Learning from Imperfect Data
Incremental Learning and Few-shot Learning

Yaoyao Liu

A dissertation submitted towards the degree

Doctor of Engineering (Dr.-Ing.)

of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken, 2023

Date of Colloquium:	Januray 27th, 2023
Dean of the Faculty:	Prof. Dr. Jürgen Steimle
Chair of the Committee:	Prof. Dr. Eddy Ilg
Reviewers:	Prof. Dr. Bernt Schiele
	Prof. Dr. Tinne Tuytelaars
	Prof. Dr. Siyu Tang
Academic Assistant:	Shaoshuai Shi, Ph.D.

To my parents and my fiancée.

ABSTRACT

In recent years, artificial intelligence (AI) has achieved great success in many fields, e.g., computer vision, speech recognition, recommendation engines, and neural language processing. Although impressive advances have been made, AI algorithms still suffer from an important limitation: they rely on large-scale datasets. In contrast, human beings naturally possess the ability to learn novel knowledge from real-world and imperfect data such as a small number of samples or a non-static continual data stream. Attaining such an ability is particularly appealing.

Specifically, an ideal AI system with human-level intelligence should work with the following imperfect data scenarios. 1) The training data distribution changes while learning. In many real scenarios, data are streaming, might disappear after a given period of time, or even can not be stored at all due to storage constraints or privacy issues. As a consequence, the old knowledge is over-written, a phenomenon called catastrophic forgetting. 2) The annotations of the training data are sparse. There are also many scenarios where we do not have access to the specific large-scale data of interest due to privacy and security reasons. As a consequence, the deep models overfit the training data distribution and are very likely to make wrong decisions when they encounter rare cases.

Therefore, the goal of this thesis is to tackle the challenges and develop AI algorithms that can be trained with imperfect data. To achieve the above goal, we study three topics in this thesis. 1) Learning with continual data without forgetting (i.e., incremental learning). 2) Learning with limited data without overfitting (i.e., few-shot learning). 3) Learning with imperfect data in real-world applications (e.g., incremental object detection).

Our key idea is learning to learn/optimize. Specifically, we use advanced learning and optimization techniques to design data-driven methods to dynamically adapt the key elements in AI algorithms, e.g., selection of data, memory allocation, network architecture, essential hyperparameters, and control of knowledge transfer. We believe that the adaptive and dynamic design of system elements will significantly improve the capability of deep learning systems under limited data or continual streams, compared to the systems with fixed and non-optimized elements. More specifically, we first study how to overcome the catastrophic forgetting problem by learning to optimize exemplar data, allocate memory, aggregate neural networks, and optimize key hyperparameters. Then, we study how to improve the generalization ability of the model and tackle the overfitting problem by learning to transfer knowledge and ensemble deep models. Finally, we study how to apply incremental learning techniques to the recent top-performance transformer-based architecture for a more challenging and realistic vision, incremental object detection.

ZUSAMMENFASSUNG

Künstliche Intelligenz (KI) hat in den letzten Jahren in vielen Bereichen große Erfolge erzielt, z. B. Computer Vision, Spracherkennung, Empfehlungsmaschinen und neuronale Sprachverarbeitung. Obwohl beeindruckende Fortschritte erzielt wurden, leiden KI-Algorithmen immer noch an einer wichtigen Einschränkung: Sie sind auf umfangreiche Datensätze angewiesen. Im Gegensatz dazu besitzen Menschen von Natur aus die Fähigkeit, neuartiges Wissen aus realen und unvollkommenen Daten wie einer kleinen Anzahl von Proben oder einem nicht statischen kontinuierlichen Datenstrom zu lernen. Das Erlangen einer solchen Fähigkeit ist besonders reizvoll.

Insbesondere sollte ein ideales KI-System mit Intelligenz auf menschlicher Ebene mit den folgenden unvollkommenen Datenszenarien arbeiten. 1) Die Verteilung der Trainingsdaten ändert sich während des Lernens. In vielen realen Szenarien werden Daten gestreamt, können nach einer bestimmten Zeit verschwinden oder können aufgrund von Speicherbeschränkungen oder Datenschutzproblemen überhaupt nicht gespeichert werden. Infolgedessen wird das alte Wissen überschrieben, ein Phänomen, das als katastrophales Vergessen bezeichnet wird. 2) Die Anmerkungen der Trainingsdaten sind spärlich. Es gibt auch viele Szenarien, in denen wir aus Datenschutz- und Sicherheitsgründen keinen Zugriff auf die spezifischen großen Daten haben, die von Interesse sind. Infolgedessen passen die tiefen Modelle zu stark an die Verteilung der Trainingsdaten an und treffen sehr wahrscheinlich falsche Entscheidungen, wenn sie auf seltene Fälle stoßen.

Daher ist das Ziel dieser Arbeit, die Herausforderungen anzugehen und KI-Algorithmen zu entwickeln, die mit unvollkommenen Daten trainiert werden können. Um das obige Ziel zu erreichen, untersuchen wir in dieser Arbeit drei Themen. 1) Lernen mit kontinuierlichen Daten ohne Vergessen (d. h. inkrementelles Lernen). 2) Lernen mit begrenzten Daten ohne Überanpassung (d. h. Lernen mit wenigen Schüssen). 3) Lernen mit unvollkommenen Daten in realen Anwendungen (z. B. inkrementelle Objekterkennung).

Unser Leitgedanke ist Lernen lernen/optimieren. Insbesondere verwenden wir fortschrittliche Lern- und Optimierungstechniken, um datengesteuerte Methoden zu entwerfen, um die Schlüsselelemente in KI-Algorithmen dynamisch anzupassen, z. B. Auswahl von Daten, Speicherzuweisung, Netzwerkarchitektur, wesentliche Hyperparameter und Steuerung des Wissenstransfers. Wir glauben, dass das adaptive und dynamische Design von Systemelementen die Leistungsfähigkeit von Deep-Learning-Systemen bei begrenzten Daten oder kontinuierlichen Streams im Vergleich zu Systemen mit festen und nicht optimierten Elementen erheblich verbessern wird. Genauer gesagt untersuchen wir zunächst, wie das katastrophale Vergessensproblem überwunden werden kann, indem wir lernen, Beispieldaten zu optimieren, Speicher zuzuweisen, neuronale Netze zu aggregieren und wichtige Hyperparameter zu optimieren. Dann untersuchen wir, wie die Verallgemeinerungsfähigkeit des Modells verbessert und das Overfitting-Problem angegangen werden kann, indem wir lernen, Wissen zu übertragen und tiefe Modelle in Ensembles zusammenzufassen. Schließlich untersuchen wir, wie man inkrementelle Lerntechniken auf die jüngste transformatorbasierte Hochleistungsarchitektur für eine anspruchsvollere und realistischere Vision, inkrementelle Objekterkennung, anwendet.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my most sincere gratitude to my main supervisor, Prof. Bernt Schiele. He is always supportive, insightful, and helpful. During the past three years, he has given me a lot of freedom to choose the research topics that I am interested in. During our meetings, he gave me many insightful suggestions on research, life, and career. He provided me with many opportunities to improve myself, including attending the ELLIS project and the ICVSS summer school. He also provided me with a lot of guidance and encouragement when I was facing difficulties. For these reasons, Bernt is an incredible advisor to have.

I would also like to thank my second supervisor, Prof. Qianru Sun. We have been working together for many years, and she was the mentor of my first research project when I was at the National University of Singapore. She taught me a lot of things, including coding, reviewing, paper writing, and presentations. She also provided me with a lot of help in my life. When I was in Singapore, she recommended participating in jogging and marathon, and introduced a lot of new friends to me. She also guided me during my first academic conference, CVPR 2019. I am truly grateful to have had the opportunity to work with her for these years.

I would like to thank my co-advisors in the ELLIS Ph.D. project, Dr. Christian Rupprecht and Prof. Andrea Vedaldi. They are truly talented and always nice. They gave me a lot of suggestions for my research project during our weekly meetings, spent a lot of time on revising my papers, and provided me with many wonderful future research ideas. They are also very supportive and helpful, providing me with many wonderful opportunities. I would really love to say thank you to them.

I am deeply thankful to all the members of my thesis committee, Prof. Tinne Tuytelaars, Prof. Siyu Tang, Prof. Eddy Ilg, and Shaoshuai Shi, Ph.D. My defense schedule is so tight, and they provide me with a lot of flexibility. During the defense, they gave me a lot of insightful suggestions and comments on my work. I am truly grateful to have them on my Ph.D. committee.

I would like to express my gratitude towards our secretary, Connie Balzert. She is super nice and provided me with a lot of assistance during my life in Germany. We worked together on many tasks, e.g., the high-level computer vision courses. She is really supportive and helpful.

I would also like to thank my wonderful colleagues at Max Planck Institute. We had in-depth discussions, weekly seminars, reading groups, retreats, lunches, and dinners together. They are all very talented and nice. I particularly want to thank Anna, Anurag, Fan, Moritz, David, Mattia, Siddhartha, Yong, Mo, Xudong, Xinting, Wenjia, Li, Shaoshuai, Jan Eric, Zhi, Max, Sukrut, Christopher, Yiting, Yang, Yongqin, Ning, Jiangxin, Dengxin, Stephan, Margret, Steffen, Jovita, Bharat, Julian, Vladimir, Aymen, Gerard, Garvita, Xiaohan, Keyang, Ahmed, Paul, Hejing, Arpit, as well as all other current and former colleagues. I would also like to thank the staff from the IST service team and the international office.

Outside MPI, I specifically want to thank my friends in Saarbrücken and other parts of the world, Miaoran Zhang, Dingfan Chen, Hui-Po Wang, Xueting Li, Rui Ye, Yingwei Li, Xin Wang, Ruibing Hou, Chen Gao, Shu Liu. They gave me a lot of help in both life and research.

Their friendship and support are very important during my Ph.D.

I would also like to thank my advisors and colleagues when I was at the National University of Singapore and Tianjin University. I particularly want to mention Prof. Tat-Seng Chua, Prof. Hanwang Zhang, Prof. An-An Liu, Prof. Yuting Su, Na Zhao, Lizi Liao, Fuli Feng, Xiang Wang, Yunshan Ma, Yixin Cao, Xindi Shang, Lixi Deng, Yujuan Ding, Xiaoyu Du, Xun Yang, Yingwei Wei, Jingjing Chen, Zikun Hu, Junbin Xiao, Zhe Jin, Wenhui Li, Ning Xu, Yao Lu, Xin Zhang, Lei Xu, Nannan Liu, Zhuang Shao, as well as all other former colleagues.

I dedicate this thesis to my parents and my fiancée for their unconditional love and support for my study in Germany. My fiancée, Yingying, is not only my partner in life but also a great company in pursuing my academic dreams. She gave me a lot of suggestions for my papers and talks. She also gave me valuable insights on how to do research. Without their love and help, this dissertation could not have been possible.

CONTENTS

INTRODUCTION 1

Contents

In recent years, artificial intelligence (AI) has achieved great success in many fields. In computer vision, AI algorithms have applications in photo tagging on social media, radiology imaging in healthcare, and self-driving cars within the automotive industry. In speech recognition, AI helps mobile devices to conduct voice searches—e.g., Siri—or improve accessibility for texting. In recommendation engines, AI algorithms can help to discover data trends that can be used to develop more effective cross-selling strategies, which are used by online retailers to make relevant product recommendations to customers during the checkout process [IBM20].

Although impressive advances have been made, AI algorithms still suffer from an important limitation: they rely on the quality and size of the training dataset. A well-built and large-scale dataset is a critical condition for learning a powerful AI model. In most AI algorithms, training data are assumed to satisfy the following two requirements: 1) they contain a massive amount of manually annotated samples [Xia20]; 2) they are independently and identically distributed (i.i.d.), i.e., the data distribution is assumed static [Les20].

In contrast, human beings naturally possess the ability to learn novel knowledge from real-world and imperfect data, such as a small number of samples or a non-static continual data stream. For example, a child can easily learn new concepts from several new things every day without forgetting the old concepts.

Attaining such an ability is particularly appealing and will push the AI models one step further toward human-level intelligence. Specifically, an ideal AI system with human-level intelligence should work with the following *imperfect data* scenarios:

- The training data distribution changes while learning. In many real scenarios, data are streaming, might disappear after a given period of time, or even can't be stored at all due to storage constraints or privacy issues. As a consequence, the old knowledge is over-written, a phenomenon called **catastrophic forgetting** [Alj19].
- The annotations of the training data are sparse. Collecting and manually labeling a large-scale amount of data can be very expensive and time-consuming. There are also

many scenarios where scientists do not have access to the specific large-scale data of interest due to privacy and security reasons. As a consequence, the deep models **overfit the training data** and are very likely to make wrong decisions when they encounter rare circumstances [Yav20].

Therefore, the goal of this thesis aims to tackle the challenges above and develop AI algorithms that can be trained with imperfect data. Our key idea is **learning to learn/optimize**, i.e., using **advanced learning and optimization techniques** to design data-driven methods to **dynamically adapt** the key elements in AI algorithms, e.g., selection of data, memory allocation, network architecture, essential hyperparameters, and control of knowledge transfer. We believe that the adaptive and dynamic design of system elements will significantly improve the capability of deep learning systems under limited data or continual streams, compared to the systems with fixed and non-optimized elements.

The key contributions are briefly summarized below regarding the main topics of this thesis.

- **Learning with continual data without forgetting (i.e., incremental learning).** In Part I, we study how to overcome the catastrophic forgetting problem by learning to optimize exemplar data, combine neural networks, and allocate memory. More specifically,
  - In Chapter 3, we propose a novel training framework by leveraging bilevel optimization to optimize a set of synthesized exemplar data to retain old knowledge.
  - In Chapter 4, we introduce a dynamic memory management strategy that learns to allocate memory budgets for different object classes in different incremental learning phases. The key technique is to utilize reinforcement learning to learn a policy to allocate memory for each class.
  - In Chapter 5, we develop a generic network architecture that learns to combine high-stability and high-plasticity neural network blocks.
  - In Chapter 6, we further balance the stability-plasticity trade-off for different data-receiving settings of incremental learning by introducing an online learning method that can adaptively optimize the tradeoff without knowing the setting as a priori.

- **Learning with limited data without overfitting (i.e., few-shot learning).** In Part II, we study how to improve the generalization ability of the model and tackle the overfitting problem by learning to transfer knowledge and ensemble deep models. Specifically,
  - In Chapter 7, we design a meta-transfer learning framework that allows us to leverage the transferrable pattern learned from existing large-scale tasks using meta-learning.
  - In Chapter 8, we introduce a method that learns to ensemble deep models to reduce the model uncertainty in few-shot learning.

- **Learning in real-world applications (e.g., incremental object detection).** In Part III, we apply incremental learning and few-shot learning algorithms in more challenging real-world applications. Specifically,
  - In Chapter 9, we propose a ContinuaL DEtection TRansformer (CL-DETR), a new method for transformer-based incremental object detection that enables effective usage of knowledge distillation and exemplar replay in a more realistic computer vision task, object detection.

For the rest of this chapter, we discuss each topic and explain our contributions. Then, we provide an outline of the thesis with relevant publications.

1.1 INCREMENTAL LEARNING: LEARNING WITH CONTINUAL DATA WITHOUT FORGETTING

Incremental learning, which is also referred to as continual learning [LAM+22, AKT19, LR17] or lifelong learning [ACT17, CL18, CRRE19], deals with the challenge of acquiring knowledge from training data obtained gradually over time. The core concept of incremental learning is to enable deep learning algorithms to learn from real-world data sources, where not all information is immediately available and needs to be processed sequentially. Incremental learning aims to overcome the limitations of traditional machine learning methods that require access to complete datasets upfront. By embracing incremental learning, deep learning models can adapt and improve as new data becomes available, allowing them to continuously enhance their performance and accumulate knowledge over time [Cha20, Les20].

In incremental learning, the key principle is that only a limited portion of the training data is accessible at any given time, and storing most of the old data may not be feasible due to storage constraints or privacy concerns [Alj19]. As a result, the available memory is often restricted to a small budget, and a subset of the old data is stored as exemplars to be replayed in the future [RKSL17, DCO+20, WZYZ22].

Incremental learning mainly has two settings: *class-incremental* and *task-incremental* settings. In the *class-incremental* setting, the training data for new classes is presented in each phase, and the model is evaluated on a joint test set for all classes encountered so far. In contrast, the *task-incremental* setting involves receiving the training data for a new task in each phase and evaluating the model on the test sets for all previous tasks separately. We primarily focus on the *class-incremental* setting, which is more practical and realistic for real-world scenarios. Moreover, class-incremental learning methods can be easily extended to the task-incremental setting [LH18, RKSL17], making it a necessary condition to address real-life challenges effectively [Les20].

The main challenge in developing an effective incremental learning method is the *catastrophic forgetting* problem of previously learned information when new knowledge is acquired [Rat90, Cha20, Alj19]. To address this issue, popular methods such as Exemplar Replay (ER) and Knowledge Distillation (KD) have been developed. ER methods [RKSL17, LSL+20, LSS21b, WZY+22, CMG+18, LWM+20] involve memorizing some of the past training data or exemplars, and replaying them in subsequent phases to recall old knowledge. On the other hand, KD methods [LH18, DCO+20, HPL+19, HTM+21, ZXG+20] involve introducing regularization terms in the learning objective to preserve previous knowledge when training the model on new data. The key concept behind KD is to encourage the new model's logits or feature maps to be similar to those of the old model. By utilizing these methods, researchers aim to create more effective and efficient incremental learning approaches that can improve upon current models.

Despite the success of ER and KD methods, there are still significant challenges that need to be addressed in incremental learning for image classification. In the following, we will discuss these challenges in detail. After that, we will provide a summary of our contributions in this area.

1.1.1 Challenges in incremental learning

- **Choice of exemplars.** How to choose the most representative exemplars is a fundamental challenge of ER methods. However, existing methods to extract exemplars are based on heuristically designed rules, e.g., the nearest neighbors around the average sample in each class (named herding) [DCO$^+$20, WZYZ22, RKSL17], but turn out to be not particularly effective [LSL$^+$20]. Thus, it is very important to design better exemplar selection methods to effectively leverage the limited memory budget.
- **Imbalanced memory usage.** As the memory budget is limited and the training environment is ever-changing, it is very important to control memory allocation precisely. However, existing methods [DCO$^+$20, WZYZ22, RKSL17] allocate memory between the old and new classes in an arbitrary and static fashion, e.g., 20 per old class vs. $1,300$ per new class for the ImageNet-Full dataset. This causes a serious imbalance between the old and new classes and can exacerbate the problem of catastrophic forgetting. Thus, it is necessary to design a dynamic memory management strategy.
- **Static network architectures.** As the deep networks observe new training data in each phase, the network architecture needs to be adjusted accordingly. However, most of the existing methods [DCO$^+$20, WZYZ22, RKSL17] apply a fixed network architecture for all phases, which is obviously sub-optimal. Some methods [XZ18, HFLR19] try to add new filters and layers to capture the new knowledge, but they violate the memory constraints of incremental learning because they make the number of network parameters grow linearly with the number of phases. Therefore, how to adjust the network architecture dynamically without increasing memory usage is another important challenge.
- **Prefixed hyperparameters.** In incremental learning, key hyperparameters, e.g., KD loss weights, learning rates, and classifier types, significantly influence the final performance [LLSS23a]. However, existing methods prefix hyperparameters for all phases before knowing how data will be received in the future, thus generating suboptimal performance. How to dynamically obtain the optimal hyperparameters for each phase remains an open question.

