ell,1} - \theta^t_{\ell,N}\|_1$ between the last epoch of task-t and the first epoch of task-$(t+1)$. The training dynamics of this metric on different groups of parameters for different networks are shown in plots (a,c) of Fig. 5.1. In CL, the unstable changes of parameters are mainly caused by the task shift, while the learning within each task usually leads to smooth changes. Hence, the second metric focuses on the difference between two consecutive tasks, e.g., the change of parameters between epoch-n of the two tasks, i.e., $C_\ell = (1/|\theta_\ell|)\|\theta^{t+1}_{\ell,n} - \theta^t_{\ell,n}\|_1$. Its results on different neural networks are displayed in plots (b,d,e) of Fig. 5.1.

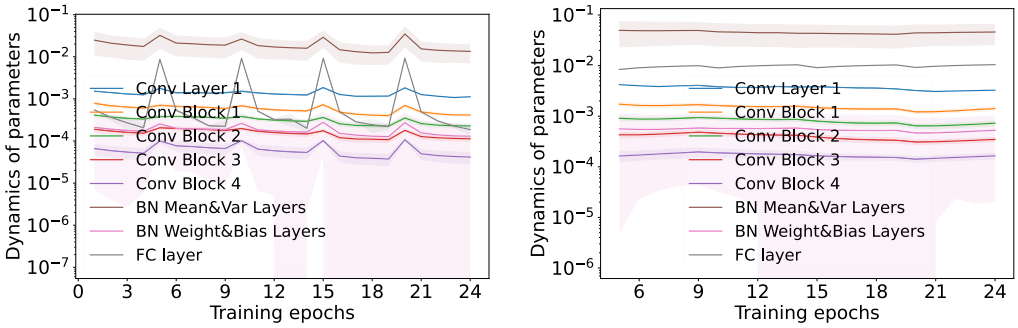
5.3.2 Forgetting of Different Parameters During CL

We first investigate and compare the training dynamics of different parameters in three types of networks. To gain insights applicable to all CL methods, we exclude any specific CL techniques but simply apply SGD to train a model on a sequence of tasks without any countermeasure to forgetting. Then, we extend the experiment to different CL methods and datasets to verify whether the observations still hold.

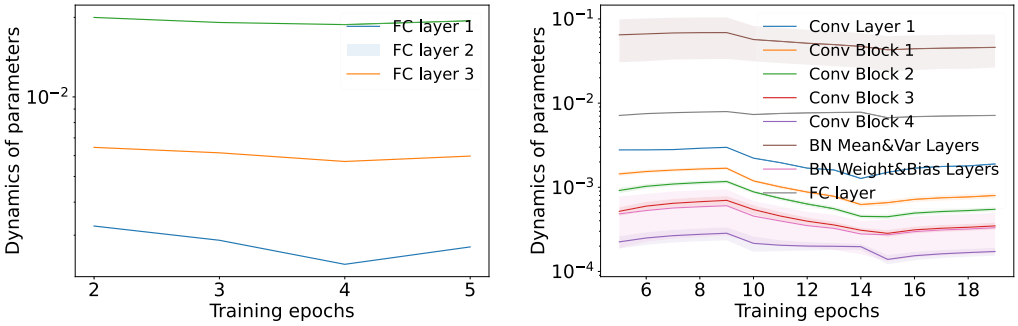
Dynamics between Consecutive Epochs Plots (a,c) of Fig. 5.1 show the training dynamics of consecutive epochs for parameters in VGG-11 and ResNet-18 when trained



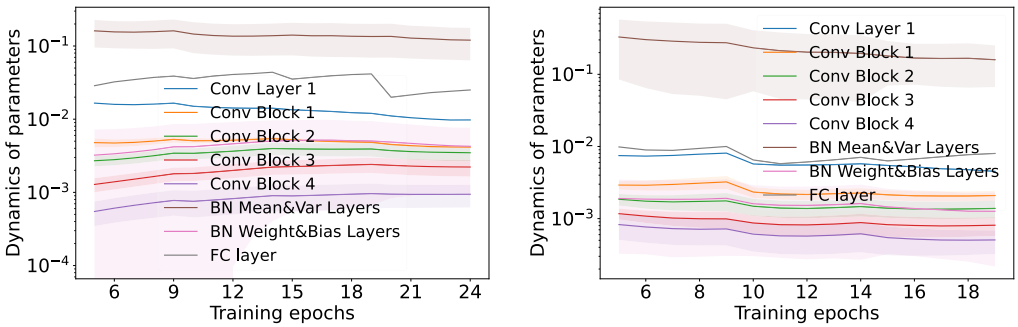
(a) VGG(consecutive epochs) (b) VGG(consecutive tasks)



(c) ResNet(consecutive epochs) (d) ResNet(consecutive tasks)



(e) MLP (f) Seq-OrganAMNIST



(g) ER(buffer size= 2000) (h) Seq-PACS

Figure 5.1: (a-e) The training dynamics of different metrics for different groups of parameters when applying SGD in CL to train three types of deep neural networks; The training dynamics of other scenarios in ResNet-18: (e) on a non-standard dataset; (g) using a different CL method with a different buffer size; (h) in the domain-IL setting. Note the y-axis is of logarithmic scale.

on Seq-CIFAR10. We partition all parameters in VGG-11 into several groups, i.e., the bottom convolutional layer (closest to the input), convolutional layers in different blocks, and three FC layers. Besides the groups of VGG-11, ResNet-18 applies batch-normalization (BN), which has two groups of parameters, i.e., (1) weights and biases and (2) mean and variance. In the plots, all parameters experience more changes at the epoch of task switching and quickly converge after a few epochs in the same task. Hence, the dynamic patterns of this metric can be used to detect task boundaries.

Dynamics between Consecutive Tasks Plots (b,d,e) of Fig. 5.1 show the training dynamics of consecutive tasks for parameters in VGG-11, ResNet-18 and MLP. We train a three-layer MLP for Seq-MNIST. Since each task in Seq-MNIST is trained only 1 epoch, the dynamics of MLP for consecutive epochs and consecutive tasks are the same. From the plots of different neural networks, the last FC layer is more sensitive to task shift than other layers. When BN is included in the network, BN layers' mean and variance become the most changed parameters. These observations are similar to studies in research domains like multi-task learning and domain adaptation [58, 203] that the last FC layer and BN layers are task-specific and cannot be shared among tasks. In CL, the last FC layer is sensitive because tasks in class-IL differ on their predicted classes, which are the outputs of the last FC layer. It is also intuitive that BN layers are task-specific since the mean and variance of BN layers capture the first and second-order moments of the distribution for the latent representations. The variance of BN weight and bias is relatively large compared to other layers.

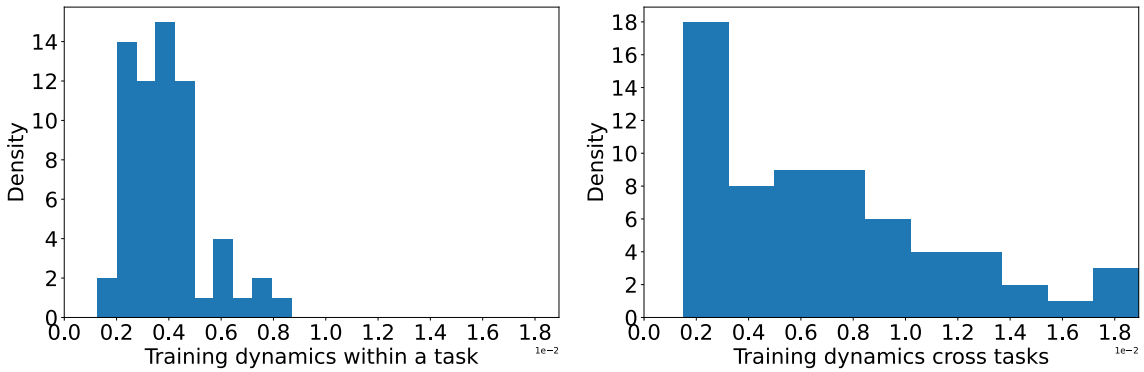
One interesting observation of VGG-11 and ResNet-18 is that the sensitivity of convolutional layers increases as the layer gets closer to the input. The reason may be that they produce representations for the input images, whose distribution shift directly impacts the bottom convolutional layer. The functionality of filters in the top layers is to integrate patterns learned in the bottom layers to produce high-level patterns, so filters

in the top layers are relatively stable. In Fig. 5.2, we further study the training dynamics of each filter within a task or cross tasks in different layers of a network. Firstly, the training dynamics of each filter in the bottom layer are much larger than that of the top layer, which aligns with our above observation. We also find that in the same layer, when tasks shift, the dynamics of a small number of filters increase significantly. These filters should be task-specific and critical to learning the new task.