1.1.2 Our contributions

In this section, we summarize our contributions to addressing the above main challenges of incremental learning for classification.

In Chapter 3, we tackle the first challenge, *sub-optimal exemplars*, by developing an automatic exemplar extraction framework called *mnemonics training*. In this framework, we parameterize the exemplars using image-size parameters and then optimize them in an end-to-end scheme. Using mnemonics training, the incremental learning model in each phase can not only learn the optimal exemplars from the new class data but also adjust the exemplars of previous phases to fit the current data distribution. Empirical results show our mnemonics exemplars yield consistently clear separations among classes, from early to late phases. We can observe that our mnemonics exemplars are mostly located on the boundary of the class data distribution, which is essential to derive high-quality classifiers. Our framework can be combined with different incremental learning baselines and consistently improves performance.

In Chapter 4, we address the second challenge, *imbalanced memory usage*, by proposing reinforced memory management (RMM), which learns an optimal memory management policy for each incremental phase using reinforcement learning. We design a new policy function to contain two sub-functions that propagate two levels of actions in a hierarchical way. The level-1 function determines how to split memory between the old and new data, and the level-2 function determines how to allocate memory for each old class. We conduct extensive experiments by plugging RMM into two top-performing incremental learning methods and testing them on different datasets. Our results show the clear and consistent superiority of our RMM.

In Chapter 5, we address the third challenge, *static network architectures*, by introducing a novel network architecture called Adaptive Aggregation Networks (AANets). In our AANets, we explicitly build two residual blocks (at each residual level): one for maintaining the knowledge of old classes (i.e., the stability) and the other for learning new classes (i.e., the plasticity). We achieve these by allowing these two blocks to have different levels of learnability, i.e., fewer learnable parameters in the stable block but more in the plastic one. We apply aggregation weights to the output feature maps of these blocks, sum them up, and pass the result maps to the next residual level. In this way, we are able to dynamically balance the usage of these blocks by updating their aggregation weights. To achieve auto-updating, we take the weights as hyperparameters and optimize them in an end-to-end manner. For evaluation, we conduct CIL experiments on three widely-used datasets, CIFAR-100, ImageNet-Subset, and ImageNet. We find that many existing CIL methods, e.g., iCaRL [RKSL17], LUCIR [HPL$^+$19], Mnemonics Training [LSL$^+$20], and PODNet [DCO$^+$20], can be directly incorporated in the architecture of AANets, yielding consistent performance improvements.

In Chapter 6, we tackle the fourth challenge, *prefixed hyperparameters*, by formulating the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose a specific algorithm to solve it. We apply local estimated rewards and a classic bandit algorithm Exp3 [ACBFS02] to address the issues when applying online MDP methods to the CIL protocol. Empirically, we find our method performs well consistently. We conduct extensive experiments by plugging our method into three top-performing methods (LUCIR [HPL$^+$19], AANets [LSS21a], and RMM [LSS21b]) and testing them on three benchmarks (i.e., CIFAR-100, ImageNet-Subset, and ImageNet-Full). Our results show consistent improvements of the proposed method in different data-receiving settings of incremental learning.

1.2 FEW-SHOT LEARNING: LEARNING WITH LIMITED DATA WITHOUT OVERFITTING

Few-shot learning aims to train deep models using a limited number of examples with supervised information, such as just five annotated samples per class [WYKN20]. While humans tend to be highly effective in this context, often able to grasp the essential connection between new concepts and their own knowledge, machine learning models still struggle with this task. For example, on the CIFAR-100 dataset, a classification model trained in fully supervised mode achieves 76% accuracy for the 100-class setting [CUH16], whereas the best-performing 1-shot model only achieves an average of 45% for the simpler 5-class setting [SLCS19]. In many real-world applications, large-scale training data may not be available, as in the medical domain, making it desirable to improve machine learning models to handle few-shot settings.

Recent methods solve the few-shot learning problem based on meta-learning [FAL17, FXL18, GFL$^+$18, FFS$^+$18, ZCG$^+$18, SLCS19, AES19, LMRS19, HMX$^+$20]. Meta-learning is a task-level optimization-based method [BBCG92, NM92, TP98]. It aims to transfer experience from similar few-shot learning tasks. Related methods follow a unified training process that contains two loops. The inner loop learns a base learner for an individual task, and the outer loop then uses the validation performance of the learned base learner to optimize the meta-learner. For example, a representative method named Model-Agnostic Meta-Learning (MAML) learns to search for the optimal initialization state to fast adapt a base learner to a new task [FAL17]. Its task-agnostic property makes it possible to generalize to few-shot supervised/semi-supervised learning as well as unsupervised reinforcement learning [FAL17, GFL$^+$18, FXL18, AES19, ZCG$^+$18, RRS$^+$19, RTR$^+$18, LSL$^+$19].

Recent methods address the few-shot learning problem by utilizing meta-learning [FAL17, FXL18, GFL$^+$18, FFS$^+$18, ZCG$^+$18, SLCS19, AES19, LMRS19, HMX$^+$20], a task-level optimization-based method [BBCG92, NM92, TP98]. It aims to transfer knowledge from related few-shot learning tasks. Meta-learning approaches follow a unified training process consisting of two loops. In the inner loop, a base learner is trained for each task, while the outer loop optimizes the meta-learner using the validation performance of the base learner. For instance, Model-Agnostic Meta-Learning (MAML) [FAL17] is a prominent meta-learning method that searches for optimal initializations to enable the base learner to adapt rapidly to a new task. MAML's task-agnostic property allows it to be applied across a range of settings, such as few-shot supervised/semi-supervised learning and unsupervised reinforcement learning [FAL17, GFL$^+$18, FXL18, AES19, ZCG$^+$18, RRS$^+$19, RTR$^+$18, LSL$^+$19].

Despite the success of the meta-learning framework, few-shot learning still poses a significant challenge due to the insufficient number of training examples when training deep neural networks [Xia20]. In the following paragraphs, we will discuss the primary challenges in image classification for few-shot learning and provide an overview of our contributions.

1.2.1 Challenges in few-shot learning

- **Risk of overfitting.** The limited number of training examples from novel classes makes directly fine-tuning deep neural networks problematic, as it can lead to overfitting. In other words, the model may fit perfectly into the small training set of novel classes but fail to generalize to unseen examples. Traditional techniques used in supervised learning may not be effective in few-shot learning due to the issue of overfitting. Thus, preventing overfitting while fine-tuning deep neural networks in few-shot learning is still an unresolved challenge [Xia20].

- **Slow convergence speed.** In meta-learning, convergence is typically slow. It is because we need to compute second-order gradients, which are computationally expensive [JLLP20]. Therefore, how to develop efficient training methods and speed up the convergence speed is another challenge we would like to tackle.

- **Unstable base-learners.** The learning process of a base learner for few-shot tasks is quite unstable [AES19], and often results in high-variance or low-confidence predictions. How to train a stable and robust base learner with a few training samples for novel classes is unsolved.

- **Sub-optimal hyperparameters.** It is well-known that the values of hyperparameters, e.g., for initializing and updating models, are critical for best performance, and are particularly important for few-shot learning. However, existing methods deploy the same hyperparameters for different few-shot tasks, leading to sub-optimal performance. Thus, it is important to produce task-specific hyperparameters for few-shot learning.

1.2.2 Our contributions

In this section, we summarize our contributions to addressing the above main challenges of incremental learning for classification.

In Chapter 7, we tackle the first and second challenges, *overfitting* and *slow convergence speed*, by proposing a novel meta-learning method called *meta-transfer learning (MTL)*. In particular, *transfer* means that deep neural network weights trained on large-scale data can be used in other tasks by two light-weight neuron operations: *Scaling* and *Shifting* (*SS*). "Meta" means that the parameters of these *SS* operations can be viewed as hyper-parameters learned with few-shot learning tasks [MY17, LZCL17, FFS$^+$18]. Our MTL t helps deeper-neural-network-based base-learners converge faster while reducing their probability to overfit when training on a few labeled data only. We further design our *hard task (HT) meta-batch* strategy to offer a challenging but effective learning curriculum, which achieves both faster convergence and stronger performance. We conduct experiments on three few-shot learning benchmarks, namely *mini*ImageNet [VBL$^+$16], *tiered*ImageNet [RTR$^+$18] and Fewshot-CIFAR100 (FC100) [ORL18], and achieve the state-of-the-art performance on both supervised and semi-supervised few-shot learning.

In Chapter 8, we address the third and fourth challenges, *unstable base-learners* and *sub-optimal hyperparameters*, by proposing a novel approach, E^{3}BM, which learns the ensemble of epoch-wise empirical Bayes models for each few-shot task. In E^{3}BM, we utilize the sequence of epoch-wise base-learners (while training a single base learner) as the ensemble to stabilize the base learner and achieve low-variance or high-confidence predictions. Further, we meta-learn hyperprior learners with meta-training tasks and use them to generate task-specific hyperparameters. We plug-in our E^{3}BM to the state-of-the-art few-shot learning methods [FAL17, SLCS19, HMX$^+$20] and obtain consistent performance boosts.

1.3 INCREMENTAL OBJECT DETECTION: LEARNING IN REAL-WORLD APPLICATIONS

Incremental object detection aims to train an object detector with the training samples for different object categories observed in different phases [SSA17]. Similar to incremental learning for image classification, the ability of the trainer to access past data is also restricted in incremental object detection. Incremental object detection is more challenging compared to incremental image classification because the catastrophic forgetting problem occurs in both localization and classification.

Existing works [YZZ$^+$22, ZCS$^+$20, FWY22] utilize popular incremental image classification techniques, *knowledge distillation* [HVD15] and *exemplar replay* [RKSL17] to address the forgetting problem in incremental object detection. However, we empirically find that *knowledge distillation* and *exemplar replay* do not work well if applied directly to recent transformer-based detectors, e.g., Deformable DETR [ZSL$^+$21]. Thus, how to adapt incremental image classification techniques to state-of-the-art transformer-based detectors remains an open problem.

Below, we elaborate on the main challenges in incremental object detection, followed by a summary of our contributions.

1.3.1 Challenges in incremental object detection

- **Unbalanced and contradictory knowledge distillation.** Transformer-based detectors work by testing a large number of object hypotheses in parallel. Because the number of hypotheses is much larger than the typical number of objects in an image, most of them are negative, resulting in an unbalanced knowledge distillation loss. Furthermore, because both old and new object categories can co-exist in any given training image, the KD loss and regular training objective can provide contradictory evidence.

- **Category distribution mismatch of the exemplars.** Exemplar replay methods for image classification try to sample the same number of exemplars for each category. In incremental object detection, this is not a good strategy because the true object category distribution is typically highly skewed. Balanced sampling causes a mismatch between the training and testing data statistics.

1.3.2 Our contributions

In this section, we summarize our contributions to addressing the above main challenges of incremental learning for classification.

In Chapter 9, we address the above challenges by proposing *ContinuaL DEtection TRansformer* (CL-DETR), a new method for transformer-based incremental object detection which enables effective usage of knowledge distillation and exemplar replay in this context. CL-DETR introduces the concept of *Detector Knowledge Distillation* (DKD), selecting the most confident object predictions from the old model, merging them with the ground-truth labels for the new categories while resolving conflicts, and applying standard joint bipartite matching between the merged labels and the current model's predictions for training. This approach subsumes the knowledge distillation loss, applying it only for foreground predictions correctly matched to the appropriate model's hypotheses. CL-DETR also improves exemplar replay by introducing a new calibration strategy to preserve the distribution of object categories observed in the training data. This is obtained by carefully engineering the set of exemplars remembered to match the desired distribution. Furthermore, each phase consists of a main training step followed by a smaller one focusing on better calibrating the model.

We demonstrate CL-DETR by applying it to different transformer-based detectors including Deformable DETR [ZSL^{+}21] and UP-DETR [DCLC21]. Our results on COCO 2017 show that CL-DETR leads to significant improvements compared to the baseline, boosting AP by 4.2 percentage points compared to a direct application of knowledge distillation and exemplar replay to the underlying detector model. We further study and justify our modeling choices via extensive ablations.

1.4 OUTLINE OF THE THESIS

In this section, we provide an overview of the thesis by briefly summarizing each chapter and drawing a connection between them. We also note the respective publications and collaborations with other researchers.

Chapter 2, Related Works. This chapter surveys related works which tackle the challenges of learning with imperfect data with a focus on the three directions of the thesis i.e., incremental learning, few-shot learning, and incremental object detection. We discuss how these works relate to the methods and contributions presented in this thesis. Discussions of related works specific to the following chapters are provided within each chapter.

Part I, Incremental learning: learning with continual data without forgetting

Chapter 3, Mnemonics Training. In this chapter, we tackle the memory limitation problem in incremental learning by proposing a novel and automatic framework called *mnemonics training*. We parameterize exemplars and optimize them in an end-to-end manner to obtain high-quality memory-efficient exemplars.

The content of this chapter corresponds to the CVPR 2020 publication with the title *Mnemonics Training: Multi-Class Incremental Learning without Forgetting* [LSL⁺20]. Yaoyao Liu is the first author of this paper, under the supervision of Prof. Bernt Schiele and Prof. Qianru Sun. It is also a collaboration with An-An Liu and Yuting Su from Tianjin University.

Chapter 4, Reinforced Memory Management. In this chapter, we further tackle the memory limitation problem in incremental learning by developing a dynamic memory management strategy, Reinforced Memory Management, which is optimized for the incremental phases and different object classes.

The content of this chapter corresponds to the NeurIPS 2021 publication with the title *RMM: Reinforced Memory Management for Class-Incremental Learning* [LSS21b]. Yaoyao Liu is the first author of this paper, under the supervision of Prof. Bernt Schiele and Prof. Qianru Sun.

Chapter 5, Adaptive Aggregation Networks. In this chapter, we alleviate the stability-plasticity trade-off in incremental learning by designing a novel network architecture called Adaptive Aggregation Networks (AANets), in which we explicitly build two types of residual blocks at each residual level (taking ResNet as the baseline architecture): a stable block and a plastic block. We aggregate the output feature maps from these two blocks and then feed the results to the next-level blocks. We adopt the aggregation weights in order to balance these two types of blocks, i.e., to balance stability and plasticity, dynamically.

The content of this chapter corresponds to the CVPR 2021 publication with the title *Adaptive Aggregation Networks for Class-Incremental Learning* [LSS21a]. Yaoyao Liu is the first author of this paper, under the supervision of Prof. Bernt Schiele and Prof. Qianru Sun.

Chapter 6, Online Hyperparameter Optimization. In this chapter, we further balance the stability-plasticity trade-off for different data-receiving settings of incremental learning by introducing an online learning method that can adaptively optimize the tradeoff without knowing the setting as a priori. Specifically, we first introduce the key hyperparameters that influence the tradeoff, e.g., knowledge distillation (KD) loss weights, learning rates, and classifier types. Then, we formulate the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose a specific algorithm to solve it.

The content of this chapter corresponds to the AAAI 2023 publication with the title *Online Hyperparameter Optimization for Class-Incremental Learning* [LLSS23a]. Yaoyao Liu is the first author of this paper, under the supervision of Prof. Bernt Schiele and Prof. Qianru Sun. It is also a collaboration with Yingying Li from Caltech.

Part II, Few-shot learning: learning with limited data without overfitting

Chapter 7, Meta-Transfer Learning. In this chapter, we tackle the over-fitting issue in few-shot learning by proposing a novel method called meta-transfer learning (MTL), which learns to adapt a deep neural network for few-shot learning tasks. Specifically, MTL is achieved by learning the scaling and shifting functions of deep neural network weights for each task. In addition, we introduce the hard task (HT) meta-batch scheme as an effective learning curriculum for MTL.

The content of this chapter was published in TPAMI 2022 with the title *Meta-Transfer Learning through Hard Tasks* [SLC$^+$22], which is an extension of our CVPR 2019 publication, *Meta-Transfer Learning for Few-Shot Learning* [SLCS19]. Yaoyao Liu and Prof. Qianru Sun are the co-first authors of this work, under the supervision of Prof. Bernt Schiele. The collaborators include Prof. Tat-Seng Chua from the National University of Singapore and Zhaozheng Chen from Singapore Management University.

Chapter 8, An Ensemble of Epoch-wise Empirical Bayes Models. In this chapter, we improve the robustness of few-shot learning by meta-learning the ensemble of epoch-wise empirical Bayes models. *Epoch-wise* means that each training epoch has a Bayes model whose parameters are specifically learned and deployed. *Empirical* means that the hyperparameters, e.g., used for learning and ensembling the epoch-wise models, are generated by hyperprior learners conditional on task-specific data.

The content of this chapter corresponds to the ECCV 2020 publication with the title *An Ensemble of Epoch-wise Empirical Bayes for Few-shot Learning* [LSS20]. Yaoyao Liu is the first author of this paper, under the supervision of Prof. Bernt Schiele and Prof. Qianru Sun.

Part III, Incremental object detection: learning with imperfect data in real-world applications

Chapter 9, Continual Detection Transformer. In this chapter, we apply the incremental learning setting to a more realistic computer vision task, object detection. We propose a ContinuaL DEtection TRansformer (CL-DETR), a new method for transformer-based incremental object detection that enables effective usage of knowledge distillation and exemplar replay.

The content of this chapter corresponds to the CVPR 2023 publication with the title *Continual Detection Transformer for Incremental Object Detection* [LSVR23]. Yaoyao Liu is the first author of this paper. This work is also part of the ELLIS Ph.D. project, under the supervision of Dr. Christian Rupprecht and Prof. Andrea Vedaldi from Oxford, and Prof. Bernt Schiele from Max Planck Institute for Informatics.

1.5 PUBLICATIONS

The content of this thesis has previously appeared in the following publications, ordered as outlined above:

- **[LSL+20] Liu, Yaoyao**, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. "Mnemonics training: Multi-class incremental learning without forgetting." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
- **[LSS21b] Liu, Yaoyao**, Bernt Schiele, and Qianru Sun. "RMM: Reinforced memory management for class-incremental learning." In Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2021.
- **[LSS21a] Liu, Yaoyao**, Bernt Schiele, and Qianru Sun. "Adaptive Aggregation Networks for Class-Incremental Learning." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.
- **[LLSS23a] Liu, Yaoyao**, Li, Yingying, Bernt Schiele, and Qianru Sun. "Online Hyper-parameter Optimization for Class-Incremental Learning." In Proceedings of AAAI Conference on Artificial Intelligence (AAAI), 2023.
- **[SLC+22]** Sun, Qianru, **Yaoyao Liu** (equal contribution), Zhaozheng Chen, Tat-Seng Chua, and Bernt Schiele. "Meta-transfer learning through hard tasks." IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).
- **[LSS21a] Liu, Yaoyao**, Bernt Schiele, and Qianru Sun. "An Ensemble of Epoch-wise Empirical Bayes for Few-shot Learning." In Proceedings of European Conference on Computer Vision (ECCV), 2020.
- **[LSVR23] Liu, Yaoyao**, Bernt Schiele, Andrea Vedaldi, and Christian Rupprecht. "Continual Detection Transformer for Incremental Object Detection." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

Further contributions were made to the following works not discussed in this thesis:

- **[LSH+21] Liu, Yaoyao**, Qianru Sun, Xiangnan He, An-An Liu, Yuting Su, and Tat-Seng Chua. "Generating face images with attributes for free." IEEE Transactions on Neural Networks and Learning Systems (2020).
- **[LHL+21]** Li, Xinzhe, Jianqiang Huang, **Yaoyao Liu**, Qin Zhou, Shibao Zheng, Bernt Schiele, and Qianru Sun. "Learning to teach and learn for semi-supervised few-shot image classification." Computer Vision and Image Understanding (2021).

- **[LLSS23b]** Zilin Luo, **Liu, Yaoyao**, Bernt Schiele, Andrea Vedaldi, and Christian Rupprecht. “Class-Incremental Exemplar Compression for Class-Incremental Learning.” In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

RELATED WORK 2

Contents

The field of learning with imperfect data covers a wide range of topics, e.g., incremental learning, few-shot learning, semi-supervised learning, and weakly-supervised learning. In this thesis, we mainly focus on incremental learning for image classification, incremental learning for object detection, and few-shot learning for image classification.

In this chapter, we formally define the research problems chosen in this thesis. We present the most relevant and recent developments in the fields and relate them to the contributions of this thesis in the conclusion of each section. The following chapters also discuss related work but targeted to the respective topic of these chapters.

2.1 INCREMENTAL LEARNING

The goal of incremental learning for image classification is to develop a classification model that gradually learns from training data of different classes, phase-by-phase. In each phase, the classifier is trained using both the new class data and the memorized samples of old classes. The model is then evaluated on the test data of both old and new classes. Incremental learning allows the classification model to continually improve its performance as it receives additional data, making it suitable for tasks where new classes are continuously being introduced.

In this section, we first formally define the incremental learning problem and introduce the existing evaluation protocols. Then we discuss popular incremental learning methods and the relations between those and our proposed approaches.

2.1.1 Problem definition

The general incremental learning pipeline is as follows. It usually assumes there are N phases during which the number of classes gradually increases to the maximum [DCO+20, HPL+19, HTM+21, LSL+20]. In the 1-st phase, we observe data $\mathcal{D}_1$, and use it to learn an initial model Θ_1. After this phase, we can only store a small subset of $\mathcal{D}_1$ (i.e., exemplars denoted as $\mathcal{E}_1$) in memory used as replay samples in later phases. In the i-th phase ($i \geq 2$), we get new class data $\mathcal{D}_i$ and load exemplars $\mathcal{E}_{1:i-1} = \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_{i-1}$ from the memory. Then, we initialize Θ_i with Θ_{i-1}, and train it using $\mathcal{E}_{1:i-1} \cup \mathcal{D}_i$. After that, we select exemplars $\mathcal{E}_{1:i}$ from $\mathcal{E}_{1:i-1} \cup \mathcal{D}_i$ and save them in the memory.

2.1.2 Evaluation protocols

There are mainly two evaluation protocols for incremental learning: *task-incremental* setting and *class-incremental* setting.

In the *task-incremental* setting, we regard the training data in each phase as an independent task. Let $\mathcal{Q}_i$ denote the test set for the i-th phase task $\mathcal{D}_i$. At inference time, we evaluate the model on a sequence of test sets, $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_N$, respectively. That means, the model has access to the task IDs and knows which task the model is evaluated on. Most early methods [LH18, CRRE19] considered this setting, which has the clear advantage that methods do not have to discriminate between classes coming from different tasks [MLT+20].

In the *class-incremental* setting, we aim to train a joint model that can classify all classes we have observed so far, thus we do not have access to the task ID at inference time. After N phases, we evaluate the model on a joint test set, $\mathcal{Q}_{1:N}$, which contains the test samples of all classes we have observed. This setting is more challenging compared to the *task-incremental* setting [MLT+20].

In this thesis, we mainly focus on the *class-incremental* setting. The reasons are as follows. 1) The *class-incremental* setting is more realistic and difficult. We cannot access task IDs in many real-world applications. 2) The *class-incremental* methods can also be used to solve the *task-incremental* problems and achieve good performance [MLT+20].

2.1.3 A literature review of incremental learning methods

Incremental learning methods can be broadly categorized into three main categories: regularization-based methods, replay-based methods, and dynamic architecture methods [Xia19].

2.1.3.1 *Regularization-based methods*

Regularization-based methods help mitigate catastrophic forgetting by introducing additional regularization terms into the loss function during training. Recent regularization-based methods can be classified into two categories based on the type of regularization used: knowledge distillation regularization and network parameter regularization.

Knowledge distillation regularization [HVD15] tries to preserve the knowledge capture in a previous version of the model by matching logits [LH18, RKSL17], feature maps [DCO+20], or other information [TCH+20, WZYZ22, SKH21, JKK+22, PZS22] in the new model. Li et al. [LH18] were the pioneers in applying knowledge distillation to incremental learning. Their approach involved forcing the predictions made by the new model to mimic the predictions made by the old model, given the new training data and old exemplars. Following this line, Rannen et al. [TABT17] proposed a different approach. They trained an encoder to capture the important features of each task and then prevented the reconstructions of these features using auto-encoders from changing. Dhar et al. [DSP+19] proposed a novel method that included an information-preserving penalty with attention distillation to overcome forgetting. Hou et al. [HPL+19] introduced a series of new regularization terms, including less-forgetting constraint and inter-class separation, to address the issue of data imbalance between old and new classes. Douillard et al. [DCO+20] proposed an effective spatial-based distillation loss applied throughout the model and also a representation comprising multiple proxy vectors for each object class. Tao et al. [TCH+20] built the framework with a topology-preserving loss to maintain the topology in the feature space. Yu et al. [YTL+20] estimated the drift of previous classes during the training of new classes. Joseph et al. [JKK+22] proposed a regularization term based on an energy manifold for the latent representations and forced previous task latent representations to have low energy and the current task latent representations to have high energy values.

Network parameter regularization mitigates the issue of forgetting by imposing a penalty on the differences between the parameters of the old and new models. Kirkpatrick et al. [KPR+17] first proposed to penalize parameters with different weights by considering the importance of each one. Zenke et al. [ZPG17] estimated the importance of weights in an online manner and took into account all previous tasks by accumulating the parameter specific contribution to changes in the total loss. Chaudhry et al. [CDAT18] combined [ZPG17] and an online version of [KPR+17] with a theoretically grounded KL-divergence based perspective. Lee et al. [LKJ+17] tried to match the moment of the posterior distribution of the neural networks, which are trained on the old and new tasks, respectively. Aljundi et al. [ABE+18] obtained the gradients of the squared L_2 norm of the network outputs and measured the importance of parameters in an unsupervised way.

2.1.3.2 *Replay-based methods*

Replay-based methods overcome the forgetting problem by storing a portion of past training samples or features and using them to “recall" previous knowledge during subsequent phases of learning. Recent replay-based methods can be broadly categorized into three groups, based on the type of saved information used: real image replay, generated image replay, and features replay.

Real image replay methods store real images as exemplars in the memory. Rebuffi et al. [RKSL17] proposed the first replay-based method based on herding [Wel09], which computes the average features for each class and selects the exemplars that can rebuild similar average features as the exemplars. Lopez-Paz et al. [LPR17] chose exemplars solving a constrained optimization problem. Castro et al. [CMG+18] proposed to finetune the final model on the exemplars for a few epochs to further alleviate the class imbalance problem. Wu et al. [WCW+19] tried both herding [Wel09] and random exemplar selection in their paper. Wang et al. [WZY+22] aimed to trade-off between the quality and quantity of exemplars by image compression using the JPEG algorithm, i.e., each exemplar is uniformly downsampled.

The objective of generated image replay methods is to train a generation model to generate exemplars that can be used for replay during subsequent phases of learning. Shin et al. [SLKK17] first proposed to train Generative Adversarial Networks (GANs) [GPAM+20] to produce the exemplars. Wu et al. [WHL+18] used conditional GANs to explicitly generate samples for each class given a one-hot conditioning vector. Zhai et al. [ZCT$^+$19] also used the conditional GAN framework and extended conditioning from one-hot vectors to images.

Features replay methods either save or generate past features and use them for replay during later stages of learning. to overcome forgetting and retain old knowledge. Kemker et al.[KK18] learned an encoder and a decoder for features extracted from a pre-trained model and generated pseudo features to recall the old knowledge. Belouadah et al. [BP19] proposed to leverage a second memory to store statistics of old classes in rather compact formats. Iscen et al. [IZLS20] preserved low dimensional features instead of raw instances to reduce the storage overhead.

2.1.3.3 *Dynamic architecture methods*

Dynamic architecture methods maintain the learned parameters associated with prior knowledge and assign new parameters in various forms, such as unused parameters and supplementary network filters, to acquire new knowledge [YXH21]. Rusu et al. [RRD$^+$16] proposed "progressive networks" to integrate the desiderata of different tasks directly into the networks. Abati et al. [ATB$^+$20] equipped each convolution layer with task-specific gating modules that select specific filters to learn each new task. Rajasegaran et al.[RHK$^+$19] progressively chose the optimal paths for the new task while encouraging parameter sharing across tasks. Xu et al. [XZ18] searched for the best neural network architecture for each coming task by leveraging reinforcement learning strategies. Hung et al. [HTW$^+$19] proposed a compaction and selection/expansion mechanism that prunes the deep model and expands the architecture alternatively with selective weight sharing.

Recently, dynamic architectures with expandable representations have demonstrated state-of-the-art performance for incremental learning. The key concept is to augment the feature dimensions using an additional network in each phase to generate more distinctive features for classification. To manage memory usage, these methods typically employ network pruning and knowledge distillation to compress the network size. Yan et al. [YXH21] proposed to freeze the previously learned representation and augment it with additional feature dimensions from a new learnable feature extractor. They also introduced a channel-level mask-based pruning strategy to control the final network size. Wang et al. [WZYZ22] suggested a method for dynamic module expansion, wherein new modules are added to capture the residuals between the target and the output of the original model. They subsequently employ an efficient distillation strategy to eliminate redundant parameters and feature dimensions, thereby preserving the single backbone model.

2.1.4 Connections to our work

In Chapter 3, we introduce an effective replay-based method, mnemonics training, which parameterizes exemplars and optimizes them in an end-to-end manner to obtain high-quality memory-efficient exemplars. Related generated image replay methods [SLKK17, KGL17, VVPL17] used Generative Adversarial Networks (GAN) [GPAM$^+$14] to generate old samples in each new phase for data replaying, and good results were obtained in the multi-task incremental setting. However, their performance strongly depends on the GAN models, which are notoriously hard to train. Moreover, storing GAN models requires memory, so these methods might not be applicable to MCIL with a strict memory budget. Our *mnemonics* exemplars are optimizable and can be regarded as synthesized, while our approach is based on the direct parameterization of exemplars without training extra models.

In Chapter 4, we propose a dynamic memory management strategy, Reinforced Memory Management, which is optimized for the incremental phases and different object classes. Existing replay-based methods [RKSL17, CMG$^+$18, WZY$^+$22] allocate memory between the old and new classes in an arbitrary and static fashion, e.g., 20 per old class *vs.* $1,300$ per new class for the ImageNet-Full dataset. This causes a serious imbalance between the old and new classes and can exacerbate the problem of catastrophic forgetting. In contrast, our approach learns an optimal memory management policy for each incremental phase using reinforcement learning. Thus, our method achieves better memory allocation and alleviates the class imbalance problem in incremental learning.

In Chapter 5, we design a novel network architecture called Adaptive Aggregation Networks (AANets) to alleviate the stability-plasticity trade-off in incremental learning. Our method is closely related to dynamic architecture methods [ATB$^+$20, RHK$^+$19, XZ18]. However, in these methods, the size of the network increases with the number of stages. In contrast, our approach does not continuously increase the network size. We validate in the experiments that under a strict memory budget, our approach can surpass many related methods, and plug-in versions of these related methods can bring consistent performance improvements.

In Chapter 6, we further balance the stability-plasticity trade-off by introducing an online learning method that can adaptively optimize the key hyperparameters in incremental learning. Related regularization-based methods [LH18, HPL$^+$19, DCO$^+$20] are either too stable or too plastic to perform well in different data-receiving settings of incremental learning. Our method learns an online policy to generate hyperparameters that balance stability and plasticity. Therefore, our method performs can adapt to different data-receiving settings and achieve better performance.

2.2 FEW-SHOT LEARNING

The goal of few-shot learning is to learn deep neural networks using only a limited number of labeled data, e.g., around five samples per class. Due to the vast number of model parameters in deep neural networks, it is not possible to train a model from scratch with such a small amount of data. Therefore, in few-shot learning, we assume that some base classes have sufficient labeled data available. The task becomes how to learn a model from these base classes that can generalize well to novel classes with only a few labeled data available [Xia20].

Recent few-shot learning related works [VBL$^+$16, RL17, FAL17, ORL18, RRS$^+$19] usually evaluate their method using the *meta-learning* framework. The framework involves training the model on multiple few-shot tasks, also known as episodes, which are generated from the base classes possessing an adequate amount of labeled data. Following this, the model's performance is evaluated on test episodes derived from novel classes.

In this section, we first formally define the few-shot learning problem and introduce the existing evaluation protocols. Then we discuss popular few-shot learning methods and the relations between those and our proposed methods.

2.2.1 Problem definition and evaluation protocol

In this section, we briefly introduce the unified episodic formulation in meta-learning, following related works [VBL+16, RL17, FAL17, ORL18, RRS+19]. Then, we introduce the task-level data denotations used in two phases, i.e., meta-train and meta-test.

The episodic formulation was proposed for tackling few-shot tasks first in [VBL+16]. It is different from traditional image classification, in three aspects: (1) the main phases are not train and test but meta-train and meta-test, each of which includes training and testing; (2) the samples in meta-train and meta-test are not data points but episodes, and each episode is a few-shot classification task; and (3) the objective is not classifying unseen data points but to fast adapt the meta-learned experience or knowledge to the learning of a new few-shot classification task.

The denotations of two phases, meta-train and meta-test, are as follows. A meta-train example is a classification task $\mathcal{T}$ sampled from a distribution $p(\mathcal{T})$. $\mathcal{T}$ is called an episode, including a training split $\mathcal{T}^{(tr)}$ to optimize the base-learner, i.e., the classifiers in our model, and a test split $\mathcal{T}^{(te)}$ to optimize the meta-learner, i.e., the scaling and shifting parameters in our model. In particular, meta-train aims to learn from a number of episodes $\{\mathcal{T}\}$ sampled from $p(\mathcal{T})$. An unseen episode $\mathcal{T}_{unseen}$ in meta-test will start from that experience of the meta-learner and adapt the base-learner. The final evaluation is done by testing a set of unseen data points in $\mathcal{T}^{(te)}_{unseen}$.

Meta-train phase. This phase aims to learn a meta-learner from multiple episodes. In each episode, meta-training has a two-stage optimization. Stage 1 is called base-learning, where the cross-entropy loss is used to optimize the parameters of the base-learner. Stage 2 contains a feed-forward test on episode test data points. The test loss (also called meta loss) is used to optimize the parameters of the meta-learner. Specifically, given an episode $\mathcal{T} \in p(\mathcal{T})$, the base-learner $\theta_{\mathcal{T}}$ is learned from episode training data $\mathcal{T}^{(tr)}$ and its corresponding loss $\mathcal{L}_{\mathcal{T}}(\theta_{\mathcal{T}}, \mathcal{T}^{(tr)})$. After optimizing this loss, the base-learner has parameters $\tilde{\theta}_{\mathcal{T}}$. Then, the meta-learner is updated using meta loss $\mathcal{L}_{\mathcal{T}}(\tilde{\theta}_{\mathcal{T}}, \mathcal{T}^{(te)})$. After meta-training on all episodes, the meta-learner is optimized by meta losses $\{\mathcal{L}_{\mathcal{T}}(\tilde{\theta}_{\mathcal{T}}, \mathcal{T}^{(te)})\}_{\mathcal{T}\in p(\mathcal{T})}$. Therefore, the number of meta-learner updates equals the number of episodes.

Meta-test phase. This phase aims to test the performance of the trained meta-learner for fast adaptation to unseen episodes. Given $\mathcal{T}_{unseen}$, the meta-learner $\tilde{\theta}_{\mathcal{T}}$ teaches the base-learner $\theta_{\mathcal{T}_{unseen}}$ to adapt to the objective of $\mathcal{T}_{unseen}$ by some means, e.g. through initialization [FAL17]. Then, the test result on $\mathcal{T}^{(te)}_{unseen}$ is used to evaluate the meta-learning approach. If there are multiple unseen episodes $\{\mathcal{T}_{unseen}\}$, the average result on $\{\mathcal{T}^{(te)}_{unseen}\}$ will be the final evaluation.