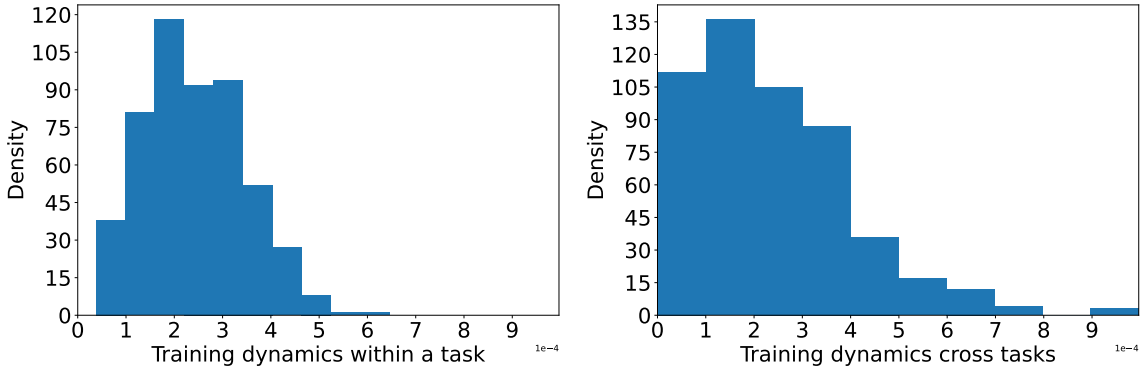
Dynamics on different scenarios The above empirical study is limited to SGD without applying any other CL techniques and only focuses on class-IL. In the following, we extend the studies to different CL methods, non-standard datasets, and domain-IL while fixing the model to be ResNet-18. Fig. 5.1 (f) extends the empirical study to a medical dataset Seq-OrganAMNIST. Compared to Seq-CIFAR-10, it differs in the number of tasks, dataset size, image size, and data type. We further replace SGD with ER using a replay buffer, whose results are reported in Fig. 5.1 (g). The ranking order of parameter groups in terms of sensitivity stays consistent under the change of dataset and replay strategy.

In domain-IL, as shown in Fig. 5.1 (h), the training dynamics of different parameters are in line with our observations in class-IL: only a small portion of parameters are task-specific. However, one difference is worth noting. Since the output classes stay the same across tasks and only the input domain changes, the last FC layer, which is the most sensitive in class-IL, becomes equally or less sensitive than the bottom convolutional layer. Hence, the plasticity and stability of parameters are impacted by how close they are to the changed data distributions.

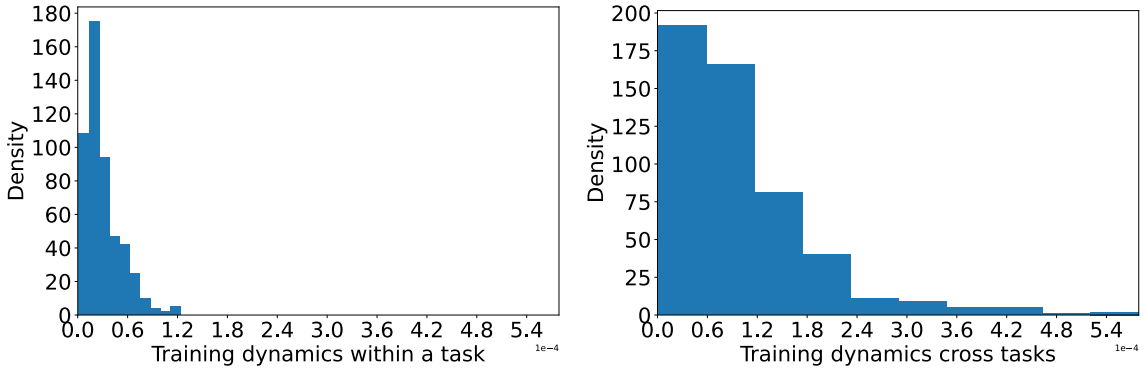
Inspiration from Empirical Studies. Above studies shed light on the improvements of CL methods. **(1)** We compare the sensitivity of different parameters in three types of deep neural networks and observe that only a small portion of them are much more sensitive than others. This implies that only finetuning these task-specific (or



(a) Bottom convolutional layer



(b) Middle convolutional layer



(c) Top convolutional layer

Figure 5.2: Comparison between the distribution of filters' training dynamics within a task and that cross tasks in different convolutional layers of VGG-11. When tasks shift, for all layers, only a small part of the filters in each layer experience more changes.

plastic) parameters may suffice to retain the previous tasks and reduce the bias. **(2)** The dynamics between consecutive epochs show that all layers experience more changes when tasks shift, which can be used to detect task boundaries during CL. Knowing task boundaries is a critical prerequisite for lots of CL methods. The proposed metric makes the CL problem much easier, makes these methods more general and can contribute to better CL methods. **(3)** According to Fig. 5.2, in convolutional layers, only a small part of task-specific filters leads to a great change of dynamics when tasks shift. The regularization or isolation of these filters can improve the performance of CL.

5.4 Forgetting Prioritized Finetuning Methods

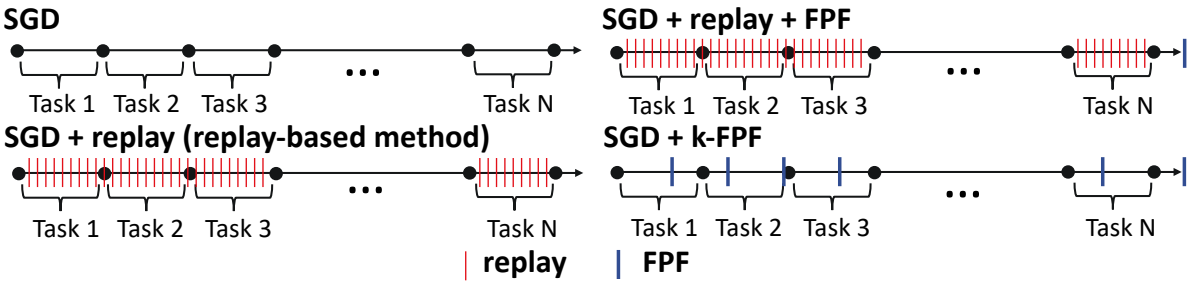


Figure 5.3: Comparison of SGD, replay-based method, FPF and k-FPF. SGD trains tasks sequentially without replay. Replay-based methods train models on buffered and current data simultaneously. FPF finetunes the most sensitive parameters for a few iterations using buffered data at the end of arbitrary CL methods. k-FPF periodically (regardless of task boundaries) applies FPF for k times during training.

The above empirical study of the training dynamics on parameters immediately motivates a simple but novel method for CL, i.e., "forgetting prioritized finetuning (FPF)", which can be applied to any existing CL method. In the more efficient k-FPF, we further remove the every-step replay and any other CL techniques but simply apply k-times of FPF in SGD training. In Fig. 5.3, we provide an illustration that compares SGD, replay-based methods and our methods. At last, we propose a metric to automatically identify sensitive parameters in each neural network. For FPF, if the existing CL method

belongs to replay-based methods, the buffer will be reserved according to this method. Otherwise, FPF will reserve the buffer by reservoir sampling. For both FPF and k-FPF, the data of each task may be trained for more than one epoch.

FPF to improve CL performance. FPF applies lightweight finetuning to the most task-specific parameters using the buffered data after the training of arbitrary CL methods. Hence, it is complementary to any existing CL methods as a correction step to remove their biases in the task-specific parameters by finetuning them on unbiased buffered data. Thereby, it can improve the performance of any existing CL methods without causing notably extra computation. The detailed procedure of IFP is described in Alg. 5.

Algorithm 5 FPF

Input : Dataset D, an existing CL method M, early epochs iteration I, number of finetuning iterations K
Output : A well-trained CL model
Initialize : Buffer $B \leftarrow \emptyset$, training iterations $i = 0$

```
for (x, y) ∈ D do
    Run a batch of CL method M;
    i ← i + 1;
    if i = I then
        Identify sensitive groups of parameters {g} by Eq. 5.1.
    end
    B ← reservoir(B, (x, y)).
end
Finetune the sensitive groups of parameters {g} for K iterations on B.
```

k-FPF to improve CL efficiency and performance. FPF is a simple technique that brings non-trivial improvement, but it is applied after the training of an existing CL method. Unfortunately, many SOTA CL methods require time-consuming replay in every step, which at least doubles the total computation. Since only a few parameters are sensitive during the task shift, can we develop a replay-free and lazy CL that replaces every-step replay with occasional FPF? We propose k-FPF that applies FPF k times during CL as shown in Fig. 5.3. Without the costly experience replay, k-FPF can still

achieve comparable performance as FPF+SOTA CL methods but only requires nearly half of their computation. We can apply k-FPF with any replay-free method, e.g., SGD, which is usually used as a lower bound for CL methods. We still maintain a small buffer by reservoir sampling, but it is only for FPF, so SGD never accesses it. We lazily apply FPF on the buffer after every τ SGD step (in total k times over $k\tau$ SGD steps) without knowing the task boundaries. The detailed procedure of IFP is described in Alg. 6.

Algorithm 6 k-FPF

Input : Dataset D, CL method SGD, early epochs iteration I, interval of FPF τ, number of finetuning iterations K
Output : A well-trained CL model
Initialize : Buffer $B \leftarrow \emptyset$, training iterations $i = 0$

for *(x, y)* $\in D$ **do**
  Run a batch of SGD on (x, y);
  $i \leftarrow i + 1$;
  if $i = I$ **then**
    Identify sensitive groups of parameters $\{g\}$ by Eq. 5.1.
  end
  if $i\%\tau = 0$ **then**
    Finetune the sensitive groups of parameters $\{g\}$ for K iterations on B.
  end
  $B \leftarrow$ reservoir$(B, (x, y))$.
end
Finetune the sensitive groups of parameters $\{g\}$ for K iterations on B.

k-FPF-CE+SGD We propose two variants of k-FPF, i.e., k-FPF-CE+SGD and k-FPF-KD+SGD. k-FPF-CE+SGD uses the cross-entropy loss to update the sensitive parameters during each FPF. In this work, k-FPF-CE refers to k-FPF-CE+SGD if not specified. The objective of FPF in k-FPF-CE is: $\min_{\Theta^\star} L(\Theta^\star) \triangleq \mathbb{E}_{(x,y)\sim B}[l_{CE}(y, f_\Theta(x))]$ where $\Theta^\star$ denotes selected groups of task-specific parameters, B refers to the buffered data and l_{CE} is the cross-entropy loss.

k-FPF-KD+SGD to further improve performance Inspired by DER [1], we further propose k-FPF-KD that introduces knowledge distillation (KD) [204] to the objective in k-FPF-CE. In this work, k-FPF-KD refers to k-FPF-KD+SGD if not specified. Same as

DER, the pre-softmax responses (i.e. logits) for buffered data at training time are stored in the buffer as well. During FPF, the current model is trained to match the buffered logits to retain the knowledge of previous models. The objective of FPF in k-FPF-KD is: $\min_{\Theta^\star} L(\Theta^\star) \triangleq \mathbb{E}_{(x,y)\sim B}[l_{CE}(y, f_\Theta(x))] + \lambda \mathbb{E}_{(x,z)\sim B}[l_{MSE}(z, h_\Theta(x))]$ where z is the logits of buffered sample x, l_{MSE} refers to the mean-squared loss, $h_\Theta(x)$ computes the pre-softmax logits and λ is a hyper-parameter balancing the two terms. Compared to the computation of k-FPF-CE, the additional computation of k-FPF-KD is negligible.

5.4.1 Selection of sensitive parameters for FPF and k-FPF

A key challenge in FPF and k-FPF is to select the task-specific parameters for finetuning. Examples of the training dynamics for different layers of various networks are shown in plots (b,d,e) of Fig. 5.1, and their ranking does not change over epochs. So we propose to select sensitive parameters for different neural networks according to their training dynamics in the early epochs. Specifically, for each neural network, its layers are partitioned into G groups as shown in Fig. 5.1, we calculate the sensitive score S_g for each group of layers in the neural network by

$$S_g = \frac{(1/|g|)\sum_{\ell \in g} C_\ell}{\sum_{g=1}^{G} (1/|g|)\sum_{\ell \in g} C_\ell} * G \tag{5.1}$$

where C_ℓ is the training dynamics mentioned in Sec. 5.3.1. We calculate the ratio of sensitivity for group g over all G groups in the network. Since each network consists of a different number of parameter groups, we multiply G to rescale the sensitivity score.

In the experiments later, under different scenarios and on various benchmarks, we evaluate the performance of FPF and k-FPF when selecting different subsets of task-specific parameters. In a nutshell, finetuning parameters of higher sensitivity achieve more improvement, which is in line with our findings in empirical studies. FPF outperforms all baselines when parameter groups whose $S_g > 1$ are regarded as sensitive parameters and account for only $1.13\%, 0.32\%$ and 0.15% of the number of all parame-

ters in MLP, VGG-11 and ResNet-18. For k-FPF, finetuning more parameters, i.e., the earlier convolutional layers, achieves the best performance. This is the price of removing replay, which halves the computational cost. We set the threshold of sensitivity score to 0.3 so that k-FPF achieves SOTA performance and only 12.40%, 1.69% and 24.91% of parameters in MLP, VGG-11 and ResNet-18 are regarded as sensitive parameters.

5.5 Experiments

In this section, to compare FPF and k-FPF with SOTA CL methods, we conduct our experiments mainly on ResNet-18. We apply FPF and k-FPF to multiple benchmark datasets and compare them with SOTA CL baselines in terms of test accuracy and efficiency. Besides, we also compare the performance of finetuning different parameters in FPF and k-FPF and show that finetuning a small portion of task-specific parameters suffices to improve CL. FPF improves SOTA CL methods by a large margin under all these scenarios, while k-FPF achieves comparable performance with FPF but is more efficient.

Implementation Details. We follow the settings in [1] to train various SOTA CL methods on different datasets, except training each task for only 5 epochs, which is more practical than 50 or 100 epochs in [1] for the streaming setting of CL. Since the epochs are reduced, we re-tune the learning rate and hyper-parameters for different scenarios by performing a grid-search on a validation set of 10% samples drawn from the original training set. For both FPF and k-FPF, we use the same optimizer, i.e., SGD with the cosine-annealing learning rate schedule, and finetune the selected parameters with a batchsize of 32 for all scenarios. The finetuning steps for FPF and k-FPF are 300 and 100, respectively. We perform a grid-search on the validation set to tune the learning rate and other hyper-parameters.

Baseline methods. We apply FPF to several SOTA memory-based CL methods:

ER, iCaRL, A-GEM [205], FDR [206], DER and DER++. Besides, we also compare our methods with GDUMB [207] and oEWC. We take JOINT as the upper bound for CL, which trains all tasks jointly, and SGD as the lower bound, which trains tasks sequentially without any countermeasure to forgetting. For FPF, k-FPF, and all memory-based methods, the performance with buffer sizes 200 and 500 are reported. All results reported in Tab. 5.1 are averaged over five trials with different random seeds.

5.5.1 Main Results

FPF considerably improves the performance of all memory-based CL methods and achieves SOTA performance over all scenarios in class-IL and domain-IL in Tab. 5.1. For methods with catastrophic forgetting, like AGEM, the accuracy of FPF increases exponentially. The surge in performance illustrates that FPF can eliminate bias by finetuning task-specific parameters to adapt to all seen tasks.

k-FPF-CE replaces the costly every-step replay with efficient occasional FPF. In Tab. 5.1, the performance of k-FPF-CE on Seq-PathMNIST, Seq-Tiny-ImageNet and Seq-PACS are better than the best CL methods, and its performance on Seq-OrganAMNIST and Seq-Cifar10 are also better than most CL methods, which implies that finetuning the task-specific parameters on a small number of buffer during SGD can help retain the previous knowledge and mitigate forgetting, **every-step replay is not necessary**. In Fig. 5.4, the number of training FLOPs and the accuracy of different methods are reported. Compared to the training FLOPs of several CL methods, the computation cost of FPF and k-FPF-CE is almost negligible. The overall training FLOPs of k-FPF-CE are still much less than SOTA CL methods while its performance is better, which shows the efficiency of k-FPF-CE.

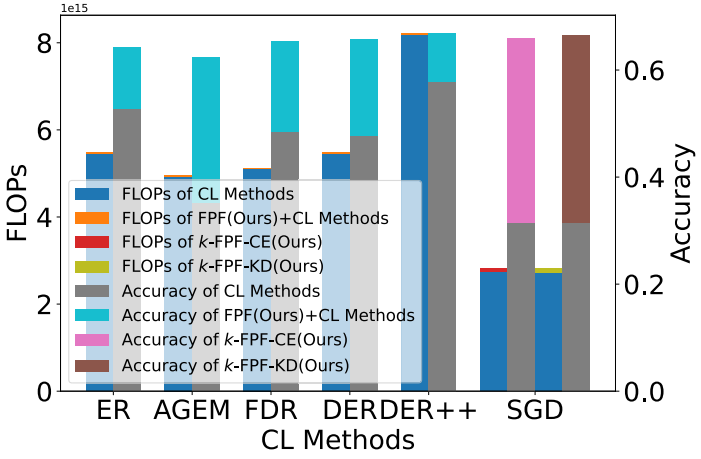
k-FPF-KD further improves the performance of k-FPF-CE to be comparable to FPF. k-FPF-CE proposes the efficiency of CL methods, but its performance is a bit

Table 5.1: Test accuracy (%) of baselines, FPF and k-FPF. "-" indicates that the algorithm is not applicable to the setting. k-FPF-KD applies an additional knowledge distillation loss to the objective of k-FPF-CE. **Bold** and **Bold gray** mark the best and second best accuracy.

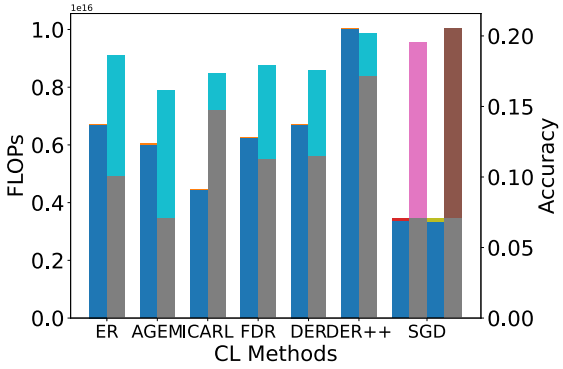
Buffer	Methods	class-IL				Domain-IL
		Seq-OrganAMNIST	Seq-PathMNIST	Seq-CIFAR-10	Seq-Tiny-ImageNet	Seq-PACS
	JOINT	91.92±0.46	82.47±2.99	81.05±1.67	41.57±0.55	70.85±8.90
	SGD	24.19±0.15	23.65±0.07	19.34±0.06	7.10±0.14	31.43±6.39
	oEWC [192]	22.71±0.67	22.36±1.18	18.48±0.71	6.58±0.12	35.96±4.59
200	GDUMB [207]	61.78±2.21	46.31±5.64	30.36±2.65	2.43±0.31	34.16±3.45
	k-FPF-CE	75.21±2.03	**72.88±3.22**	57.97±1.53	13.76±0.72	60.70 ±2.81
	k-FPF-KD	**80.32±1.16**	**74.68±4.72**	58.50±1.03	**14.74±0.94**	63.15±1.19
	ER [139]	71.69±1.71	51.66±5.86	45.71±1.44	8.15±0.25	51.53±5.10
	FPF+ER	76.92±2.26	67.34±2.68	57.68±0.76	13.08±0.65	**65.16±1.97**
	AGEM [205]	24.16±0.17	27.93±4.24	19.29±0.04	7.22±0.15	40.54±3.43
	FPF+AGEM	72.22±2.45	66.88±3.05	55.33±2.19	12.27±0.49	57.33±0.76
	iCaRL [42]	79.61±0.56	54.35±0.94	59.60±1.06	12.13±0.20	-
	FPF+iCaRL	80.28±0.58	71.20±2.19	**63.36±0.91**	**16.99±0.37**	-
	FDR [206]	68.29±3.27	44.27±3.20	41.77±4.24	8.81±0.19	45.91±3.54
	FPF+FDR	76.10±0.87	70.06±2.78	51.91±2.77	11.52±0.72	57.17±1.31
	DER [1]	73.28±1.33	54.45±5.92	47.04±3.03	9.89±0.58	46.93±4.94
	FPF+DER	79.63±1.21	67.29±3.75	56.67±2.19	12.65±0.60	61.49±1.37
	DER++ [1]	78.22±2.05	62.00±3.79	59.13±0.81	12.12±0.69	55.75±2.02
	FPF+DER++	**80.99±0.91**	68.78±2.99	**61.69±0.97**	13.72±0.40	**65.28±1.02**
500	GDUMB [207]	73.29±1.82	63.55±5.62	42.18±2.05	3.67±0.25	43.29±2.53
	k-FPF-CE	81.28±0.71	76.72±1.94	64.35±0.87	19.57±0.37	65.90±0.72
	k-FPF-KD	85.16±0.67	**79.20±3.89**	66.43±0.50	**20.56±0.32**	**66.42±2.21**
	ER [139]	80.45±0.99	57.54±3.05	57.64±4.27	10.09±0.34	52.72±4.01
	FPF+ER	84.07±1.26	69.83±2.87	65.47±2.64	18.61±0.70	64.27±1.91
	AGEM [205]	24.00±0.18	27.33±3.93	19.47±0.03	7.14±0.10	35.29±4.94
	FPF+AGEM	78.98±1.80	73.32±3.73	57.84±1.98	16.16±0.30	62.40±1.89
	iCaRL [42]	82.95±0.47	57.67±1.13	62.26±1.09	14.81±0.37	-
	FPF+iCaRL	83.88±0.69	73.56±3.00	**67.75±0.67**	16.69±0.29	-
	FDR [206]	76.62±1.81	40.08±4.13	43.52±1.74	11.33±0.33	48.50±4.67
	FPF+FDR	82.32±0.91	73.64±3.85	63.09±0.81	17.10±0.35	65.39±1.83
	DER [1]	82.52±0.52	66.71±3.40	55.98±3.35	11.54±0.70	47.63±3.85
	FPF+DER	**85.18±0.39**	74.13±3.12	67.52±0.83	17.34±0.53	65.69±1.66
	DER++ [1]	84.25±0.47	71.09±2.60	67.06±0.31	17.14±0.66	57.77±2.54
	FPF+DER++	**85.40±0.26**	**77.37±1.32**	**69.08±0.92**	**20.17±0.35**	**66.89±1.32**