2.2.2 A literature review of few-shot learning methods

We can divide meta-learning methods into three categories. 1) *Metric learning* methods [VBL+16, SSZ17, SYZ+18, LDM+19, YHZS20, HCB+19, DSM19, PDH18] learn a similarity space in which learning is particularly efficient for few-shot training examples. Examples of distance metrics include cosine similarity [VBL+16, CLK+19, PDH18], Euclidean distance to the prototypical representation of a class [SSZ17], CNN-based relation module [SYZ+18], ridge regression based [BHTV19], and graph model based [SE18, LLP+19]. Some recent works also tried to generate task-specific feature representation for few-shot episodes based on metric learning, like [LDM$^+$19, LED$^+$19] 2) *Memory network* methods [MY17, ORL18, MRCA18, PDH18, BRCŚ$^+$17] learn to store "experience" when learning seen tasks and then generalize it to unseen tasks. The key idea is to design a model specifically for fast learning with a few training steps. A family of model architectures uses external memory storage, including Neural Turing Machines [SBB$^+$16], Meta Networks [MY17], Neural Attentive Learner (SNAIL) [MRCA18], and Task Dependent Adaptive Metric (TADAM) [ORL18]. For test, general meta memory and specific task information are combined to make predictions in neural networks. 3) *Gradient descent* based meta-learning methods [FAL17, FXL18, AES19, RL17, LC18, GFL$^+$18, ZCG$^+$18, SLCS19, LMRS19, HMX$^+$20, LC18] intend for adjusting the optimization algorithm so that the model can converge within a small number of optimization steps (with a few examples). The optimization algorithm can be explicitly modeled with two learning loops that outer-loop has a *meta-learner* that learns to adapt an inner-loop *base-learner* (to few-shot examples) through different tasks. For example, Ravi *et al.* [RL17] introduced a method that compresses the base-learners' parameter space in an LSTM meta-learner. Rusu *et al.* [RRS$^+$19] designed a classifier generator as the meta-learner which outputs parameters for each specific base-learning task. Finn *et al.* [FAL17] proposed a meta-learner called MAML that learns to effectively initialize a base-learner for a new task. Lee *et al.* [LC18] proposed MT-net, where the meta-learner determines a sub-space and a corresponding metric that task-specific learners can learn in, thus setting the degrees of freedom of task-specific learners to an appropriate amount. Lee *et al.* [LMRS19] presented a meta-learning approach with convex base-learners for few-shot tasks. Other related works in this category include Hierarchical Bayesian model [GFL$^+$18], Bilevel Programming [FFS$^+$18], and GAN based meta model [ZCG$^+$18]. Hu et al. [HMX$^+$20] proposed to update base-learner with synthetic gradients generated by a variational posterior conditional on unlabeled data.

Among them, MAML is a fairly general optimization algorithm, compatible with any model that learns through gradient descent. Its meta-learner optimization is done by gradient descent using the validation loss of the base-learner.

2.2.3 Connections to our work

In Chapter 7, we propose a novel few-shot learning method called meta-transfer learning (MTL), which learns to adapt a deep neural network for few-shot learning tasks. Specifically, MTL is achieved by learning the scaling and shifting functions of deep neural network weights for each task. In addition, we introduce the hard task (HT) meta-batch scheme as an effective learning curriculum for MTL. Our method differs from existing few-shot learning methods in the following two aspects. 1) Our method transfer the knowledge from the pre-trained model using the scaling and shifting functions of deep neural network weights, while the existing methods [ZZW$^+$21, ZCLS22] directly finetune the pre-trained model. Thus, our method is less prone to overfitting to the novel few-shot tasks. 2) We first introduce the hard task learning curriculum for few-shot learning, which significantly improves the convergence speed during meta-training.

In Chapter 8, we improve the robustness of few-shot learning by meta-learning the ensemble of epoch-wise empirical Bayes models. Our approach is closely related to gradient descent based methods [FAL17, AES19, SLCS19, SLCS19, SLC$^+$22, HMX$^+$20]. An important difference is that we learn how to combine an ensemble of epoch-wise base-learners and how to generate efficient hyperparameters for base-learners, while other methods such as MAML [FAL17], MAML++ [AES19], LEO [RRS+19], MTL [SLCS19, SLC+22], and SIB [HMX+20] use a single base-learner.

2.3 INCREMENTAL OBJECT DETECTION

Incremental object detection aims to train a detection model with the training data of different categories gradually coming phase-by-phase. In each phase, the detector is re-trained on new category data and old category memory, then evaluated on the test data for all observed categories.

In this section, we first formally define the incremental object detection problem and introduce the existing evaluation protocols. Then we discuss popular incremental object detection methods and the relations between those and our proposed method.

2.3.1 Problem definition and evaluation protocol

In incremental object detection, the goal is to train a detector in phases, where in each phase the model is only given annotations for a subset of the object categories. Formally, let $\mathcal{D} = \{(x, y)\}$ be a dataset of images x with corresponding object annotations y, such as COCO 2017 [LMB+14], and let $\mathcal{C} = \{1, \ldots, C\}$ be the set of object categories. We adopt such a dataset for benchmarking IOD as follows. First, we partition $\mathcal{D}$ and $\mathcal{C}$ into M subsets $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_M$ and $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_M$, one for each training phase. For each phase i, we modify the samples $(x, y) \in \mathcal{D}_i$ so that y only contains annotations for objects of class $\mathcal{C}_i$ and drop the others.

In phase i of training, the model is only allowed to observe images $\mathcal{D}_i$ with annotations for objects of types $\mathcal{C}_i \subset \mathcal{C}$. Notably, images can and do contain objects of any possible type $\mathcal{C}$, but only types $\mathcal{C}_i$ are annotated in this phase. After phase i is complete, training switches to the next phase $i + 1$, so the model observes different images $\mathcal{D}_{i+1}$ and annotations for objects of different types $\mathcal{C}_{i+1}$.

For exemplar replay, we relax this training protocol and allow the model to memorize a small number of *exemplars* $\mathcal{E}_i \subset \mathcal{D}_i$ from the previous phases. In this case, the model is trained on the union $\mathcal{D}_i \cup \mathcal{E}_{1:i-1}$ where $\mathcal{E}_{1:i-1} = \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_{i-1}$ forms the exemplar memory.

The evaluation process is as follows. At the end of each phase, the detection model will be evaluated on a test set that contains the objects of all categories observed so far. Following [FWY22], the standard COCO metrics are used for evaluation, i.e., AP, AP_{50}, AP_{75}, AP_S, AP_M, and AP_L [FWY22].

2.3.2 A literature review of incremental object detection methods

Incremental object detection applies incremental learning to object detection specifically. This is more challenging than incremental image classification, as images can contain multiple objects, both of old and new types, with only the new types being annotated in any given training phase. Both knowledge distillation [LH18, HPL+19, DCO+20] and exemplar replay [RKSL17, CMG+18] methods have been applied to detection before. [SSA17] applies KD to the output of Faster R-CNN [Gir15]. Inspired by this, recent incremental object detection methods extended the knowledge distillation framework to other detectors (e.g., Faster-RCNN [RHGS17] and GFL [LWW+20]) by adding KD terms on the intermediate feature maps [YZZ^+22, ZCS^+20, FWY22] and region proposal networks [CYC19, HFJT19, PZL20]. [JKKB21] proposes instead to store a set of exemplars and fine-tune the model on the exemplars after each incremental step. [LYR^+20] proposes an adaptive sampling strategy to achieve more efficient exemplar selection for incremental object detection.

2.3.3 Connections to our work

In Chapter 9, we apply the incremental learning setting to recent top-performing transformer-based detection models, e.g., Deformable DETR [ZSL^+21]. Existing incremental object detection methods are designed based on conventional detectors such as Faster-RCNN [RHGS17] and GFL [LWW^+20]. Empirically, we find that a direct application of existing knowledge distillation and exemplar replay methods to current state-of-the-art transformer-based detectors such as Deformable DETR [ZSL^+21] and UP-DETR [DCLC21] does not work well. In contrast, we propose fixes to this issue by proposing detector knowledge distillation and distribution preserving calibration to improve knowledge distillation and exemplar replay, respectively.

I

Incremental Learning: Learning with Continual Data without Forgetting

In the first part of this thesis, we focus on incremental learning and study how to overcome the catastrophic forgetting problem by learning to optimize exemplar data, combine neural networks, and allocate memory. More specifically,

Specifically, in Chapter 3, we study how to efficiently select exemplars for incremental learning. We parameterize exemplars and optimize them in an end-to-end manner to obtain high-quality memory-efficient exemplars.

In Chapter 4, we further tackle the memory limitation problem in incremental learning by developing a dynamic memory management strategy, Reinforced Memory Management, which is optimized for the incremental phases and different object classes.

In Chapter 5, we intend to find a suitable dynamic network architecture for incremental learning. To this end, we propose Adaptive Aggregation Networks (AANets), which maintain two types of residual blocks, stable and plastic blocks, to alleviate the stability-plasticity trade-off in incremental learning. Compared to previous dynamic architecture methods, our method does not continuously increase the network size.

In Chapter 6, we further balance the stability-plasticity trade-off by introducing a method that can adaptively optimize the key hyperparameters in the incremental learning process. We formulate the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose an online learning algorithm to solve it.

3 LEARNING TO OPTIMIZE EXEMPLAR DATA

Contents

Incremental learning aims to learn new concepts by incrementally updating a model trained on previous concepts. However, there is an inherent trade-off to effectively learning new concepts without catastrophically forgetting previous ones. To alleviate this issue, it has been proposed to keep around a few examples of the previous concepts. However, the effectiveness of this approach heavily depends on the representativeness of these examples. In this chapter, we propose a novel and automatic framework we call *mnemonics training*, where we parameterize exemplars and make them optimizable in an end-to-end manner. We train the framework through bi-level optimizations, i.e., model-level and exemplar-level. We conduct extensive experiments on three incremental learning benchmarks, CIFAR-100, ImageNet-Subset, and ImageNet, and show that using *mnemonics* exemplars can surpass the state-of-the-art by a large margin. Interestingly and quite intriguingly, the *mnemonics* exemplars tend to be on the boundaries between different classes.

This chapter is based on [LSL$^+$20]. As the first author of [LSL$^+$20], Yaoyao Liu conducted all experiments and was the main writer. This paper was selected as *an oral paper* in CVPR 2020 (4%) and has received *more than 170 citations*. It has been recognized in a widely-cited review paper [MLT$^+$20] as an exciting new direction.

3.1 INTRODUCTION

Natural learning systems, such as humans, inherently work in an incremental manner as the number of concepts increases over time. They naturally learn new concepts while not forgetting previous ones. In contrast, current machine learning systems, when continuously

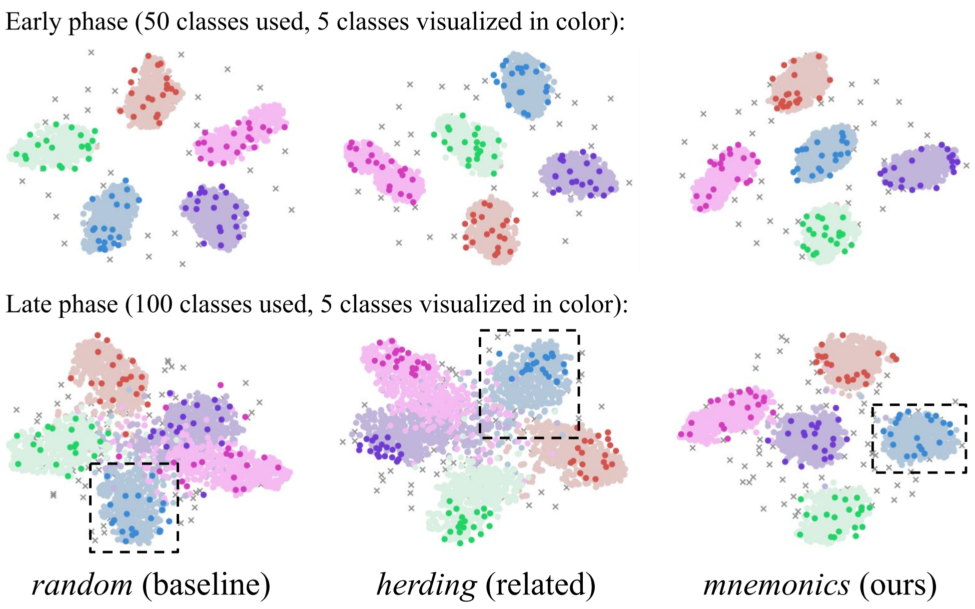


Figure 3.1: The t-SNE [MH08] results of three exemplar methods in two phases. The original data of 5 colored classes occur in the early phase. In each colored class, deep-color points are exemplars, and light-color ones show the original data as a reference of the real data distribution. Gray crosses represent other participating classes, and each cross is for one class. We have two main observations. (1) Our approach results in much clearer separation in the data, than *random* (where exemplars are randomly sampled in the early phase) and *herding* (where exemplars are nearest neighbors of the mean sample in the early phase) [RKSL17, HPL+19, WCW+19, CMG+18]. (2) Our learned exemplars are mostly located on the boundaries between classes.

updated using novel incoming data, suffer from catastrophic forgetting (or catastrophic interference), as the updates can override knowledge acquired from previous data [MC89, MH93, Rat90, SLKK17, KMA+18]. This is especially true for class incremental learning (CIL), where one cannot replay all previous inputs. Catastrophic forgetting, therefore, becomes a major problem for CIL systems.

Motivated by this, a number of works have recently emerged [RKSL17, LH18, HPL+19, WCW+19, CMG+18, LCL19]. Rebuffi et al. [RKSL17] firstly defined a protocol for evaluating CIL methods, i.e., to tackle the image classification task where the training data for different classes comes in sequential training phases. As it is neither desirable nor scaleable to retain all data from previous concepts, in their protocol, they restrict the number of exemplars that can be kept around per class, e.g., only 20 exemplars per class can be stored and passed to the subsequent training phases. These "20 exemplars" are important to CIL as they are the key resource for the model to refresh its previous knowledge. Existing methods to extract exemplars are based on heuristically designed rules, e.g., nearest neighbors around the average sample in each class (named *herding* [Wel09]) [RKSL17, HPL+19, WCW+19, CMG+18], but turn out to be not particularly effective. For example, iCaRL [RKSL17] with *herding* sees an accuracy drop of around 25% in predicting 50 previous classes in the last phase (when the number of classes increases to 100) on CIFAR-100, compared to the upperbound performance of using all examples. A t-SNE visualization of *herding* exemplars is given in Figure 3.1, and shows that the separation between classes becomes weaker in later training phases.

In this chapter, we address this issue by developing an automatic exemplar extraction framework called *mnemonics training*, where we parameterize the exemplars using image-size parameters, and then optimize them in an end-to-end scheme. Using *mnemonics training*, the CIL model in each phase can not only learn the optimal exemplars from the new class data but also adjust the exemplars of previous phases to fit the current data distribution. As demonstrated in Figure 3.1, *mnemonics* exemplars yield consistently clear separations among classes, from early to late phases. When inspecting individual classes (as e.g. denoted by the black dotted frames in Figure 3.1 for the "blue" class), we observe that the *mnemonics* exemplars (dark blue dots) are mostly located on the boundary of the class data distribution (light blue dots), which is essential to derive high-quality classifiers.

Technically, *mnemonics training* has two models to optimize, i.e., the conventional model and the parameterized *mnemonics* exemplars. The two are not independent and can not be jointly optimized, as the exemplars learned in the current phase will act as the input data of later-phase models. We address this issue using a bi-level optimization program (BOP) [SMD18, MVL$^+$19] that alternates the learning of two levels of models. We iterate this optimization through the entire incremental training phases. In particular, for every single phase, we perform a local BOP that aims to distill the knowledge of new class data into the exemplars. First, a temporary model is trained with exemplars as input. Then, a validation loss on new class data is computed, and the gradients are back-propagated to optimize the input layer, i.e., the parameters of the *mnemonics* exemplars. Iterating these two steps allows for deriving representative exemplars for later training phases. To evaluate the proposed *mnemonics* method, we conduct extensive experiments for FOUR different baseline architectures and on THREE CIL benchmarks – CIFAR-100, ImageNet-Subset, and ImageNet. Our results reveal that *mnemonics training* consistently achieves top performance compared to baselines, e.g., 17.9% and 4.4% higher than *herding*-based iCaRL [RKSL17] and LUCIR [HPL$^+$19], respectively, in the 25-phase setting on the ImageNet [RKSL17].

Our contributions include: (1) A novel *mnemonics training* framework that alternates the learning of exemplars and models in a global bilevel optimization program, where bilevel includes *model-level* and *exemplar-level*; (2) A novel local bilevel optimization program (including *meta-level* and *base-level*) that trains exemplars for new classes as well as adjusts exemplars of old classes in an end-to-end manner; (3) In-depth experiments, visualization and explanation of *mnemonics* exemplars in the feature space.

3.2 RELATED WORK

In this section, we discuss related works on bi-level optimization problems. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Bi-level optimization problem (BOP) [WZTE18, VSV52, GPAM$^+$14] aims to solve two levels of problems in one framework where the A-level problem is the constraint to solve the B-level problem. It can be traced back to the Stackelberg competition [VSV52] in the area of game theory. Nowadays, it is widely applied in the area of machine learning. For instance, Training GANs [GPAM$^+$14] can be formulated as a BOP with two optimization problems: maximizing the reality score of generated images and minimizing the real-fake classification loss. Meta-learning [FAL17, SLCS19, ZLL$^+$19b, LSL$^+$19, ZCLS20, SLC$^+$22] is another BOP in which a meta-learner is optimized subject to the optimality of the base-learner. Recently, MacKay et al. [MVL+19] formulated hyperparameter optimization as a BOP where the optimal model parameters in a certain time phase depend on hyperparameters, and vice versa. In this work, we introduce a global BOP that alternatively optimizes the parameters of the CIL models and the *mnemonics* exemplars across all phases. Inside each phase, we exploit a local BOP to learn (or adjust) the *mnemonics* exemplars specific to the new class (or the previous classes).

3.3 METHODOLOGY

As illustrated in Figure 3.2, the proposed *mnemonics* training alternates the learning of classification models and *mnemonics* exemplars across all phases, where *mnemonics* exemplars are not just data samples but can be optimized and adjusted online. First, we elaborate on the denotations (Section 3.3.1) and introduce the distillation loss used in related works (Section 3.3.2). Then, we formulate this alternative learning with a global *Bilevel Optimization Program (BOP)* composed of *model-level* and *exemplar-level* problems (Section 3.3.3), and provide the solutions in Section 3.3.4 and Section 3.3.5, respectively.

3.3.1 Denotations for CIL

Assume there are $N+1$ phases (i.e, 1 initial phase and N incremental phases) in the class-incremental learning system. In the initial (the 0-th) phase, we learn the model Θ_0 on data D_0 using a conventional classification loss, e.g. cross-entropy loss, and then save Θ_0 to the memory of the system. Due to the memory limitation, we can not keep the entire D_0, but instead, we select and store a handful of exemplars $\mathcal{E}_0$ (evenly for all classes) as a replacement of D_0 with $|\mathcal{E}_0| \ll |D_0|$. In the i-th incremental phase, we denote the previous exemplars $\mathcal{E}_0 \sim \mathcal{E}_{i-1}$ shortly as $\mathcal{E}_{0:i-1}$. We load Θ_{i-1} and $\mathcal{E}_{0:i-1}$ from the memory, and then use $\mathcal{E}_{0:i-1}$ and the new class data D_i to train Θ_i initialized by Θ_{i-1}. During training, we use a classification loss and a CIL-specific distillation loss [LH18, RKSL17]. After each phase, the model is evaluated on unseen data for all classes observed by the system so far. We report the average accuracy over all $N+1$ phases as the final evaluation, following [RKSL17, WCW+19, HPL+19].