worse than that of FPF. One of the most differences between k-FPF and FPF is the experience replay during the training of CL. Inspired by DER, we propose k-FPF-KD, which uses knowledge distillation to match the outputs of previous models on buffered data, hence retaining the knowledge of previous tasks. The results of k-FPF-KD in Tab. 5.1 show that it is comparable to FPF in most scenarios. Fig. 5.4 shows that the

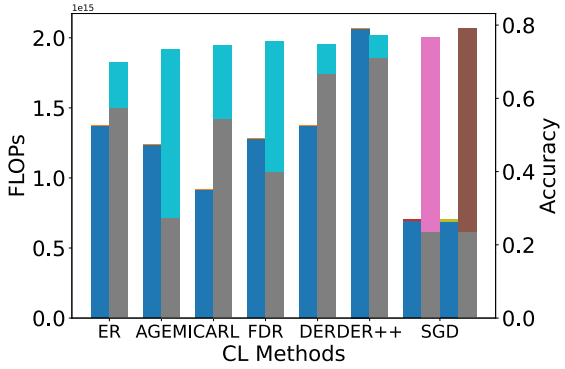
FLOPs of k-FPF-KD are similar to k-FPF-CE but much less than other CL methods and FPF, and in some cases, it outperforms FPF. **k-FPF-KD shows SOTA performance in both efficiency and accuracy**.



(a) Seq-PACS



(b) Seq-Tiny-ImageNet



(c) Seq-PathMNIST

Figure 5.4: Comparison of FLOPs and accuracy between FPF, k-FPF and SOTA methods. **FPF improves all CL methods by a large margin without notably extra computation. k-FPF consumes much less computation but achieves comparable performance as FPF**.

5.5.2 Finetuning Different Parameters in FPF and k-FPF

FPF and k-FPF get the best performance when only a small portion of task-specific parameters are finetuned. In Fig. 5.5, the accuracy, training FLOPs and the

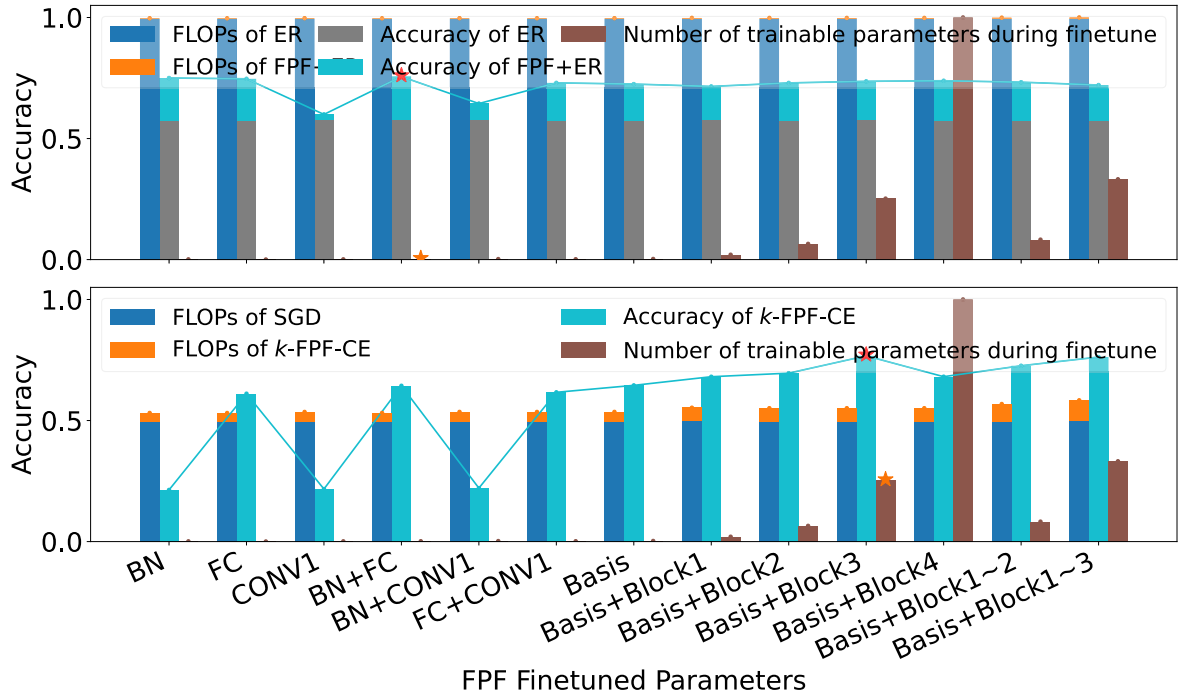


Figure 5.5: Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show **both FPF and k-FPF only require to finetune a small portion of task-specific parameters. k-FPF halves FPF's FLOPs**.

number of trainable parameters during finetune of applying FPF or k-FPF to different task-specific parameters in ResNet-18 on Seq-PathMNIST are compared. Overall different scenarios, k-FPF only needs about half FLOPs of FPF with better performance (indicated by Red Stars). When finetuning on different task-specific parameters, FPF performs the best when BN+FC layers are finetuned, which is only 0.127% of all parameters (indicated by Orange Stars). This is consistent with our observations in empirical studies where BN and FC layers are the most sensitive parameters to distribution shift. And the results show that only finetuning a small portion of task-specific parameters can mitigate catastrophic forgetting and generalize the model.

The phenomenon for k-FPF is a little different. (1) In the bottom plot of Fig. 5.5, when

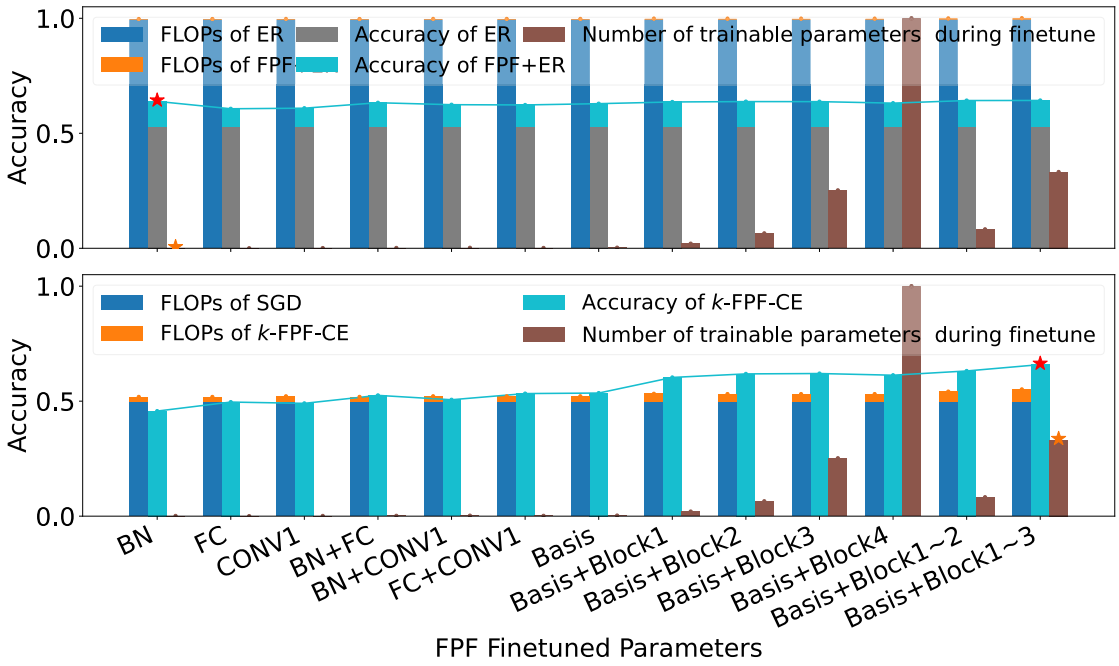


Figure 5.6: Comparison of FLOPs, number of finetuned parameters, and accuracy for FPF(Top) and k-FPF(Bottom) finetuning different combinations of parameters. All FLOPs are normalized together to (0,1], as well as the number of finetuning parameters. "Basis" in the x-label refers to "BN+FC+CONV1". Red stars highlight the best accuracy and show both FPF and k-FPF only require finetuning a small portion of task-specific parameters. k-FPF halves FPF's FLOPs. **Different from the results of k-FPF in class-IL, in Seq-PACS, since the output classes for different tasks are always the same, the last FC layer will not have a large bias on particular classes. Only finetuning BN or CONV1 layers for k-FPF can get comparable performance with ER.** Similar to class-IL, since experience replay is not allowed during the training of CL method SGD, a few more parameters are required to be finetuned by k-FPF to get comparable performance with FPF (about 24.92% of all parameters).

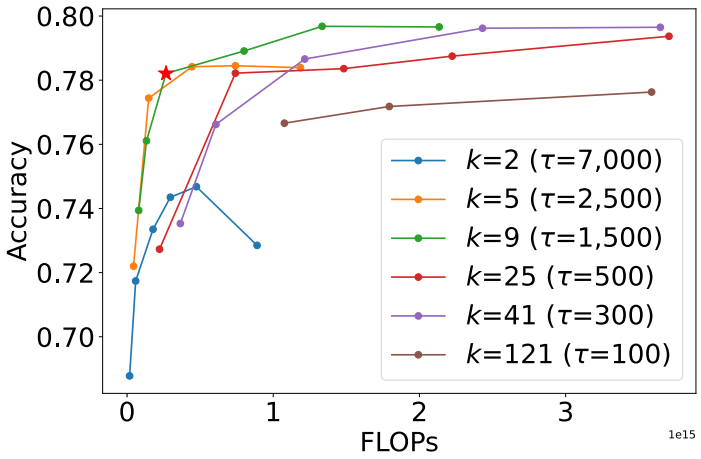
FC layer is not selected for finetuning in k-FPF, the performance is much worse. This is because, in class-IL, the output classes change across tasks, so the FC layer is trained to only output the classes for the current task [208]. In contrast, as shown in Fig. 5.6, when applying k-FPF to domain-IL on Seq-PACS, where the output classes keep the same for different tasks, Fig. 5.6 shows that finetuning FC layer performs similarly as finetuning other parameters. Hence, the last FC layer is more sensitive in class-IL than in Domain-IL. This is also shown in Fig. 5.1 (d,h). (2) As the red star indicates, k-FPF

needs to finetune a little more parameters (Block3 of convolutional layers, 18.91% of all parameters) to achieve a comparable accuracy with FPF. Without experience replay during SGD, the model has a larger bias on the current task, and thus more task-specific parameters are needed to be finetuned. This also indicates that such bias of task-specific parameters is the main reason for catastrophic forgetting. When Block4 (75.22% of all parameters) is finetuned, since it is the most stable group of parameters in our empirical study, the performance of k-FPF degrades.

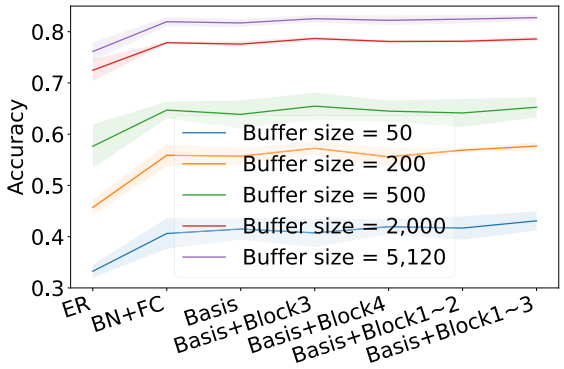
5.5.3 Analysis of FPF and k-FPF in Different Scenarios

Different training FLOPs for k-FPF In Fig. 5.7(a), we study the trade-off between the training FLOPs and the accuracy of k-FPF on Seq-PathMNIST by changing k and the number of finetuning steps. τ in the legend refers to the interval of two consecutive FPF. Fixing k, k-FPF saturates quickly as the finetuning steps increase. This implies that k-FPF is efficient on FLOPs to achieve the best performance. For experiments with small k, e.g., k=2, though the computation required is very low, performance cannot be further improved. This implies that FPF must be applied more frequently on buffered samples to mitigate forgetting. When k is large, e.g., k=41 or 121, the accuracy slightly improves with the price of much more required computation.

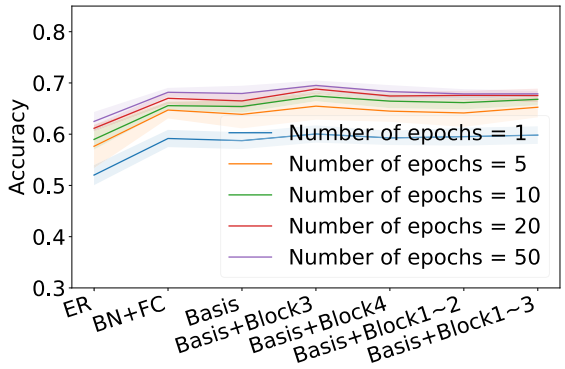
As the red star in the plot indicates, the optimal trade-off between computation and accuracy is achieved when $k = 9(\tau = 1{,}500)$. We observe that, when $k = 9$, FPF is applied 1-2 times during the training of each task, which is crucial for reaching a balanced trade-off. With $k = 9$, the interval between two consecutive FPF isn't excessively long, allowing for the prompt replay of previous task data along with all buffered data right after they are learned, preventing potential forgetting due to the acquisition of knowledge from the next task by the DNNs. Additionally, the interval between two consecutive FPF isn't too small, thereby minimizing the computational requirements. We suggest that applying



(a) FLOPs-Accuracy in k-FPF



(b) Effect of different buffer sizes for FPF



(c) Effect of different training epochs for FPF

Figure 5.7: **Top**: Trade-off between FLOPs and accuracy for k-FPF with different k and τ (the SGD steps between two consecutive FPF)(a). By increasing the finetuning steps per FPF, the accuracy quickly saturates. The best trade-off is highlighted at the top-left corner when $k = 9(\tau = 1,500)$. **Bottom**: Comparison between ER and FPF+ER finetuning different parameters with different buffer sizes(b) and numbers of epochs per task(c). In all scenarios, FPF can significantly improve ER by only finetuning BN+FC.

FPF at least once during the training of each task effectively mitigates forgetting and leads to improved performance.

The transferability of different modules for sequential tasks in DNNs The buffer size and the training epochs per task are usually crucial in replay-based CL methods. In Fig. 5.7(b) and (c), when the buffer size or the number of epochs increases, the performance of ER improves as well. However, increasing the buffer size brings more benefits. When the buffer size or epochs grow too large, the performance of ER

seems to saturate and increase slowly. For all scenarios, finetuning BN+FC layers is highly effective in alleviating the current task's bias and promoting performance, which is consistent with our observations from the empirical studies.

The fairness of k-FPF across sequential tasks In Tab. 5.2 and Tab. 5.3, the average test accuracy of previous tasks at the end of each task during the training of CL on Seq-PathMNIST and Seq-PACS is reported. The results show that k-FPF can always achieve the best performance among various CL methods during training. Whenever the training stops, k-FPF can always achieve a model performing well on previous tasks. The results clearly demonstrate that k-FPF can achieve balance across tasks and minimize bias during the training process, resulting in a fair CL model.

Table 5.2: The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PathMNIST. **Bold** and **Bold gray** mark the best and second best accuracy.

Methods	Task 1	Task 2	Task 3	Task 4
k-FPF-CE	99.95±0.04	**95.41±1.98**	**81.92±2.26**	**76.72±1.94**
ER	98.62±1.59	83.06±3.12	74.60±3.18	57.54±3.05
AGEM	99.71±0.19	46.58±3.13	36.12±3.17	27.33±3.93
iCaRL	**99.98±0.02**	86.86±5.47	66.62±5.64	57.67±1.13
FDR	99.97±0.06	48.06±0.82	55.75±6.55	40.08±4.13
DER	**99.98±0.02**	91.92±3.42	76.50±5.77	66.71±3.40
DER++	99.95±0.06	**94.06±6.14**	**80.35±3.32**	**71.09±2.60**

Table 5.3: The average accuracy of previous tasks at the end of each task during the training of CL on Seq-PACS. **Bold** and **Bold gray** mark the best and second best accuracy.

Methods	Task 1	Task 2	Task 3	Task 4
k-FPF-CE	**70.94±2.02**	**73.75±2.68**	**62.37±0.49**	**65.90±0.72**
ER	56.64±9.04	54.34±9.44	46.79±8.48	52.72±4.01
AGEM	47.34±7.35	38.02±5.82	32.70±7.13	35.29±4.94
FDR	**58.59±4.36**	54.00±4.01	46.38±4.80	48.50±4.67
DER	48.49±9.40	45.28±8.88	34.48±7.81	47.63±3.85
DER++	55.33±7.45	**64.43±6.50**	**50.19±7.30**	**57.77±2.54**

5.6 Ablation Studies

Comparison between FPF and the method finetuning all parameters In Tab.5.4, we compare FPF with FPF-ALL (which finetunes all parameters) when applied to different CL methods for two types of CL, i.e., class-IL and domain-IL. The results show that FPF consistently achieves comparable or slightly higher accuracy than FPF-ALL by spending significantly fewer FLOPs. This demonstrates the advantage of FPF on efficiency.