3.3.2 Distillation Loss for CIL

Distillation loss was originally proposed in [HVD15] and was applied to CIL in [LH18, RKSL17]. It encourages the new Θ_i and previous Θ_{i-1} to maintain the same prediction ability on old classes. Assume there are K classes in $D_{0:i-1}$. Let x be an image in D_i. $\hat{p}_k(x)$ and $p_k(x)$ denote the prediction logits of the k-th class from Θ_{i-1} and Θ_i, respectively. The distillation loss is formulated as

$$\mathcal{L}_d(\Theta_i;\Theta_{i-1};x) = -\sum_{k=1}^{K} \hat{\pi}_k(x)\log\pi_k(x), \tag{3.1a}$$

$$\hat{\pi}_k(x) = \frac{e^{\hat{p}_k(x)/\tau}}{\sum_{j=1}^{K} e^{\hat{p}_j(x)/\tau}}, \quad \pi_k(x) = \frac{e^{p_k(x)/\tau}}{\sum_{j=1}^{K} e^{p_j(x)/\tau}}, \tag{3.1b}$$

where τ is a temperature scalar set to be greater than 1 to assign larger weights to smaller values.

We use the softmax cross entropy loss as the Classification Loss $\mathcal{L}_c$. Assume there are M classes in $D_{0:i}$,. This loss is formulated as

$$\mathcal{L}_c(\Theta_i; x) = -\sum_{k=1}^{K+M} \delta_{y=k} \log p_k(x), \tag{3.2}$$

where y is the ground truth label of x, and $\delta_{y=k}$ is an indicator function.

3.3.3 Global BOP

In CIL, the classification model is trained incrementally in each phase on the union of new class data and old class *mnemonics* exemplars. In turn, based on this model, the new class *mnemonics* exemplars (i.e., the parameters of the exemplars) are trained before omitting new class data. In this way, the optimality of the model derives a constraint to optimizing the exemplars, and vice versa. We propose to formulate this relationship with a global BOP in which each phase uses the optimal model to optimize exemplars, and vice versa.

Specifically, in the i-th phase, our CIL system aims to learn a model Θ_i to approximate the *ideal* one named Θ_i^* which minimizes the classification loss $\mathcal{L}_c$ on both D_i and $D_{0:i-1}$, i.e.,

$$\Theta_i^* = \arg\min_{\Theta_i} \mathcal{L}_c(\Theta_i; D_{0:i-1} \cup D_i). \tag{3.3}$$

Since $D_{0:i-1}$ was omitted (i.e., not accessible) and only $\mathcal{E}_{0:i-1}$ is stored in memory, we approximate $\mathcal{E}_{0:i-1}$ towards the optimal replacement of $D_{0:i-1}$ as much as possible. We formulate this with the global BOP, where "global" means operating through all phases, as follows,

$$\min_{\Theta_i} \mathcal{L}_c(\Theta_i; \mathcal{E}^*_{0:i-1} \cup D_i) \tag{3.4a}$$

$$\text{s.t. } \mathcal{E}^*_{0:i-1} = \arg\min_{\mathcal{E}_{0:i-1}} \mathcal{L}_c\big(\Theta_{i-1}(\mathcal{E}_{0:i-1}); \mathcal{E}_{0:i-2} \cup D_{i-1}\big), \tag{3.4b}$$

where $\Theta_{i-1}(\mathcal{E}_{0:i-1})$ denotes that Θ_{i-1} was fine-tuned on $\mathcal{E}_{0:i-1}$ to reduce the bias caused by the imbalanced sample numbers between new class data D_{i-1} and old exemplars $\mathcal{E}_{0:i-2}$, in the $i-1$-th phase. Please refer to the last paragraph in Section 3.3.5 for more details. In the following, Problem 3.4a and Problem 3.4b are called *model-level* and *exemplar-level* problems, respectively.

3.3.4 Model-level problem

As illustrated in Figure 3.2, in the i-th phase, we first solve the *model-level* problem with the *mnemonics* exemplars $\mathcal{E}_{0:i-1}$ as part of the input and previous Θ_{i-1} as the model initialization. According to Problem 3.4, the objective function can be expressed as

$$\mathcal{L}_{\text{all}} = \lambda \mathcal{L}_c(\Theta_i; \mathcal{E}_{0:i-1} \cup D_i) + (1-\lambda)\mathcal{L}_d(\Theta_i; \Theta_{i-1}; \mathcal{E}_{0:i-1} \cup D_i), \tag{3.5}$$

where λ is a scalar manually set to balance between $\mathcal{L}_d$ and $\mathcal{L}_c$ (introduced in Section 3.3.2). Let α_1 be the learning rate, Θ_i is updated with gradient descent as follows,

$$\Theta_i \leftarrow \Theta_i - \alpha_1 \nabla_\Theta \mathcal{L}_{\text{all}}. \tag{3.6}$$

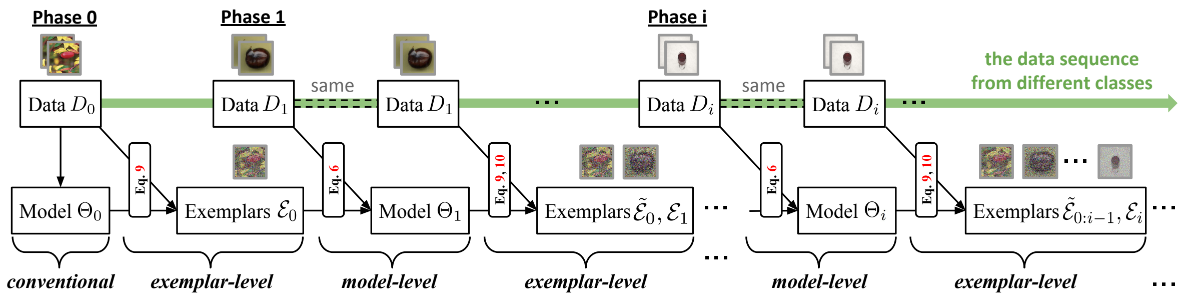


Figure 3.2: The computing flow of the proposed *mnemonics* training. It is a global BOP that alternates the learning of *mnemonics* exemplars (we call *exemplar-level* optimization) and CIL models (*model-level* optimization). The *exemplar-level* optimization within each phase is detailed in Figure 3.3. $\tilde{\mathcal{E}}$ denotes the old exemplars adjusted to the current phase.

Then, Θ_i will be used to train the parameters of the *mnemonics* exemplars, i.e., to solve the *exemplar-level* problem in Section 3.3.5.

3.3.5 Exemplar-level problem

Typically, the number of exemplars $\mathcal{E}_i$ is set to be greatly smaller than that of the original data D_i. Existing methods [RKSL17, HPL+19, WCW+19, CMG+18] are always based on the assumption that the models trained on the few exemplars also minimize its loss on the original data. However, there is no guarantee particularly when these exemplars are heuristically chosen. In contrast, our approach explicitly aims to ensure a feasible approximation of that assumption, thanks to the differentiability of our *mnemonics* exemplars.

To achieve this, we train a temporary model Θ'_i on $\mathcal{E}_i$ to maximize the prediction on D_i, for which we use D_i to compute a validation loss to penalize this temporary training with respect to the parameters of $\mathcal{E}_i$. The entire problem is thus formulated in a local BOP, where "local" means within a single phase, as

$$\min_{\mathcal{E}_i} \mathcal{L}_c\big(\Theta'_i(\mathcal{E}_i); D_i\big) \tag{3.7a}$$

$$\text{s.t. } \Theta'_i(\mathcal{E}_i) = \arg\min_{\Theta_i} \mathcal{L}_c(\Theta_i; \mathcal{E}_i). \tag{3.7b}$$

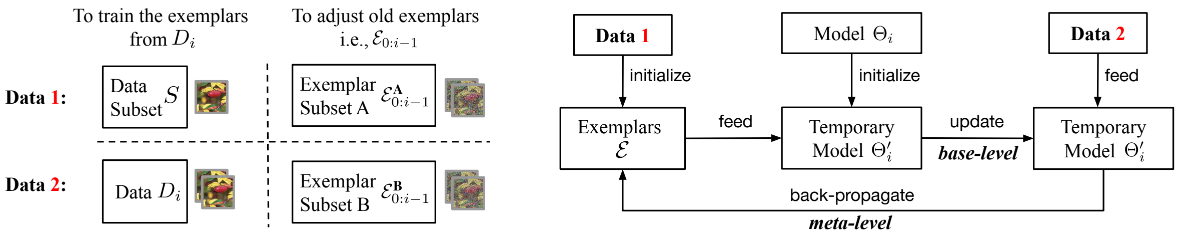
We name the temporary training in Problem 3.7b as *base-level* optimization and the validation in Problem 3.7a as *meta-level* optimization, similar to the naming in meta-learning applied to tackling few-shot tasks [FAL17].

Training $\mathcal{E}_i$. The training flow is detailed in Figure 3.3(b) with the data split on the left of Figure 3.3(a). First, the image-size parameters of $\mathcal{E}_i$ are initialized by a random sample subset S of D_i. Second, we initialize a temporary model Θ'_i using Θ_i and train Θ'_i on $\mathcal{E}_i$ (denoted uniformly as $\mathcal{E}$ in 3.3(b)), for a few iterations by gradient descent:

$$\Theta'_i \leftarrow \Theta'_i - \alpha_2 \nabla_{\Theta'} \mathcal{L}_c(\Theta'_i; \mathcal{E}_i), \tag{3.8}$$

where α_2 is the learning rate of fine-tuning temporary models. Finally, as the Θ'_i and $\mathcal{E}_i$ are both differentiable, we are able to compute the loss of Θ'_i on D_i, and back-propagate this validation loss to optimize $\mathcal{E}_i$,

$$\mathcal{E}_i \leftarrow \mathcal{E}_i - \beta_1 \nabla_{\mathcal{E}} \mathcal{L}_c\big(\Theta'_i(\mathcal{E}_i); D_i\big), \tag{3.9}$$


To train the exemplars from D_i
To adjust old exemplars i.e., $\mathcal{E}_{0:i-1}$
Data 1:
Data Subset S
Exemplar Subset A $\mathcal{E}^{\mathbf{A}}_{0:i-1}$
Data 2:
Data D_i
Exemplar Subset B $\mathcal{E}^{\mathbf{B}}_{0:i-1}$
Data 1
Model Θ_i
Data 2
initialize
initialize
feed
Exemplars $\mathcal{E}$
feed
Temporary Model Θ'_i
update
base-level
Temporary Model Θ'_i
back-propagate
meta-level

(a) data splits in two cases (b) uniform computing flow

Figure 3.3: The proposed local BOP framework that uses a uniform computing flow in (b) to handle two cases of *exemplar-level* learning: training new class exemplars $\mathcal{E}_i$ from D_i; and adjusting old exemplars $\mathcal{E}_{0:i-1}$, with the data respectively given in (a). Note that (1) $\mathcal{E}^{\mathbf{A}}_{0:i-1}$ and $\mathcal{E}^{\mathbf{B}}_{0:i-1}$ are used as the validation set alternately for each other when adjusting $\mathcal{E}_{0:i-1}$; (2) $\mathcal{E}$ in (b) denote the *mnemonics* exemplars which are $\mathcal{E}_i$, $\mathcal{E}^{\mathbf{A}}_{0:i-1}$, and $\mathcal{E}^{\mathbf{B}}_{0:i-1}$ in Eq. 3.9, 3.10a and 3.10b, respectively.

where β_1 is the learning rate. In this step, we basically need to back-propagate the validation gradients till the input layer, through unrolling all training gradients of Θ'_i. This operation involves a gradient through a gradient. Computationally, it requires an additional backward pass through $\mathcal{L}_c(\Theta'_i;\mathcal{E}_i)$ to compute Hessian-vector products, which is supported by standard numerical computation libraries such as TensorFlow [AAB+16] and PyTorch [SDC+19].

Adjusting $\mathcal{E}_{0:i-1}$. The *mnemonics* exemplars of a previous class were trained when this class occurred. It is desirable to adjust them to the changing data distribution online. However, old class data $D_{0:i-1}$ are not accessible, so it is not feasible to directly apply Eq. 3.9. Instead, we propose to split $\mathcal{E}_{0:i-1}$ into two subsets and subject to $\mathcal{E}_{0:i-1} = \mathcal{E}^{\mathbf{A}}_{0:i-1} \cup \mathcal{E}^{\mathbf{B}}_{0:i-1}$. We use one of them, e.g. $\mathcal{E}^{\mathbf{B}}_{0:i-1}$, as the validation set (i.e., a replacement of $D_{0:i-1}$) to optimize the other one, e.g., $\mathcal{E}^{\mathbf{A}}_{0:i-1}$, as shown on the right of Figure 3.3(a). Alternating the input and target data in Figure 3.3(b), we adjust all old exemplars in two steps:

$$\mathcal{E}^{\mathbf{A}}_{0:i-1} \leftarrow \mathcal{E}^{\mathbf{A}}_{0:i-1} - \beta_2 \nabla_{\mathcal{E}^{\mathbf{A}}} \mathcal{L}_c\big(\Theta'_i(\mathcal{E}^{\mathbf{A}}_{0:i-1}); \mathcal{E}^{\mathbf{B}}_{0:i-1}\big), \tag{3.10a}$$

$$\mathcal{E}^{\mathbf{B}}_{0:i-1} \leftarrow \mathcal{E}^{\mathbf{B}}_{0:i-1} - \beta_2 \nabla_{\mathcal{E}^{\mathbf{B}}} \mathcal{L}_c\big(\Theta'_i(\mathcal{E}^{\mathbf{B}}_{0:i-1}); \mathcal{E}^{\mathbf{A}}_{0:i-1}\big), \tag{3.10b}$$

where β_2 is the learning rate. $\Theta'_i(\mathcal{E}^{\mathbf{B}}_{0:i-1})$ and $\Theta'_i(\mathcal{E}^{\mathbf{A}}_{0:i-1})$ are trained by replacing $\mathcal{E}_i$ in Eq. 3.8 with $\mathcal{E}^{\mathbf{B}}_{0:i-1}$ and $\mathcal{E}^{\mathbf{A}}_{0:i-1}$, respectively. We denote the adjusted exemplars as $\tilde{\mathcal{E}}_{0:i-1}$. Note that we can also split $\mathcal{E}_{0:i-1}$ into more than 2 subsets, and optimize each subset using its complement as the validation data, following the same strategy in Eq. 3.10.

Fine-tuning models on only exemplars. The model Θ_i has been trained on $D_i \cup \mathcal{E}_{0:i-1}$, and may suffer from the classification bias caused by the imbalanced sample numbers, e.g., 1000 *versus* 20, between the classes in D_i and $\mathcal{E}_{0:i-1}$. To alleviate this bias, we propose to fine-tune Θ_i on $\mathcal{E}_i \cup \tilde{\mathcal{E}}_{0:i-1}$ in which each class has exactly the same number of samples (exemplars).

3.4 WEIGHT TRANSFER OPERATIONS

We deploy weight transfer operations [SLCS19, PSdV+18] to train the weight scaling and shifting parameters (named $\mathbf{T}_i$) in the i-th phase which specifically transfer the network weights Θ_{i-1} to Θ_i. The aim is to preserve the structural knowledge of Θ_{i-1} when learning Θ_i on new class data.

Specifically, we assume the q-th layer of Θ_{i-1} contains R neurons, so we have R neuron weights and biases as $\{W_{q,r}, b_{q,r}\}_{r=1}^{R}$. For conciseness, we denote them as W_q and b_q. For W_q, we learn R scaling parameters denoted as $\mathcal{T}_q^W$, and for b_q, we learn R shifting parameters denoted as $\mathcal{T}_q^b$. Let X_{q-1} and X_q be the input and output (feature maps) of the q-th layer. We apply $\mathcal{T}_q^W$ and $\mathcal{T}_q^b$ to W_q and b_q as,

$$X_q = (W_q \odot \mathcal{T}_q^W)X_{q-1} + (b_q + \mathcal{T}_q^b), \tag{3.11}$$

where $\odot$ donates the element-wise multiplication. Assuming there are Q layers in total, the scaling and shifting parameters are denoted as $\mathbf{T}_i = \{\mathcal{T}_q^W, \mathcal{T}_q^b\}_{q=1}^{Q}$.

Therefore, to learn the CIL model Θ_i, we use the indirect way of training $\mathbf{T}_i$ (instead of the direct way of training Θ_i) on $D_i \cup \mathcal{E}_{0:i-1}$ and keeping Θ_{i-1} fixed. During the training, both classification loss and distillation loss [LH18, RKSL17] are used. Let $\odot_L$ donate the function of applying $\mathbf{T}_i$ to Θ_{i-1} by layers (Eq. 3.12). The objective function Eq. 3.5 can be rewritten as:

$$\begin{aligned}\mathcal{L}_{\text{all}} =& \lambda\mathcal{L}_c(\mathbf{T}_i \odot_L \Theta_{i-1}; \mathcal{E}_{0:i-1} \cup D_i) \\ &+ (1-\lambda)\mathcal{L}_d(\mathbf{T}_i \odot_L \Theta_{i-1}; \Theta_{i-1}; \mathcal{E}_{0:i-1} \cup D_i),\end{aligned} \tag{3.12}$$

where λ is a scalar manually set to balance two loss terms. Let α_1 be the learning rate, $\mathbf{T}_i$ is updated with gradient descent as follows,

$$\mathbf{T}_i \leftarrow \mathbf{T}_i - \alpha_1 \nabla_{\mathbf{T}} \mathcal{L}_{\text{all}}. \tag{3.13}$$

After the learning of $\mathbf{T}_i$, we compute Θ_i as follows:

$$\Theta_i \leftarrow \mathbf{T}_i \odot_L \Theta_{i-1}. \tag{3.14}$$

3.5 ALGORITHM

In Algorithm 1, we summarize the overall process of the proposed *mnemonics* training. Steps 1-16 show the alternative learning of classification models and *mnemonics* exemplars. Specifically in each phase, Step 8 executes the *model-level* training, while Step 11 and 14 are the *exemplar-level*. Step 17 is optional due to different CIL settings regarding the memory budget. We conduct experiments in two settings: (1) each class has a fixed number (e.g., 20) of exemplars, and (2) the system consistently keeps a fixed memory budget in all phases, therefore, the system in earlier phases can store more exemplars per class and needs to discard old exemplars in later phases gradually. Step 18 fine-tunes the model on adjusted and balanced examples. It is helpful to reduce the previous model bias (Step 8) caused by the imbalance of sample numbers between new class data D_i and old exemplars $\mathcal{E}_{0:i-1}$. Step 19 is to evaluate the learned model Θ_i in the current phase, and the average over all phases will be reported as the final evaluation. Step 20 updates the memory to include new exemplars.

Algorithm 1: Mnemonics Training

```
Input: Data flow $\{D_i\}_{i=0}^N$.
Output: MCIL models $\{\Theta_i\}_{i=0}^N$, and mnemonics exemplars $\{\mathcal{E}_i\}_{i=0}^N$.
1  for $i$ in $\{0, 1, ..., N\}$ do
2      Get $D_i$;
3      if $i = 0$ then
4          Randomly initialize $\Theta_0$ and train it on $D_0$;
5      else
6          Get $\mathcal{E}_{0:i-1}$ from memory;
7          Initialize $\Theta_i$ with $\Theta_{i-1}$;
8          Train $\Theta_i$ on $\mathcal{E}_{0:i-1} \cup D_i$ by Eq. 3.6;
9      end
10     Sample $S$ from $D_i$ to initialize $\mathcal{E}_i$;
11     Train $\mathcal{E}_i$ using $\Theta_i$ by Eq. 3.9;
12     while $i \geq 1$ do
13         Split $\mathcal{E}_{0:i-1}$ into subsets $\mathcal{E}_{0:i-1}^{\mathbf{A}}$ and $\mathcal{E}_{0:i-1}^{\mathbf{B}}$ ;
14         Optimize $\mathcal{E}_{0:i-1}^{\mathbf{A}}$ and $\mathcal{E}_{0:i-1}^{\mathbf{B}}$ by Eq. 3.10;
15         Get the adjusted old exemplars $\tilde{\mathcal{E}}_{0:i-1}$
16     end
17     (Optional) delete part of the exemplars in $\tilde{\mathcal{E}}_{0:i-1}$;
18     Finetune $\Theta_i$ on $\mathcal{E}_i \cup \tilde{\mathcal{E}}_{0:i-1}$;
19     Run test and record the results;
20     Update $\mathcal{E}_{0:i} \leftarrow \mathcal{E}_i \cup \tilde{\mathcal{E}}_{0:i-1}$ in memory.
21 end
```

3.6 EXPERIMENTS

We evaluate the proposed *mnemonics* training approach on two popular datasets (CIFAR-100 [KH+09] and ImageNet [RDS+15]) for four different baseline architectures [LH18, RKSL17, WCW+19, HPL+19], and achieve consistent improvements. Below we describe the datasets and implementation details (Section 3.6.1), followed by results and analyses (Section 3.6.2), including comparisons to the state-of-the-art, ablation studies, and visualization results.

3.6.1 Datasets and implementation details

Datasets. We conduct CIL experiments on two datasets, CIFAR-100 [KH+09] and ImageNet [RDS+15], which are widely used in related works [RKSL17, CMG+18, WCW+19, HPL+19]. **CIFAR-100** [KH+09] contains 60,000 samples of 32×32 color images from 100 classes. Each class has 500 training and 100 test samples. **ImageNet** (ILSVRC 2012) [RDS+15] contains around 1.3 million samples of 224×224 color images from 1,000 classes. Each class has about 1,300 training and 50 test samples. ImageNet is typically used in two CIL settings [HPL+19, RKSL17]: one based on only a subset of 100 classes and the other based on the entire 1,000 classes. The 100-class data in **ImageNet-Subeset** are randomly sampled from ImageNet with an identical random seed (1993) by NumPy, following [RKSL17, HPL+19].

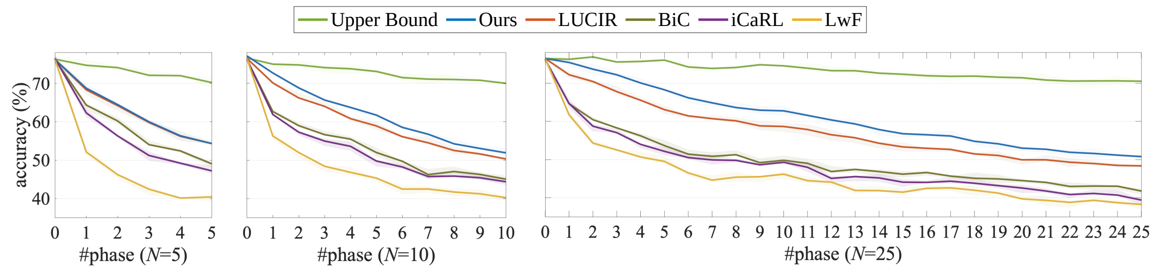
The architectures of Θ. Following the uniform setting [RKSL17, WCW+19, HPL+19], we use a 32-layer ResNet [HZRS16] for CIFAR-100 and an 18-layer ResNet for ImageNet. We deploy the weight transfer operations [SLCS19, PSdV+18] to train the network, rather than using standard weight overwriting. This helps to reduce *forgetting* between adjacent models (i.e., Θ_{i-1} and Θ_i).

The architecture of $\mathcal{E}$. It depends on the size of the image and the number of exemplars we need. On the CIFAR-100, each *mnemonics* exemplar is a $32 \times 32 \times 3$ tensor. On the ImageNet, it is a $224 \times 224 \times 3$ tensor. The number of exemplars is set in two manners [HPL+19]. (1) 20 samples are uniformly used for every class. Therefore, the parameter size of the exemplars per class is equal to tensor$\times$20. (2) The system keeps a fixed memory budget, e.g. at most 2,000 exemplars in total, in all phases. It thus saves more exemplars per class in earlier phases and discards old exemplars afterward. In both settings, we have the consistent finding that *mnemonics training* is the most efficient approach, surpassing the state-of-the-art by large margins with little computational or parametrization overheads.

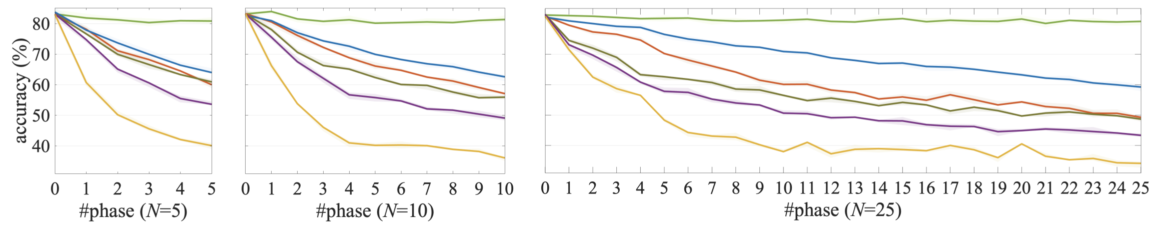
***Model-level* hyperparameters.** The SGD optimizer is used to train Θ. Momentum and weight decay parameters are set to 0.9 and 0.0005, respectively. In each (i.e. i-th) phase, the learning rate α_1 is initialized as 0.1. On the CIFAR-100 (ImageNet), Θ_i is trained in 160 (90) epochs for which α_1 is reduced to its $\frac{1}{10}$ after 80 (30) and then 120 (60) epochs. In Eq. 3.5, the scalar λ and temperature τ are set to 0.5 and 2, respectively, following [RKSL17, HPL+19].

***Exemplar-level* hyperparameters.** An SGD optimizer is used to update *mnemonics* exemplars $\mathcal{E}_i$ and adjust $\mathcal{E}_{0:i-1}$ (as in Eq. 3.9 and Eq. 3.10 respectively) in 50 epochs. In each phase, the learning rates β_1 and β_2 are initialized as 0.01 uniformly and reduced to their half after every 10 epochs. Gradient descent is applied to update the temporary model Θ' in 50 epochs (as in Eq. 3.8). The learning rate α_2 is set to 0.01. We deploy the same set of hyperparameters for fine-tuning Θ_i on $\mathcal{E}_i \cup \tilde{\mathcal{E}}_{0:i-1}$.

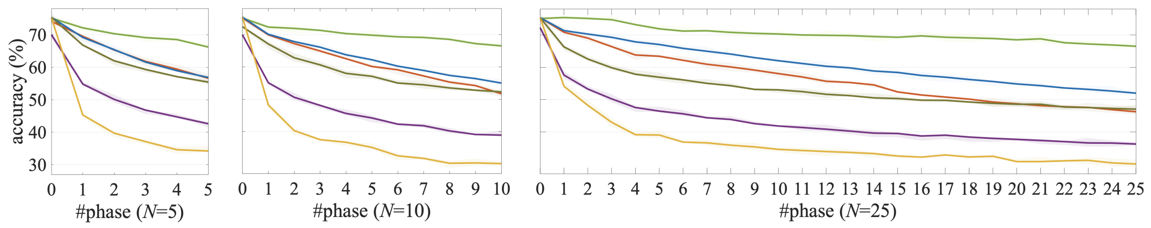
Benchmark protocol. This work follows the protocol in the most recent work — LU-



(a) CIFAR-100 (100 classes). In the 0-th phase, Θ_0 is trained on 50 classes, the remaining classes are given evenly in the subsequent phases.



(b) ImageNet-Subset (100 classes). In the 0-th phase, Θ_0 is trained on 50 classes, the remaining classes are given evenly in the subsequent phases.



(c) ImageNet (1000 classes). In the 0-th phase, Θ_0 on is trained on 500 classes, the remaining classes are given evenly in the subsequent phases.

Figure 3.4: Phase-wise accuracies (%). Light-color ribbons are visualized to show the 95% confidence intervals. Comparing methods: Upper Bound (the results of joint training with all previous data accessible in each phase); LUCIR (2019) [HPL+19]; BiC (2019) [WCW+19]; iCaRL (2017) [RKSL17]; and LwF (2016) [LH18]. We show Ours results using "LUCIR *w/* ours". Please refer to the average accuracy of each curve in Table 3.1.

Metric	Method	*CIFAR-100*			*ImageNet-Subset*			*ImageNet*		
		N=5	10	25	5	10	25	5	10	25
	LwF$^\diamond$ (2016) [LH18]	49.59	46.98	45.51	53.62	47.64	44.32	44.35	38.90	36.87
	LwF *w/* ours	54.21	52.72	51.59	60.94	59.25	59.71	52.70	50.37	50.79
	iCaRL (2017) [RKSL17]	57.12	52.66	48.22	65.44	59.88	52.97	51.50	46.89	43.14
Average acc. (%) $\uparrow$	iCaRL *w/* ours	60.00	57.37	54.13	72.34	70.50	67.12	60.61	58.62	53.46
$\bar{\mathcal{A}} = \frac{1}{N+1}\sum_{i=0}^{N}\mathcal{A}_i$	BiC (2019) [WCW+19]	59.36	54.20	50.00	70.07	64.96	57.73	62.65	58.72	53.47
	BiC *w/* ours	60.67	58.11	55.51	71.92	70.73	69.22	**64.63**	62.71	60.20
	LUCIR (2019) [HPL+19]	63.17	60.14	57.54	70.84	68.32	61.44	64.45	61.57	56.56
	LUCIR *w/* ours	**63.34**	**62.28**	**60.96**	**72.58**	**71.37**	**69.74**	64.54	**63.01**	**61.00**
	LwF$^\diamond$ (2016) [LH18]	43.36	43.58	41.66	55.32	57.00	55.12	48.70	47.94	49.84
	LwF *w/* ours	40.00	36.50	34.26	41.07	39.76	39.99	37.46	38.42	37.95
	iCaRL (2017) [RKSL17]	31.88	34.10	36.48	43.40	45.84	47.60	26.03	33.76	38.80
Forgetting rate (%) $\downarrow$	iCaRL *w/* ours	25.94	26.92	28.92	20.96	24.12	29.32	20.26	24.04	17.49
$\mathcal{F} = \mathcal{A}_N^Z - \mathcal{A}_0^Z$	BiC (2019) [WCW+19]	31.42	32.50	34.60	27.04	31.04	37.88	25.06	28.34	33.17
	BiC *w/* ours	22.42	24.50	25.52	18.43	19.20	21.43	18.32	19.72	20.50
	LUCIR (2019) [HPL+19]	18.70	21.34	26.46	31.88	33.48	35.40	24.08	27.29	30.30
	LUCIR *w/* ours	**10.91**	**13.38**	**19.80**	**17.40**	**17.08**	**20.83**	**13.85**	**15.82**	**19.17**

$^\diamond$ Using *herding* exemplars as [HPL+19, RKSL17, WCW+19] for fair comparison.

Table 3.1: Average accuracies $\bar{\mathcal{A}}$ (%) and forgetting rates $\mathcal{F}$ (%) for the state-of-the-art [HPL+19] and other baseline architectures [LH18, RKSL17, WCW+19] with and without our *mnemonics* training approach as a plug-in module. Let D_i^{test} be the test data corresponding to D_i in the i-th phase. $\mathcal{A}_i$ denotes the average accuracy of $D_{0:i}^{\text{test}}$ by Θ_i. $\mathcal{A}_i^Z$ is the average accuracy of D_0^{test} by Θ_i in the i-th phase. Note that the weight transfer operations are applied in "*w/* ours" methods.

Exemplar		*CIFAR-100*			*ImagNet-Subset*		
		N=5	10	25	5	10	25
random w/o adj.		61.87	60.23	58.57	70.67	69.15	67.17
random		62.64	60.61	58.82	70.69	69.67	67.46
herding w/o adj.	↑	62.98	61.23	60.36	71.66	71.02	69.40
herding		62.96	61.76	60.38	71.76	71.04	69.61
ours *w/o* adj.		63.25	61.86	60.46	71.91	71.08	69.68
ours		**63.34**	**62.28**	**60.96**	**72.58**	**71.37**	**69.74**
random w/o adj.		11.16	13.42	**12.26**	18.92	19.56	23.64
random		12.13	14.80	**12.26**	17.92	17.91	23.60
herding w/o adj.	↓	12.69	13.63	16.36	17.16	18.00	**20.00**
herding		11.00	14.38	15.60	**15.80**	17.84	20.72
ours *w/o* adj.		**9.80**	13.44	16.68	18.27	18.08	20.96
ours		10.91	**13.38**	15.22	17.40	**17.08**	20.83

Table 3.2: Ablation study. The top and the bottom blocks present average accuracies $\bar{\mathcal{A}}$ (%) and forgetting rates $\mathcal{F}$ (%), respectively. "*w/o* adj." means without old exemplar adjustment. Note that the weight transfer operations are applied in all these experiments.

CIR [HPL+19]. We also implement all other methods [RKSL17, CMG+18, WCW+19] on this protocol for a fair comparison. Given a dataset, the model (Θ_0) is firstly trained on half of the classes. Then, the model (Θ_i) learns the remaining classes evenly in the subsequent phases. Assume an CIL system has 1 initial phase and N incremental phases. The total number of incremental phases N is set to be 5, 10, or 25 (for each the setting is called "N-phase" setting). At the end of each individual phase, the learned Θ_i is evaluated on the test data $D_{0:i}^{\text{test}}$ where "$0 : i$" denote all seen classes so far. The average accuracy $\bar{\mathcal{A}}$ (over all phases) is reported as the final evaluation [RKSL17, HPL+19]. In addition, we propose a forgetting rate, denoted as $\mathcal{F}$, by calculating the difference between the accuracies of Θ_0 and Θ_N on the same initial test data D_0^{test}. The lower forgetting rate is better.

3.6.2 Results and analyses

Table 3.1 shows the comparisons with the state-of-the-art [HPL+19] and other baseline architectures [LH18, RKSL17, WCW+19], with and without our *mnemonics* training as a plug-in module. Note that "without" in [LH18, RKSL17, WCW+19, HPL+19] means using *herding* exemplars (we add *herding* exemplars to [LH18] for fair comparison). Figure 3.4 shows the phase-wise results of our best model, i.e., LUCIR [HPL+19] *w/* ours, and those of the baselines. Table 3.2 demonstrates the ablation study for evaluating two key components: training *mnemonics* exemplars; and adjusting old *mnemonics* exemplars. Figure 3.5 visualizes the differences between *herding* and *mnemonics* exemplars in the data space.

Compared to the state-of-the-art. Table 3.1 shows that taking our *mnemonics* training as a plug-in module on the state-of-the-art [HPL+19] and other baseline architectures consistently improves their performance. In particular, LUCIR [HPL+19] *w/* ours achieves the highest average accuracy and lowest forgetting rate, e.g. respectively 61.00% and 19.17% on the most challenging 25-phase ImageNet. The overview on forgetting rates $\mathcal{F}$ reveals that our approach is greatly helpful in reducing forgetting problems for every method. For example, LUCIR (*w/* ours) sees its $\mathcal{F}$ reduced to around the third and the half on the 25-phase

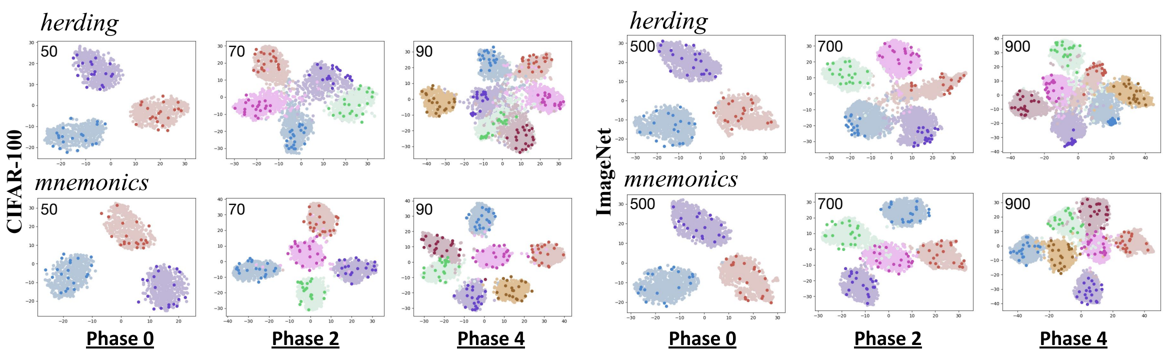


Figure 3.5: The t-SNE [MH08] results of *herding* and our *mnemonics* exemplars on two datasets. N=5. In each colored class, deep-color points are exemplars, and light-color ones are original data referring to the real data distribution. The total number of classes (used in the training) is given in the top-left corner of each sub-figure. For clear visualization, Phase-0 randomly picks 3 classes from 50 (500) classes on CIFAR-100 (ImageNet). Phase-2 and Phase-4 increases to 5 and 7 classes, respectively.

CIFAR-100 and ImageNet, respectively.