Table 5.4: Comparison of accuracy and FLOPs between FPF and FPF-ALL(finetuning all parameters). **Bold** and **Bold gray** mark the best and second best accuracy.

Methods	Seq-PathMNIST		Seq-PACS	
	Accuracy	FLOPs(B)	Accuracy	FLOPs(B)
k-**FPF-CE**	76.72±1.94	21.35	65.90±0.72	148.25
k-**FPF-ALL-CE**	75.74±2.91	43.95	64.48±2.23	174.60
FPF+ER	69.83±2.87	4.68	64.27±1.91	24.39
FPF-ALL+ER	70.64±4.00	8.79	63.81±2.33	34.92
FPF+AGEM	73.32±3.73	7.07	62.40±1.89	18.47
FPF-ALL+AGEM	74.80±3.12	8.79	62.65±1.65	34.92
FPF+iCaRL	73.56±3.00	4.27	-	-
FPF-ALL+iCaRL	72.77±4.12	8.79	-	-
FPF+FDR	73.64±3.85	2.94	65.39±1.83	11.70
FPF-ALL+FDR	74.24±1.48	8.79	64.88±2.28	34.92
FPF+DER	74.13±3.12	2.96	65.69±1.66	18.47
FPF-ALL+DER	74.54±3.19	8.79	**66.22±0.87**	34.92
FPF+DER++	**77.37±1.32**	4.68	**66.89±1.32**	24.39
FPF-ALL+DER++	**77.16±1.45**	8.79	65.19±1.33	34.92

Results of other neural networks In Tab. 5.5, the results of various CL benchmarks and FPF on MLP and VGG-11 are reported. Similar to the results in Tab.5.1, by finetuning the most sensitive parameters in MLP and VGG-11, FPF can further improve the performance of all SOTA CL methods and achieve the best performance. k-FPF-CE also achieves comparable performance as FPF + SOTA methods. Our methods can be generalized to various neural networks.

Comparison between k-FPF, DER and DER++ with a large number of epochs

Table 5.5: Classification results for CL benchmarks and FPF on MLP and VGG-11. **Bold** and **Bold gray** mark the best and second best accuracy.

Buffer	Methods	Class-IL	
		Seq-Mnist(MLP)	Seq-CIFAR10(VGG-11)
	JOINT	95.58±0.33	69.50±0.73
	SGD	19.64±0.07	18.71±0.33
	oEWC	20.69±1.34	18.46±0.23
500	GDUMB	90.60±0.37	41.65±0.78
	k-FPF-CE	90.63±0.57	55.45±1.16
	ER	86.73±1.03	46.27±1.18
	FPF+ER	91.15±0.16	53.48±1.08
	AGEM	51.03±4.94	19.40±1.09
	FPF+AGEM	89.26±0.52	29.84±1.37
	iCaRL	58.12±1.94	45.63±1.94
	FTF+iCaRL	80.83±0.49	48.03±0.65
	FDR	83.79±4.15	45.56±2.23
	FPF+FDR	89.67±0.37	**55.59±1.56**
	DER	91.17±0.94	51.12±2.47
	FPF+DER	**91.25±0.89**	**57,46±1.15**
	DER++	91.18±0.74	47.60±3.23
	FTF+DER++	**91.22±0.67**	54.69±0.73

for each task. We compare the accuracy and FLOPs of our methods with the original results in [1] when allowing a large number of epochs on the same data for each task. The results are shown in Tab. 5.6. R-MNIST is a domain-IL dataset applied in [1]. In both class-IL and domain-IL, k-FPF-CE is comparable to DER++ and k-FPF-KD is better than DER++ on the accuracy but spends much less FLOPs. These results demonstrate that our methods can outperform SOTA methods in various scenarios.

Table 5.6: Comparison of accuracy and FLOPs of k-FPF with the original results in [1] which has a large number of training epochs for each task.

Methods	Seq-CIFAR-10 Accuracy	Seq-CIFAR-10 FLOPs (B)	R-MNIST Accuracy	R-MNIST FLOPs (B)
k-FPF-CE	71.93±0.58	9208.85	91.15±0.29	0.64
k-FPF-KD	**74.32±0.32**	9208.85	**93.61±0.45**	0.64
DER	70.51±1.67	16726.26	92.24±1.12	1.29
DER++	72.70±1.36	25089.39	92.77±1.05	1.93

5.7 Conclusion

We study a fundamental problem in CL, i.e., which parts of a neural network are task-specific and more prone to strong bias and catastrophic forgetting. Extensive empirical studies in diverse settings consistently show that only a small portion of parameters is task-specific and sensitive. This discovery leads to an effective "forgetting prioritized finetuning (FPF)" that only finetunes a subset of these parameters on the buffered data before model deployment. FPF is complementary to existing CL methods and can consistently improve their performance. We further replace the costly every-step replay with k-times of occasional FPF during CL to improve efficiency. Such k-FPF achieves comparable performance as FPF+SOTA CL while consuming nearly half of its computation. Our proposed methods address the issue of bias during training, ultimately promoting fairness across sequential tasks.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The main objective of this thesis is to investigate knowledge sharing between tasks in the deep neural network to improve its efficacy and trustworthiness. The study is carried out in three stages; each focused on a specific aspect of the problem.

The first stage focuses on gaining a deeper understanding of the pre-trained model's behavior for a given task. To provide a human-readable explanation, an automatic pruning method is applied to eliminate redundant filters, layers, and blocks in the pre-trained model for the task. Subsequently, we visualize the retained layers and assemble them layer by layer to illustrate the inference chains for the given task. The improved interpretability allows us to identify critical parameters and establish a clearer understanding of the relationships between tasks, making it easy to share knowledge across tasks.

In the second stage, I address the efficiency challenge of generating a pruned model for a new task or device by sharing knowledge. Specifically, we conduct an empirical

study to examine the relationship between pruned models of different tasks and discover that tasks with greater similarity share more parameters in the pre-trained model. We harness the knowledge from similar tasks for each new task to rapidly construct a reliable, high-performing pruned model tailored to the new task. Our proposed method is validated through experiments using real-world datasets, showcasing its reliability and superiority over existing approaches.

In the final stage, I concentrate on the issue of knowledge sharing when tackling sequential tasks with the aim of reducing bias across tasks. We perform an empirical study to analyze the training dynamics of various components within deep neural networks, revealing that only a small fraction of parameters in the model are truly task-specific and sensitively change between tasks. Building on this observation, we propose finetuning the selected sensitive parameters to share knowledge between tasks and removing the every-step replay to enhance the efficiency of continual learning. Extensive experiments on different datasets demonstrate that our proposed methods can achieve fairness across tasks and significantly outperform existing methods in terms of efficiency and accuracy.

In summary, this thesis offers a deeper understanding of knowledge sharing across tasks and integrates knowledge sharing into various machine learning scenarios to improve the trustworthiness and efficiency of DNNs. The proposed methods are validated through comprehensive quantitative analyses and experiments, showcasing their effectiveness and offering valuable insights for future research in this field.

6.2 Future Work

The research presented in this thesis has paved the way for numerous future work opportunities. With the advancement of large language models (LLMs), our approach can address challenges not only in traditional deep learning but also in the realm of

large-scale models. Here are some potential directions for subsequent research:

In the first problem, we explain the inference process of conventional pre-trained models for a given task. Despite the remarkable reasoning capabilities demonstrated by LLMs in NLP, the underlying mechanisms remain unclear [209]. It is essential to grasp the connections between different tasks in LLMs and understand if a shared contextual framework across tasks leads to emergent capabilities in these models [210]. Our method enables us to pinpoint crucial parameters for each task within LLMs, revealing the role of each module and layer. This facilitates the analysis of inter-task relationships and contributes to a deeper understanding of the internal workings of LLMs.

Compared to conventional models, our proposed meta-vote pruning (MVP) method can significantly enhance pruning efficiency and accuracy on LLMs. Due to the large size of LLMs, starting pruning from scratch is time-consuming [211]. Additionally, diverse tasks and data volumes restrict the potential for structural pruning ratio in LLMs to only 20% [212, 213], leading to suboptimal outcomes. Our proposed method MVP can leverage the multi-task capabilities of LLMs to facilitate knowledge sharing between tasks. This enables the model to preserve more information with fewer parameters, thereby enhancing the overall efficiency and accuracy of LLMs pruning.

To address the problem of catastrophic forgetting in deep models, we can further investigate the stability and plasticity of deep models [214] through parameter analysis. Task-specific parameters promote model plasticity, enabling efficient learning for specific tasks. Conversely, task-agnostic parameters ensure model stability, allowing retention of learned knowledge across tasks. By strategically coordinating these two sets of parameters based on specific requirements, we can create a well-balanced model for addressing the current problem. Furthermore, catastrophic forgetting is a significant concern in LLMs when fine-tuning on new datasets [215, 216]. Extending our approach to LLMs can help identify task-specific and task-agnostic parameters. By combining our

method with regulation-based techniques, we can facilitate knowledge sharing between sequential tasks, achieving a balance between stability and plasticity in LLMs.

BIBLIOGRAPHY

[1] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

[2] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[3] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[4] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[5] Sumit Chopra, Suhrid Balakrishnan, and Raghuraman Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML workshop on challenges in representation learning*, volume 2. Citeseer, 2013.

[6] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[7] 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*, pages 2661–2671, 2019.

[8] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[9] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[10] Md Adnan Arefeen, Sumaiya Tabassum Nimi, Md Yusuf Sarwar Uddin, and Yugyung Lee. Transjury: Towards explainable transfer learning through selection of layers from deep neural networks. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 978–984, 2021.

[11] Dowan Kim, Woohyun Lim, Minye Hong, and Hyeoncheol Kim. The structure of deep neural network for interpretable transfer learning. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–4, 2019.

[12] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[14] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[15] Caglar Uyulan, Turker Tekin Erguzel, Omer Turk, Shams Farhad, Baris Metin, and Nevzat Tarhan. A class activation map-based interpretable transfer learning model for automated detection of adhd from fmri data. *Clinical EEG and Neuroscience*, 54(2):151–159, 2023.

[16] Md Nazmul Islam, Mehedi Hasan, Md Kabir Hossain, Md Golam Rabiul Alam, Md Zia Uddin, and Ahmet Soylu. Vision transformer and explainable transfer learning models for auto detection of kidney cyst, stone and tumor from ct-radiography. *Scientific Reports*, 12(1):11440, 2022.

[17] T Anjali and V Masilamani. An explainable transfer learning based approach for detecting face mask. In *Computer Vision and Image Processing: 7th International Conference, CVIP 2022, Nagpur, India, November 4–6, 2022, Revised Selected Papers, Part I*, pages 72–86. Springer, 2023.

[18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016.

[19] Md Manjurul Ahsan, Tareque Abu Abdullah, Md Shahin Ali, Fatematuj Jahora, Md Khairul Islam, Amin G Alhashim, and Kishor Datta Gupta. Transfer learning and local interpretable model agnostic based visual approach in monkeypox disease detection and classification: A deep learning insights. *arXiv preprint arXiv:2211.05633*, 2022.

[20] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[21] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[22] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[23] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. *Advances in neural information processing systems*, 29, 2016.

[24] Hao Li, Asim Kadav, Igor Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2017.

[25] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[26] Xinghao Chen, Yiman Zhang, and Yunhe Wang. Mtp: multi-task pruning for efficient semantic segmentation networks. In *2022 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2022.

[27] Hanjing Cheng, Zidong Wang, Lifeng Ma, Xiaohui Liu, and Zhihui Wei. Multi-task pruning via filter index sharing: A many-objective optimization approach. *Cognitive Computation*, 13:1070–1084, 2021.

[28] Song Han, J. Pool, John Tran, and W. Dally. Learning both weights and connections for efficient neural network. *ArXiv*, abs/1506.02626, 2015.

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

[30] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

[31] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*, 2020.

[32] Shaohui Lin, R. Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and B. Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, 2018.

[33] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020.

[34] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

[35] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[36] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *International Conference on Learning Representations*, 2022.

[37] Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020.

[38] Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406*, 2020.

[39] Bingyan Liu, Yifeng Cai, Yao Guo, and Xiangqun Chen. Transtailor: Pruning the pre-trained model for improved transfer learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8627–8634, 2021.

[40] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[41] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[42] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[43] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[44] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

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

[46] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[47] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[48] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[49] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[50] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

[51] Fuchao Yu, Xianchao Xiu, and Yunhui Li. A survey on deep transfer learning and beyond. *Mathematics*, 10(19):3619, 2022.

[52] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[53] Karsten M Borgwardt, Arthur Gretton, Malte J Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57, 2006.

[54] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010.

[55] Muhammad Ghifary, W Bastiaan Kleijn, and Mengjie Zhang. Domain adaptive neural networks for object recognition. In *PRICAI 2014: Trends in Artificial Intelligence: 13th Pacific Rim International Conference on Artificial Intelligence, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings 13*, pages 898–904. Springer, 2014.

[56] Mingsheng Long, Jianmin Wang, Guiguang Ding, Sinno Jialin Pan, and S Yu Philip. Adaptation regularization: A general framework for transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1076–1089, 2013.

[57] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[58] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015.

[59] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K Sriperumbudur. Optimal kernel choice for large-scale two-sample tests. *Advances in neural information processing systems*, 25, 2012.

[60] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 443–450. Springer, 2016.

[61] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[62] Chaohui Yu, Jindong Wang, Yiqiang Chen, and Meiyu Huang. Transfer learning with dynamic adversarial adaptation network. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 778–786. IEEE, 2019.

[63] Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. Multi-adversarial domain adaptation. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

[64] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 435–442. Springer, 2016.

[65] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Beyond sharing weights for deep domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 41(4):801–814, 2018.

[66] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Residual parameter transfer for deep domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4339–4348, 2018.

[67] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4805–4814, 2019.

[68] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12299–12310, 2021.

[69] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[70] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[71] Tongkun Xu, Weihua Chen, Pichao Wang, Fan Wang, Hao Li, and Rong Jin. Cdtrans: Cross-domain transformer for unsupervised domain adaptation. *arXiv preprint arXiv:2109.06165*, 2021.

[72] Jinyu Yang, Jingjing Liu, Ning Xu, and Junzhou Huang. Tvt: Transferable vision transformer for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 520–530, 2023.

[73] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 94–108. Springer, 2014.

[74] Wu Liu, Tao Mei, Yongdong Zhang, Cherry Che, and Jiebo Luo. Multi-task deep visual-semantic embedding for video thumbnail selection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3707–3715, 2015.

[75] Sijin Li, Zhi-Qiang Liu, and Antoni B Chan. Heterogeneous multi-task learning for human pose estimation with deep convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 482–489, 2014.

[76] Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning point goal navigation. In *Conference on Robot Learning*, pages 498–516. PMLR, 2021.

[77] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019.

[78] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[79] Yusuke Shinohara. Adversarial multi-task learning of deep neural networks for robust speech recognition. In *Interspeech*, pages 2369–2372. San Francisco, CA, USA, 2016.

[80] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. *arXiv preprint arXiv:1704.05742*, 2017.

[81] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[82] Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The tc algorithm. In *ICML*, volume 96, pages 489–497. Citeseer, 1996.

[83] Koby Crammer and Yishay Mansour. Learning multiple tasks using shared hypotheses. *Advances in Neural Information Processing Systems*, 25, 2012.

[84] Wenzhao Lian, Ricardo Henao, Vinayak Rao, Joseph Lucas, and Lawrence Carin. A multitask point process predictive model. In *International conference on machine learning*, pages 2030–2038. PMLR, 2015.

[85] Qiang Zhou and Qi Zhao. Flexible clustered multi-task learning by learning representative tasks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):266–278, 2015.

[86] Theodoros Evgeniou, Charles A Micchelli, Massimiliano Pontil, and John Shawe-Taylor. Learning multiple tasks with kernel methods. *Journal of machine learning research*, 6(4), 2005.

[87] Tsuyoshi Kato, Hisashi Kashima, Masashi Sugiyama, and Kiyoshi Asai. Multi-task learning via conic programming. *Advances in Neural Information Processing Systems*, 20, 2007.

[88] Sergey Feldman, Maya R Gupta, and Bela A Frigyik. Revisiting stein's paradox: multi-task averaging. *J. Mach. Learn. Res.*, 15(1):3441–3482, 2014.

[89] Yu Zhang and Dit-Yan Yeung. A regularization approach to learning task relationships in multitask learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3):1–31, 2014.

[90] Carlo Ciliberto, Youssef Mroueh, Tomaso Poggio, and Lorenzo Rosasco. Convex learning of multiple tasks and their structure. In *International Conference on Machine Learning*, pages 1548–1557. PMLR, 2015.

[91] Yu Zhang and Qiang Yang. Learning sparse task relations in multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[92] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S Yu. Learning multiple tasks with multilinear relationship networks. *Advances in neural information processing systems*, 30, 2017.

[93] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1930–1939, 2018.

[94] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019.

[95] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[96] Yue Tan, Yixin Liu, Guodong Long, Jing Jiang, Qinghua Lu, and Chengqi Zhang. Federated learning on non-iid graphs via structural knowledge sharing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 9953–9961, 2023.

[97] Guodong Long, Yue Tan, Jing Jiang, and Chengqi Zhang. Federated learning for open banking. In *Federated Learning: Privacy and Incentive*, pages 240–254. Springer, 2020.

[98] Guodong Long, Tao Shen, Yue Tan, Leah Gerrard, Allison Clarke, and Jing Jiang. Federated learning for privacy-preserving open innovation future on digital health. In *Humanity Driven AI: Productivity, Well-being, Sustainability and Partnership*, pages 113–133. Springer, 2021.

[99] Shaoxiong Ji, Guodong Long, Shirui Pan, Tianqing Zhu, Jing Jiang, and Sen Wang. Detecting suicidal ideation with data protection in online communities. In *International conference on database systems for advanced applications*, pages 225–229. Springer, 2019.

[100] Shengchao Chen, Guodong Long, Tao Shen, Tianyi Zhou, and Jing Jiang. Spatial-temporal prompt learning for federated weather forecasting. *arXiv preprint arXiv:2305.14244*, 2023.

[101] Shengchao Chen, Guodong Long, Tao Shen, and Jing Jiang. Prompt federated learning for weather forecasting: Toward foundation models on meteorological data. *arXiv preprint arXiv:2301.09152*, 2023.

[102] Jing Jiang, Shaoxiong Ji, and Guodong Long. Decentralized knowledge acquisition for mobile internet applications. *World Wide Web*, 23(5):2653–2669, 2020.

[103] Peng Yan and Guodong Long. Personalization disentanglement for federated learning. *arXiv preprint arXiv:2306.03570*, 2023.

[104] Chunxu Zhang, Guodong Long, Tianyi Zhou, Peng Yan, Zijian Zhang, Chengqi Zhang, and Bo Yang. Dual personalization on federated recommendation. *arXiv preprint arXiv:2301.08143*, 2023.