Different total phases (N = 5, 10, 25). Table 3.1 and Figure 3.4 demonstrate that the boost by our *mnemonics* training becomes larger in more-phase settings, e.g. on ImageNet-Subset, LUCIR *w/* ours gains 1.74% on 5-phase while 8.30% on 25-phase. When checking the ending points of the curves from N=5 to N=25 in Figure 3.4, we find related methods, LUCIR, BiC, iCaRL, and LwF, all suffer from performance drop. The possible reason is that their models get more and more seriously overfitted to *herding* exemplars which are heuristically chosen and fixed. In contrast, our best model (LUCIR *w/* ours) does not have such problem, thanks to our *mnemonics* exemplars being given both *strong optimizability* and *flexible adaptation ability* through the BOP.

Ablation study. Table 3.2 concerns six ablative settings and compares the efficiencies between our *mnemonics* training approach (*w/* and *w/o* adjusting old exemplars) and two baselines: *random* and *herding* exemplars. Concretely, our approach achieves the highest average accuracies and the lowest forgetting rates in all settings. Dynamically adjusting old exemplars brings consistent improvements, i.e., average 0.34% on both datasets. In terms of forgetting rates, our results are the lowest (best). It is interesting that *random* achieves lower (better) performance than *herding*. *Random* selects exemplars both on the center and boundary of the data space (for each class), but *herding* considers the center data only which strongly relies on the data distribution in the current phase but can not take any risk of distribution change in subsequent phases. This weakness is further revealed through the visualization of exemplars in the data space, e.g., in Figure 3.5.

Visualization results. Figure 3.5 demonstrates the t-SNE results for *herding* (deep-colored) and our *mnemonics* exemplars (deep-colored) in the data space (light-colored). We have two main observations. (1) Our *mnemonics* approach results in much clearer separation in the data than *herding*. (2) Our *mnemonics* exemplars are optimized to mostly locate on the boundaries between classes, which is essential to yielding high-quality classifiers. Comparing the Phase-4 results of two datasets (i.e., among the sub-figures on the rightmost column), we can see that learning more classes (i.e., on the ImageNet) clearly causes more confusion among classes in the data space, while our approach is able to yield stronger intra-class compactness and inter-class separation.

3.7 conclusion

In this chapter, we develop a novel *mnemonics training* framework for tackling multi-class incremental learning tasks. Our main contribution is the *mnemonics* exemplars which are not only efficient data samples but also flexible, optimizable and adaptable parameters contributing a lot to the flexibility of online systems. Quite intriguingly, our *mnemonics training* approach is *generic* that it can be easily applied to existing methods to achieve large-margin improvements. Extensive experimental results on four different baseline architectures validate the high efficiency of our approach, and the in-depth visualization reveals the essential reason is that our *mnemonics* exemplars are automatically learned to be the optimal replacement of the original data which can yield high-quality classification models.

4 LEARNING TO ALLOCATE MEMORY

Contents

Class-incremental learning (CIL) [RKSL17] trains classifiers under a strict memory budget: in each incremental phase, learning is done for new data, most of which is abandoned to free space for the next phase. The preserved data are exemplars used for replaying. However, existing methods use a static and ad hoc strategy for memory allocation, which is often sub-optimal. In this chapter, we propose a dynamic memory management strategy that is optimized for the incremental phases and different object classes. We call our method reinforced memory management (RMM), leveraging reinforcement learning. RMM training is not naturally compatible with CIL as the past, and future data are strictly non-accessible during the incremental phases. We solve this by training the policy function of RMM on pseudo CIL tasks, e.g., the tasks built on the data of the 0-th phase, and then applying it to target tasks. RMM propagates two levels of actions: Level-1 determines how to split the memory between old and new classes, and Level-2 allocates memory for each specific class. In essence, it is an optimizable and general method for memory management that can be used in any replaying-based CIL method. For evaluation, we plug RMM into two top-performing baselines (LUCIR+AANets and POD+AANets [LSS21a]) and conduct experiments on three benchmarks (CIFAR-100, ImageNet-Subset, and ImageNet-Full). Our results show clear improvements, e.g., boosting POD+AANets by 3.6%, 4.4%, and 1.9% in the 25-Phase settings of the above benchmarks, respectively.

This chapter is based on [LSS21b]. As the first author, Yaoyao Liu conducted all experiments and was the main writer. This work has been integrated into a popular open-source class-incremental learning toolbox [ZWYZ21].

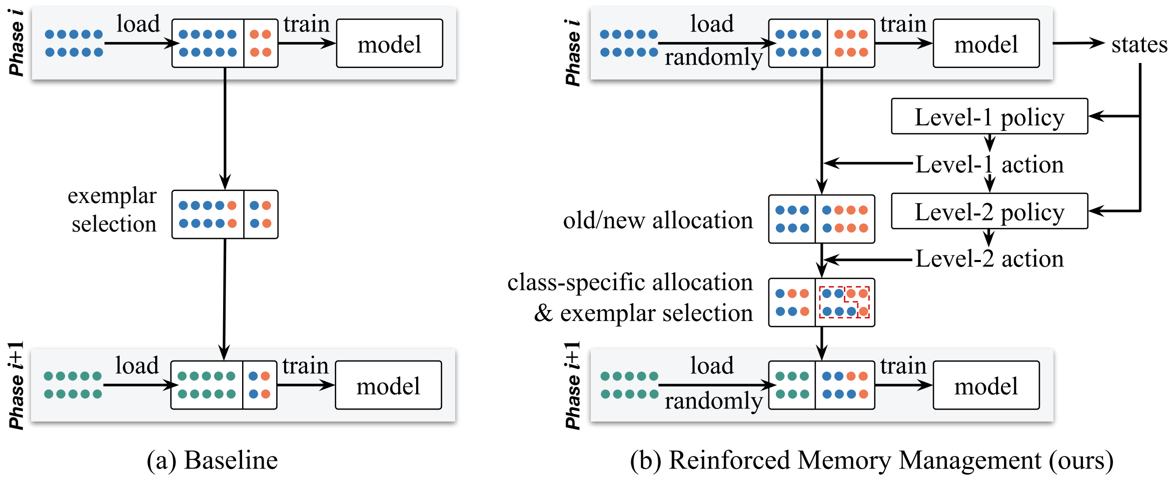


Figure 4.1: (a) Existing CIL methods [HPL+19, LSS21a, RKSL17] allocate memory between old and new classes in an arbitrary and frozen way, causing the data imbalance between old and new classes and exacerbating the catastrophic forgetting of old knowledge in the learned model. (b) Our proposed method—Reinforced Memory Management (RMM)—is able to learn the optimal and class-specific memory sizes in different incremental phases. Please note we use orange, blue, and green dots to denote the samples observed in the (i-1)-th, i-th, and (i+1)-th phases, respectively.

4.1 INTRODUCTION

Ideally, AI systems should be adaptive to ever-changing environments—where the data are continuously observed by sensors. Their models should be capable of learning new concepts from data while maintaining the ability to recognize previous ones. In practice, the systems often have constrained memory budgets because of which most of the historical data have to be abandoned [HTM+21]. However, deep-learning-based AI systems, when continuously updated using new data and limited historical data, often suffer from catastrophic forgetting, as the updates can override knowledge acquired from previous data [MC89, MH93, Rat90].

To encourage research on the forgetting problem, Rebuffi et al. [RKSL17] defined a standard protocol of class-incremental learning (CIL) for image classification, where the training data of different object classes come in phases. In each phase, the classifier is evaluated on all classes observed so far. As the total memory size is limited [RKSL17], CIL systems abandon the majority of the data and only preserve a small number of exemplars, e.g., 20 exemplars per class, which will be used for replaying in subsequent phases. Replaying usually happens for multiple epochs [DCO+20, HPL+19, LSS21a, RKSL17], so both the old class exemplars and new class data need to be stored in the limited memory. Existing CIL methods allocate memory between the old and new classes in an arbitrary and static fashion, e.g., 20 per old class *vs.* 1,300 per new class for the ImageNet-Full dataset. This causes a serious imbalance between the old and new classes and can exacerbate the problem of catastrophic forgetting.

To address this, we propose to learn an optimal memory management policy for each incremental phase with continuously reinforced model performance and call our method reinforced memory management (RMM). Detailed actions include 1) allocating the memory between the existing (old) and the coming (new) data for each phase, and 2) specifying the memory for each old class according to its recognition difficulty before abandoning any of its data. To this end, we leverage reinforcement learning [LCL19, LL19, LZQL19, Wil92, ZL17] and design a new policy function to contain two sub-functions that propagate two levels of actions in a hierarchical way. Level-1 function determines how to split memory between the old and new data. Its output action is then inputted into the Level-2 function to determine how to allocate memory for each old class. The overall objective of the function is to maximize the cumulative evaluation accuracy across all incremental phases. However, this is not naturally compatible with the standard protocol of CIL [RKSL17] where neither past nor future data are accessible for evaluation. To tackle this issue, we propose to pre-train the function on pseudo CIL tasks and then adopt it in the learning process of our target task. In principle, we can build such pseudo tasks using any available categorical data, e.g., the data in the 0-th phase of the target CIL task or the data from another dataset. Even though this is a non-stationary reinforcement learning problem, we can regard the pseudo and target CIL tasks as a sequence of stationary tasks and train the policy function to exploit the dependencies between these consecutive tasks. Such continuous adaptation in non-stationary environments is feasible based on the empirical analysis given in [ABB+18].

Technically, we propose the following method to guarantee the transferability of policy functions between pseudo and target CIL tasks. We take a Level-1 action based on the ratio of the number of new classes to the total number of classes observed so far. A lower (higher) ratio will result in weakening the stability (plasticity) of the classification model. Then, we take a Level-2 action for each individual class conditioned on both the Level-1 action and the training entropy of that class. A higher entropy denotes a more difficult class, leading to more memory allocated to the class. For evaluation, we conduct extensive CIL experiments by plugging RMM into two top-performing methods (LUCIR+AANets, POD+AANets) and testing them on three benchmarks (CIFAR-100, ImageNet-Subset, and ImageNet-Full). Our results show the clear and consistent superiority of RMM, e.g., it boosts the state-of-the-art POD+AANets by 3.6%, 4.4%, and 1.9% in the 25-Phase settings of the above benchmarks, respectively.

Our technical contribution is three-fold. 1) A hierarchical reinforcement learning algorithm called RMM to manage the memory in a way that can be conveniently modified through incremental phases and for different classes. 2) A pseudo task generation strategy that requires only in-domain available data (small-scale) or cross-domain datasets (large-scale), relieving the data incompatibility between reinforcement learning and class-incremental learning. 3) Extensive experiments, visualization, and interpretation for RMM in three CIL benchmarks and using two top models as baselines.

4.2 RELATED WORK

In this section, we discuss related works on reinforcement Learning problems. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Reinforcement Learning defines an agent that needs to decide its actions in an unknown environment by maximizing the expected cumulative reward. It has been widely applied to many optimization problems, e.g., neural architecture search [XZ18, ZL17] and neural machine translation [RCAZ16, SCH+16]. Reinforcement learning has also been introduced to solve incremental learning problems. Xu et al. [XZ18] proposed to increase convolution filters once a new task arrives and optimize the increased number by reinforcement learning. Gao et al. [GLK20] proposed an improved version that makes the minimal expansion of the network, reducing memory and computing overheads. Veniat et al. [VDR21] introduced a modular architecture, where each module represents a different atomic skill, and used the REINFORCE algorithm [Wil92] to optimize it. Huang et al. [HFLR19] combined reinforcement learning with Net2Net [CGS16] and designed a NAS-based CIL method. In our work, we also use the REINFORCE algorithm [Wil92], but **differ** in three aspects. First, we are the first to optimize memory allocation for CIL in a reinforced way. Second, we learn the policy functions on generated pseudo CIL tasks, where we can access both past, and future data (for each incremental phase) and thus are able to compute the cross-phase (long-term) rewards. In contrast, the related work [GLK20, XZ18] could use only current-phase data to estimate a short-term reward. Third, our reinforcement learning has a hierarchical structure that specially fits the nature of the data stream in the CIL settings.

4.3 METHODOLOGY

Our RMM approach learns policy functions that propagate two levels of actions in a hierarchical way, specially designed for CIL. As illustrated in Figure 4.1 (b), Level-1 determines the memory split between exemplars and new data, and Level-2 allocates the memory for each individual class.

In the following, we first elaborate on the denotations in Section 4.3.1. Then, we introduce the preliminaries for reinforcement learning in Section 4.3.2. After that, we motivate and introduce the formulation of RMM, including the definitions of states, actions, rewards, and hierarchical policy functions in Section 4.3.3. In Section 4.3.4, we detail the steps of creating pseudo CIL tasks on which we learn the policy functions. In Section 4.3.5, we summarize the algorithm.

4.3.1 Denotations for CIL

CIL usually assumes $(N+1)$ learning phases: an initial phase and N incremental phases during which the number of classes gradually increases till the maximum [DCO+20, HPL+19, HTM+21, LSL+20]. We assume that total memory $\mathcal{M}$ is bounded and fixed for all incremental phases [RKSL17]. $\mathcal{M}$ is used to store the exemplars and new coming data as both kinds of data need to be loaded repeatedly during training epochs. In the initial (0-th) phase, data $\mathcal{D}_0$, containing the training samples of $\mathcal{C}_0$ classes, are used to learn the initial classification model Θ_0. In the i-th incremental phase, we split $\mathcal{M}$ into two dynamic partitions: the exemplar memory $\mathcal{M}_{\text{old}}$ and new data memory $\mathcal{M}_{\text{new}}$. We select $\mathcal{E}_t$ as representative samples of the data seen in the t-th phase, and denote total exemplars $\mathcal{E}_0 \sim \mathcal{E}_{i-1}$ shortly as $\mathcal{E}_{0:i-1}$. We save $\mathcal{E}_{0:i-1}$ into $\mathcal{M}_{\text{old}}$ and free $\mathcal{M}_{\text{new}}$. Then, we observe new data that contain $\mathcal{C}_i$ new classes. We randomly load new data into $\mathcal{M}_{\text{new}}$ until $\mathcal{M}_{\text{new}}$ is full, and all the other new data are discarded. We denote the loaded new data as $\mathcal{D}_i$. Then, we initialize Θ_i with Θ_{i-1}, and train it using $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. The resulting model Θ_i will be evaluated with a test set containing all classes observed so far. We repeat this training and testing, and report the average accuracy across all phases.

4.3.2 Preliminaries for Reinforcement Learning

Reinforcement learning (RL) aims to learn an optimal policy function π for an agent interacting in an unknown environment [Wil92, XZ18, ZL17]. In the CIL scenario, in each incremental phase, the agent observes the current state s_i from the environment, and then takes an action a_i (how to allocate memory) according to the policy function $\pi(a_i|s_i)$. Subsequently, the environment is updated to a new state s_{i+1} and the reward r_i is calculated to optimize the parameters of $\pi(a_i|s_i)$ through back-propagation. Specifically, the learning objective of $\pi(a_i|s_i)$ is to maximize the expected cumulative reward $R_i = \sum_{t=i}^{\infty} \gamma^{t-i} r_t$, where $\gamma \in [0, 1)$ is a discounting factor that determines the weights of future rewards. Please note that in our case, the (N+1)-phase CIL task is a finite horizon problem [Gl000, ZL17], so we remove the discounting factor and use $R = \sum_{t=0}^{N} r_t$, which is actually the cumulative validation accuracy of all training CIL tasks. In Section 4.3, we discuss the proposed RL algorithm for memory allocation and how to generate pseudo tasks for training its policy function.

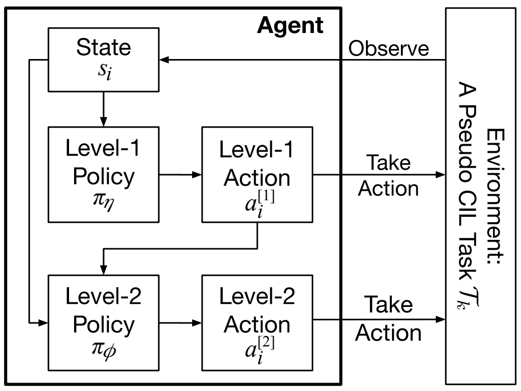
4.3.3 Formulation of RMM

In the i-th incremental phase CIL, we manage the memory for two kinds of data: exemplars $\mathcal{E}_{0:i-1}$ and new data $\mathcal{D}_i$. For the former, we have access to their images and labels so we can allocate a different memory size to a different class, e.g., based on its recognition difficulty. For the latter, we do not have such access before loading the data (otherwise, causing a violation to the CIL protocol), so we are only able to learn a total memory size, i.e., the memory size for all new classes (and then split it evenly for each individual class). Therefore, memory management in CIL settings is inherently hierarchical: 1) coarse memory allocation between exemplars and new data; and then 2) fine-grained memory allocation among specific classes. To this end, we modify the standard reinforcement learning into a hierarchical structure.

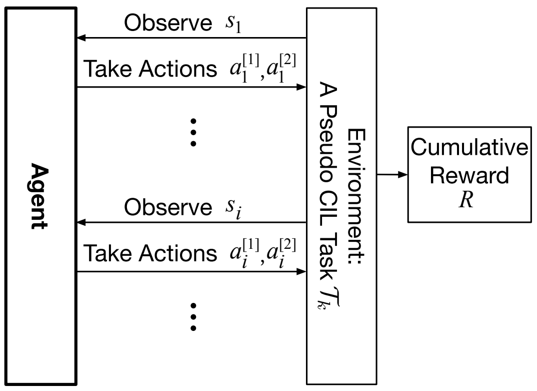
As illustrated in Figure 4.2 (a), in the i-th incremental phase of CIL (i.e., the environment), the argent receives a state value s_i. Level-1 policy π_η takes s_i as the input to produce an action $a_i^{[1]} \sim \pi_\eta(s_i)$. $a_i^{[1]}$ determines how to split memory between the exemplars and new data. After that, Level-2 policy π_ϕ takes s_i and $a_i^{[1]}$ as inputs to produce the second action $a_i^{[2]} \sim \pi_\phi(s_i, a_i^{[1]})$ that distributes the exemplar memory for each individual class.

States, defined for our CIL settings, should have two properties. 1) Being transferable between CIL tasks, e.g., from a small-scale CIL task including 50 classes (in total) to a large one including 100 classes. The reason is that we need to transfer the policy functions learned from pseudo CIL tasks (defined in Section 4.3.4) to the target task. The states, the inputs of policy functions, should also be transferable. 2) Being distinct in each incremental phase. This is to enable the state variable to represent a specific forgetting or data imbalance degree at each different learning phase of the CIL model. To fulfill these properties, we formulate the state in the i-th phase as $s_i = \left(\frac{\mathcal{C}_i}{\sum_{t=0}^{i-1} \mathcal{C}_t}, \frac{|\mathcal{M}_{\text{old}}|}{|\mathcal{M}|} \right)$, where $\mathcal{C}_i$ denotes the number of classes in $\mathcal{D}_i$, $\mathcal{M}_{\text{old}}$ denotes the memory allocated to exemplars $\mathcal{E}_{0:i-1}$, and $\mathcal{M}$ is the total memory.