[105] Fengwen Chen, Guodong Long, Zonghan Wu, Tianyi Zhou, and Jing Jiang. Personalized federated learning with graph. *arXiv preprint arXiv:2203.00829*, 2022.

[106] Guodong Long, Ming Xie, Tao Shen, Tianyi Zhou, Xianzhi Wang, and Jing Jiang. Multi-center federated learning: clients clustering for better personalization. *World Wide Web*, 26(1):481–500, 2023.

[107] Jie Ma, Guodong Long, Tianyi Zhou, Jing Jiang, and Chengqi Zhang. On the convergence of clustered federated learning. *arXiv preprint arXiv:2202.06187*, 2022.

[108] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. Fedproto: Federated prototype learning across heterogeneous clients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8432–8440, 2022.

[109] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[110] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3319–3328. JMLR. org, 2017.

[111] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[112] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[113] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[114] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. A unified view of gradient-based attribution methods for deep neural networks. In *NIPS 2017-Workshop on Interpreting, Explaining and Visualizing Deep Learning*. ETH Zurich, 2017.

[115] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *arXiv preprint arXiv:1702.04595*, 2017.

[116] Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6261–6270, 2019.

[117] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[118] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[119] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9194–9203, 2018.

[120] Christos Louizos, M. Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. *ArXiv*, abs/1712.01312, 2018.

[121] Zhuang Liu, J. Li, Zhiqiang Shen, Gao Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763, 2017.

[122] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, X. Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. In *NeurIPS*, 2020.

[123] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[124] Yue Bai, Huan Wang, Zhiqiang Tao, Kunpeng Li, and Yun Fu. Dual lottery ticket hypothesis. *arXiv preprint arXiv:2203.04248*, 2022.

[125] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[126] Mao Ye, L. Wu, and Qiang Liu. Greedy optimization provably wins the lottery: Logarithmic number of winning tickets is enough. *ArXiv*, abs/2010.15969, 2020.

[127] Pedro H. P. Savarese, Hugo Silva, and M. Maire. Winning the lottery with continuous sparsification. *ArXiv*, abs/1912.04427, 2020.

[128] Eran Malach, Gilad Yehudai, S. Shalev-Shwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *ICML*, 2020.

[129] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

[130] Fang Yu, Kun Huang, Meng Wang, Yuan Cheng, Wei Chu, and Li Cui. Width & depth pruning for vision transformers. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 2022, 2022.

[131] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR, 2020.

[132] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12165–12174, 2022.

[133] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable vision transformers with hierarchical pooling. In *Proceedings of the IEEE/cvf international conference on computer vision*, pages 377–386, 2021.

[134] Z. Liu, H. Mu, X. Zhang, Zichao Guo, X. Yang, K. Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3295–3304, 2019.

[135] Yawei Li, Shuhang Gu, K. Zhang, L. Gool, and R. Timofte. Dhp: Differentiable meta pruning via hypernetworks. *ArXiv*, abs/2003.13683, 2020.

[136] Hongduan Tian, Bo Liu, X. Yuan, and Qingshan Liu. Meta-learning with network pruning. *ArXiv*, abs/2007.03219, 2020.

[137] Yang He, Ping Liu, Linchao Zhu, and Y. Yang. Meta filter pruning to accelerate deep convolutional neural networks. *ArXiv*, abs/1904.03961, 2019.

[138] Tianxiang Sun, Yunfan Shao, Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu, and X. Huang. Learning sparse sharing architectures for multiple tasks. *ArXiv*, abs/1911.05034, 2020.

[139] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[140] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[141] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*, 2019.

[142] Chen He, Ruiping Wang, Shiguang Shan, and Xilin Chen. Exemplar-supported generative reproduction for class incremental learning. In *BMVC*, page 98, 2018.

[143] Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao Tao, Dongyan Zhao, Jinwen Ma, and Rui Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. In *International conference on learning representations*, 2019.

[144] Ye Xiang, Ying Fu, Pan Ji, and Hua Huang. Incremental learning using conditional adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6619–6628, 2019.

[145] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[146] Bo Zhao and Hakan Bilen. Dataset condensation with gradient matching. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[147] Zhiwei Deng and Olga Russakovsky. Remember the past: Distilling datasets into addressable memories for neural networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[148] Jang-Hyun Kim, Jinuk Kim, Seong Joon Oh, Sangdoo Yun, Hwanjun Song, Joonhyun Jeong, Jung-Woo Ha, and Hyun Oh Song. Dataset condensation via efficient synthetic-data parameterization. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 11102–11118, 2022.

[149] Lucien Le Cam. *Asymptotic methods in statistical decision theory*. Springer Science & Business Media, 2012.

[150] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[151] Quang Pham, Chenghao Liu, and Steven Hoi. Dualnet: Continual learning, fast and slow. *Advances in Neural Information Processing Systems*, 34:16131–16144, 2021.

[152] Elahe Arani, Fahad Sarfraz, and Bahram Zonooz. Learning fast, learning slow: A general continual learning method based on complementary learning system. In *International Conference on Learning Representations*, 2022.

[153] Quang Pham, Chenghao Liu, and Steven HOI. Continual normalization: Rethinking batch normalization for online continual learning. In *International Conference on Learning Representations*, 2022.

[154] Timothée Lesort, Thomas George, and Irina Rish. Continual learning in deep networks: an analysis of the last layer. *arXiv preprint arXiv:2106.01834*, 2021.

[155] Tongtong Wu, Massimo Caccia, Zhuang Li, Yuan-Fang Li, Guilin Qi, and Gholamreza Haffari. Pretrained language model in continual learning: A comparative study. In *International Conference on Learning Representations*, 2022.

[156] Vinay V Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

[157] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

[158] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[159] Haiyan Zhao, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Extracting local reasoning chains of deep neural networks. *Transactions on Machine Learning Research*, 2022.

[160] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[161] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[162] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[163] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg

Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[164] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021.

[165] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron C. Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *International Conference on Learning Representations*, 2017.

[166] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. *Advances in neural information processing systems*, 29, 2016.

[167] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[168] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, pages 7775–7784, 2018.

[169] Ting-Wu Chin, Ruizhou Ding, C. Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1515–1525, 2020.

[170] Kaiwen Yang, Yanchao Sun, Jiahao Su, Fengxiang He, Xinmei Tian, Furong Huang, Tianyi Zhou, and Dacheng Tao. Adversarial auto-augment with label preservation: A representation learning principle guided approach. *arXiv preprint arXiv:2211.00824*, 2022.

[171] Kaiwen Yang, Tianyi Zhou, Xinmei Tian, and Dacheng Tao. Identity-disentangled adversarial augmentation for self-supervised learning. In *International Conference on Machine Learning*, pages 25364–25381. PMLR, 2022.

[172] Yue Tan, Guodong Long, Jie Ma, Lu Liu, Tianyi Zhou, and Jing Jiang. Federated learning from pre-trained models: A contrastive learning approach. *arXiv preprint arXiv:2209.10083*, 2022.

[173] Haiyan Zhao, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. *Voting from Nearest Tasks: Meta-Vote Pruning of Pre-trained Models for Downstream Tasks*, pages 52–68. 09 2023.

[174] Chelsea Finn, P. Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[175] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[176] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[177] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv*, abs/1708.04552, 2017.

[178] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*, 2020.

[179] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[180] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pages 7294–7305. PMLR, 2020.

[181] Ted Pedersen, Siddharth Patwardhan, Jason Michelizzi, et al. Wordnet:: Similarity-measuring the relatedness of concepts. In *AAAI*, volume 4, pages 25–29, 2004.

[182] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. *arXiv preprint cmp-lg/9406033*, 1994.

[183] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[184] Paul Jaccard. Etude de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:547–579, 01 1901.

[185] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *ArXiv*, abs/1803.02999, 2018.

[186] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34, 2021.

[187] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[188] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[189] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[190] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[191] Matteo Boschini, Lorenzo Bonicelli, Pietro Buzzega, Angelo Porrello, and Simone Calderara. Class-incremental continual learning into the extended der-verse. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[192] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[193] Haiyan Zhao, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Does continual learning equally forget all parameters? In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 42280–42303. PMLR, 23–29 Jul 2023.

[194] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[195] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

[196] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification. *arXiv preprint arXiv:2110.14795*, 2021.

[197] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[198] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[199] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[200] Riccardo Volpi, Diane Larlus, and Grégory Rogez. Continual adaptation of visual representations via domain randomization and meta-learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4443–4453, 2021.

[201] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Sequential learning for domain generalization. In *European Conference on Computer Vision*, pages 603–619. Springer, 2020.

[202] Mohammad Mahdi Derakhshani, Ivona Najdenkoska, Tom van Sonsbeek, Xiantong Zhen, Dwarikanath Mahapatra, Marcel Worring, and Cees GM Snoek.

Lifelonger: A benchmark for continual disease classification. *arXiv preprint arXiv:2204.05737*, 2022.

[203] Woong-Gi Chang, Tackgeun You, Seonguk Seo, Suha Kwak, and Bohyung Han. Domain-specific batch normalization for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 7354–7362, 2019.

[204] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[205] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[206] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[207] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European conference on computer vision*, pages 524–540. Springer, 2020.

[208] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

[209] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey. *arXiv preprint arXiv:2309.01029*, 2023.

[210] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al.

Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[211] Elias Frantar and Dan Alistarh. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

[212] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

[213] Mingyang Zhang, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, Bohan Zhuang, et al. Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*, 2023.

[214] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[215] Yuexiang Zhai, Shengbang Tong, Xiao Li, Mu Cai, Qing Qu, Yong Jae Lee, and Yi Ma. Investigating the catastrophic forgetting in multimodal large language models. *arXiv preprint arXiv:2309.10313*, 2023.

[216] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.