Level-1 Actions. In the 1-st incremental phase, our Level-1 policy function produces an action to allocate the memory for exemplars $\mathcal{E}_0$ and new data $\mathcal{D}_1$. We denote this action as $a_1^{[1]}$ and assign its value with the ratio of the number of the exemplars $|\mathcal{E}_0|$ to the memory


Agent
State s_i
Level-1 Policy π_η
Level-1 Action $a_i^{[1]}$
Level-2 Policy π_ϕ
Level-2 Action $a_i^{[2]}$
Observe
Take Action
Take Action
Environment: A Pseudo CIL Task $\mathcal{T}_k$

(a) The i-th incremental phase of the k-th pseudo CIL task


Agent
Observe s_1
Take Actions $a_1^{[1]}, a_1^{[2]}$
Observe s_i
Take Actions $a_i^{[1]}, a_i^{[2]}$
Environment: A Pseudo CIL Task $\mathcal{T}_k$
Cumulative Reward R

(b) The k-th pseudo CIL task

Figure 4.2: (a) In the i-th phase of the k-th pseudo CIL task, Level-1 policy π_η takes s_i as the input, and produces action $a_i^{[1]}$. Level-2 policy π_ϕ takes s_i and $a_i^{[1]}$ as the inputs, then produces action $a_i^{[2]}$. (b) For the k-th pseudo CIL task, we allocate memory for N times (i.e., in N phases) using the policies π_η and π_ϕ, and compute the cumulative reward R.

size $|\mathcal{M}|$, so we have $a_1^{[1]} \in (0, 1)$. In the i-th phase ($i \geq 2$), the definition of $a_i^{[1]}$ is different to $a_1^{[1]}$ as it is a relative change over $a_{i-1}^{[1]}$. Specifically, $a_i^{[1]}$ is the ratio of increased (if its value is positive) or decreased (if negative) memory size of $\mathcal{M}_{\text{old}}$ compared to the (i-1)-th phase. Using this definition aims for smooth and continuous memory management. In the formulation, the memory sizes of exemplars $\mathcal{E}_{0:i-1}$ and new data $\mathcal{D}_i$ are, respectively,

$$|\mathcal{M}_{\text{old}}| = |\mathcal{E}_{0:i-1}| = \sum_{t=1}^{i} a_t^{[1]} |\mathcal{M}|, \quad |\mathcal{M}_{\text{new}}| = |\mathcal{D}_i| = \left(1 - \sum_{t=1}^{i} a_t^{[1]}\right) |\mathcal{M}|. \tag{4.1}$$

We set a constrain $a_i^{[1]} \in [-0.1, 0.1]$ for $i \geq 2$. Otherwise, if $a_i^{[1]}$ is too big, there are not enough exemplars to fill the memory, as most old-class data has been abandoned. If $a_i^{[1]}$ is too small, many exemplars will be permanently deleted in this phase, making it hard or even impossible to adjust $\mathcal{M}_{\text{old}}$ back to a high value in the future phases. If $\sum_{t=1}^{i} a_t^{[1]} > 1$, $\mathcal{M}_{\text{new}}$ will be negative. So, we force $\sum_{t=1}^{i} a_t^{[1]} \leq 1$ by rejection sampling [Bis06], i.e., using π_η to output another action until it is feasible to execute. Note that this situation rarely happens in real training, because when $\mathcal{M}_{\text{new}}$ becomes very low, π_η tends to produce an action to increase it.

Level-2 Actions. Here, we elaborate on how to get class-specific memory allocation. In the $(i-1)$-th phase, we split the classes for $\mathcal{D}_{i-1}$ into two groups evenly according to training entropy values: classes with higher values (difficult classes) are in one group and the rest in the other group. Therefore, Level-2 action $a_i^{[2]} \in (0, 1)$ determines how to split memories between harder and easier classes. During initial experiments, we observed that using two groups already yields improved results and using more groups causes a decrease.

Let $\mathcal{M}_j^A$ and $\mathcal{M}_j^B$ denote the memory allocated for the high-entropy and low-entropy groups, respectively, in the j-th phase ($j \le i$):

$$|\mathcal{M}_j^A| = a_{j+1}^{[2]}|\mathcal{E}_j| = \frac{a_{j+1}^{[2]}\mathcal{C}_j}{\sum_{t=1}^{i}\mathcal{C}_t}|\mathcal{M}_{\text{old}}|, \ \ |\mathcal{M}_j^B| = (1-a_{j+1}^{[2]})|\mathcal{E}_j| = \frac{(1-a_{j+1}^{[2]})\mathcal{C}_j}{\sum_{t=1}^{i}\mathcal{C}_t}|\mathcal{M}_{\text{old}}|. \quad (4.2)$$

Then, we allocate memory evenly to the classes within the group, e.g., if the high-entropy group has 10 classes, each class will have a memory size of $\frac{1}{10}|\mathcal{M}_j^A|$.

Rewards. The objective of CIL is that the trained model (in any phase) should be efficient to recognize all classes seen so far. It is intuitive and convenient to use the validation accuracy as the reward in each phase. In the i-th phase, the objective of RMM is to maximize the expected cumulative reward, i.e., $R = \sum_{i=0}^{N} r_i$, where r_i denotes the validation accuracy in the i-th phase.

4.3.4 Optimization

In the CIL protocol, it is impossible to see past or future data in any incremental phase. It is thus not intuitive how to compute cumulative rewards till the last phase. We propose to solve the issue by generating pseudo CIL tasks (where all data are accessible).

Pseudo CIL Tasks should meet two requirements: 1) their training and validation data are fully accessible for computing cumulative rewards, and 2) they have the same format (e.g., the same number of phases) of the target CIL task. ***Data Sources:*** For requirement 1, an intuitive solution is to use $\mathcal{D}_0$ (available in the 0-th phase). Based on the CIL protocol [DCO+20, HPL+19, HTM+21, LSS21a], $\mathcal{D}_0$ contains half of the classes of the whole dataset, e.g., 50 classes on CIFAR-100, which supplies enough data to build downsized CIL tasks. When building the tasks, we randomly choose 10% training samples of each class (from $\mathcal{D}_0$) to compose a pseudo validation set (note that we are not allowed to use the original validation set in training). When aiming for larger-scale data in CIL, we can leverage smaller datasets. For example, the pseudo tasks for ImageNet-Subset can be built on the data of CIFAR-100. This is also meaningful to evaluate the transferability of RMM policy functions (discussed in the Ablation Study). ***Task Generation Protocol*** is based on requirement 2. If using another dataset, we simply follow its original CIL protocol. If using the data accessed in the 0-th phase (i.e., $\mathcal{D}_0$), we can reduce the number of classes (in each phase) by half. For example, for CIFAR-100, we use 50-class $\mathcal{D}_0$ to generate a 5-phase pseudo CIL task as follows: loading 25 classes in the 0-th phase, and after that, five classes per phase. To generate another pseudo task, we simply change the order of classes.

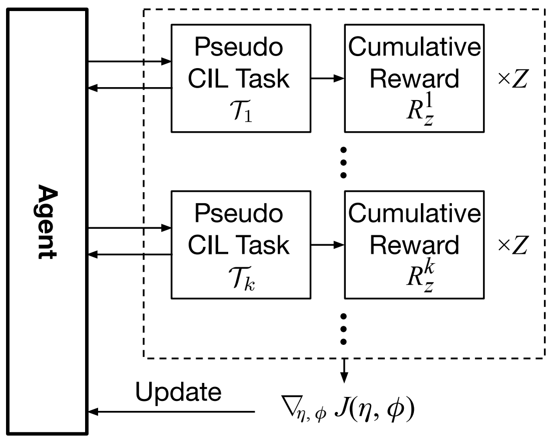

Agent
Pseudo CIL Task $\mathcal{T}_1$
Cumulative Reward R_z^1
$\times Z$
Pseudo CIL Task $\mathcal{T}_k$
Cumulative Reward R_z^k
$\times Z$
Update
$\nabla_{\eta,\phi} J(\eta,\phi)$

Figure 4.3: Updating η and ϕ in one epoch. To get stable gradients for $J(\eta, \phi)$, we create K different pseudo CIL tasks, and run each task for Z times.

Training. We elaborate the steps of learning Level-1 policy π_η and Level-2 policy π_ϕ in the following. The goal is to optimize the parameters η and ϕ by maximizing the expected cumulative reward $J(\eta, \phi)$. We denote any pseudo CIL task and its cumulative reward as $\mathcal{T}$ and R, respectively, and have,

$$J(\eta, \phi) = \mathbb{E}_{\mathcal{T}} \mathbb{E}_{\pi_\eta, \pi_\phi}[R]. \tag{4.3}$$

Policy Gradient Estimation. According to the policy gradient theorem [Wil92], we can compute the gradients for $J(\eta, \phi)$ as follows,

$$\nabla_{\eta,\phi} J(\eta, \phi) = \mathbb{E}_{\mathcal{T}} \left[\sum_{i=1}^{N} \mathbb{E}_{\pi_\eta, \pi_\phi} [\nabla_{\eta,\phi} \log(\pi_\eta(a_i^{[1]}|s_i)\pi_\phi(a_i^{[2]}|s_i, a_i^{[1]}))R] \right]. \tag{4.4}$$

Following the REINFORCE algorithm [Wil92], we replace the expectations $\mathbb{E}_{\mathcal{T}}[\cdot]$ and $\mathbb{E}_{\pi_\eta, \pi_\phi}[\cdot]$ with sample averages using the Monte Carlo method [Ham13]. Specifically, in each epoch, we create K pseudo tasks and run each task for Z times, as shown in Figure 4.3. Thus we can derive the empirical approximation of $\nabla_{\eta,\phi} J(\eta, \phi)$ as,

$$\nabla_{\eta,\phi} J(\eta, \phi) = \frac{1}{ZK} \sum_{k=1}^{K} \sum_{z=1}^{Z} \sum_{i=1}^{N} \nabla_{\eta,\phi} \log(\pi_\eta(a_i^{[1]}|s_i)\pi_\phi(a_i^{[2]}|s_i, a_i^{[1]}))(R_z^k - b), \tag{4.5}$$

where R_z^k denotes the z-th reward for the k-th pseudo task $\mathcal{T}_k$, and b denotes the baseline function—the moving average of previous rewards. Using this baseline function is a common trick in RL to reduce the variance of estimated policy gradients [KvHW19, RMM+17, ZL17].

Updating Parameters. We update η and ϕ in each epoch according to the gradient ascent rule [XZ18, ZL17]:

$$\eta := \eta + \beta_1 \nabla_\eta J(\eta, \phi), \quad \phi := \phi + \beta_2 \nabla_\phi J(\eta, \phi), \tag{4.6}$$

where β_1 and β_2 are the learning rates. We iterate this update for m epochs in total.

4.3.5 Algorithm

Algorithm 2 summarizes the overall training steps of the proposed RMM. There are four loops in the algorithm: 1) we train the RMM agent for m epochs; 2) we create K pseudo CIL tasks in each epoch; 3) we run each pseudo CIL task for Z times; and 4) there are N+1 learning phases each time. Specifically, Line 3 initializes the parameters of policy functions. Line 6 creates the k-th pseudo CIL task. Line 8 initializes the classification model. Lines 10-16 allocate the memory according to the actions produced by RMM policy. Line 17 loads new data. Lines 18-19 train the classification model and compute the accuracy. Line 20 estimates the z-th cumulative reward. Lines 21-22 compute the gradients and update policy functions.

Algorithm 2: Learning policy functions in RMM

```
Input: Data $\mathcal{D}$ for generating pseudo CIL tasks
Output: Policy functions $\pi_\eta$, $\pi_\phi$.
Initialize $\eta$ and $\phi$;
for $m$ epochs do
    for $k$ in $1, ..., K$ do
        Create a new pseudo task $\mathcal{T}_k$ using $\mathcal{D}$;
        for $z$ in $1, ..., Z$ do
            Initialize classification model $\Theta_0$;
            for $i$ in $0, ..., N$ do
                if $i \geq 1$ then
                    Observe $s_i$ and produce $a_i^{[1]} \sim \pi_\eta(s_i)$;
                    Allocate $\mathcal{M}_{\text{old}}$ and $\mathcal{M}_{\text{new}}$ using Eq. 4.1;
                    Produce $a_i^{[2]} \sim \pi_\phi(a_i^{[1]}, s_i)$;
                    Allocate $\{\mathcal{M}_j^A\}_{j=0}^i$ and $\{\mathcal{M}_j^B\}_{j=0}^i$ using Eq. 4.2;
                    Update $\mathcal{E}_{0:i-1}$ using herding [RKSL17];
                    Save $\mathcal{E}_{0:i-1}$ in $\mathcal{M}_{\text{old}}$ and free $\mathcal{M}_{\text{new}}$;
                end
                Observe new data and load $\mathcal{D}_i$ into $\mathcal{M}_{\text{new}}$ randomly;
                Initialize $\Theta_i$ with $\Theta_{i-1}$ and train it using $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$;
                Compute validation accuracy $r_i$;
            end
            Compute $R_z^k = \sum_{i=0}^N r_i$ and update $b$;
        end
    end
    Compute $\nabla_{\eta,\phi} J(\eta, \phi)$ using Eq. 4.5;
    Update $\eta$ and $\phi$ using Eq. 4.6.
end
```

4.4 EXPERIMENTS

We evaluate the proposed RMM method on three CIL benchmarks: CIFAR-100 [KH+09], ImageNet-Subset [RKSL17], and ImageNet-Full [RDS+15], and use two top-performing methods LUCIR+AANets and POD+AANets [LSS21a] as baselines. Below we introduce the datasets and implementation details (Section 4.4.1), followed by the experimental results and analyses (Section 4.4.2).

4.4.1 Datasets and Implementation Details

Datasets. We use three benchmarks based on two datasets, CIFAR-100 [KH+09] and ImageNet [RDS+15], following common settings [DCO+20, HPL+19, RKSL17, LSS21a]. CIFAR-100 [KH+09] contains $60,000$ samples of 32×32 color images from 100 classes. There are 500 training and 100 test samples for each class. ImageNet (ILSVRC 2012) [RDS+15] contains around 1.3 million samples of 224×224 color images from $1,000$ classes. There are about $1,300$ training and 50 test samples for each class. ImageNet has two CIL settings: ImageNet-Subset is based on a subset of 100 classes; and ImageNet-Full uses the full set of $1,000$ classes. The 100-class data for the ImageNet-Subset are sampled from ImageNet. For the experiments on PODNet [DCO+20] and POD-AANets [LSS21a], we use the same class orders and hyperparameters as [DCO+20]. For the experiments on LUCIR [HPL+19] and LUCIR-AANets [LSS21a], we use the same class orders and hyperparameters as [HPL+19].

Benchmarks. We follow the benchmark protocol used in [DCO+20, HPL+19, LSS21a, LSL+20]. Given a dataset, the initial (the 0-th phase) model is trained on the data of half of the classes. Then, it learns the remaining classes evenly in the subsequent N phases. Assume there is an initial phase and N incremental phases in the CIL system. The total number of incremental phases N is set to be 5, 10, or 25 (for each the setting is called "N-phase" setting). At the end of each individual phase, the learned model in each phase is evaluated on the test set containing all seen classes. In the tables, we report average accuracy over all phases and the last-phase accuracy, where the latter indicates the degree of forgetting.

Network Architectures. Following [HPL+19, LSS21a, RKSL17, WCW+19], we use a 32-layer ResNet [RKSL17] for CIFAR-100 and an 18-layer ResNet [HZRS16] for ImageNet. Please note that it is standard to use a shallower ResNet for ImageNet. The 32-layer ResNet consists of an initial convolution layer and three residual blocks (in a single branch). Each block has ten convolution layers with 3×3 kernels. The number of filters starts from 16 and is doubled every next block. After these three blocks, there is an average-pooling layer to compress the output feature maps to a feature embedding. The 18-layer ResNet follows the standard settings in [HZRS16]. We deploy AANets using the same parameters as its original paper [LSS21a]. For policy functions π_η and π_ϕ, we use two-layer FC networks. All actions are discretized at 0.1 intervals to reduce the search space and get a tolerable training overhead.

Hyperparameters and Configuration. The training of the classification model Θ exactly follows the uniform setting in [DCO+20, HPL+19, LSS21a, LSL+20]. On CIFAR-100 (ImageNet-Subset/Full), we train it for 160 (90) epochs in each phase, and divide the learning rate by 10 after 80 (30) and then after 120 (60) epochs. Then, we fine-tune the model for 20 epochs using only exemplars (including the preserved exemplars of the new data to be used in

Method	*CIFAR-100*			*ImageNet-Subset*			*ImageNet-Full*		
	N=5	10	25	5	10	25	5	10	25
LwF [LH18]	56.79	53.05	50.44	58.83	53.60	50.16	52.00	47.87	47.49
iCaRL [RKSL17]	60.48	56.04	52.07	67.33	62.42	57.04	50.57	48.27	49.44
LUCIR [HPL+19]	63.34	62.47	59.69	71.21	68.21	64.15	65.16	62.34	57.37
Mnemonics [LSL+20]	64.59	62.59	61.02	72.60	71.66	70.52	65.40	64.02	62.05
PODNet [DCO+20]	64.60	63.13	61.96	76.45	74.66	70.15	66.80	64.89	60.28
LUCIR-AANets [LSS21a]	66.88	65.53	63.92	72.80	69.71	68.07	65.31	62.99	61.21
w/ RMM (ours)	68.42	67.17	64.56	73.58	72.83	72.30	65.81	64.10	62.23
POD-AANets [LSS21a]	66.61	64.61	62.63	77.36	75.83	72.18	67.97	65.03	62.03
w/ RMM (ours)	**68.86**	**67.61**	**66.21**	**79.52**	**78.47**	**76.54**	**69.21**	**67.45**	**63.93**

Table 4.1: Average accuracies (%) across all phases using two state-of-the-art methods (LUCIR+AANets and POD+AANets [LSS21a]) *w/* and *w/o* our RMM plugged in. The upper block is for recent CIL methods. For fair comparison, we re-implement these methods using our strict memory budget (see "**Memory Budget**" in Section 4.4.1) based on the public code.

future phases). We use an SGD optimizer and an ADAM optimizer for the classification model and policy functions, respectively.

Memory Budget. There are two popular settings about memory budget in related work. One uses a bounded memory budget with a fixed capacity for all phases [HPL+19, LSL+20, RKSL17]. Another one allows the memory budget to grow along with phases [HPL+19, HTM+21, TCH+20]. The first one is more strict and thus used as the major setting in this chapter.In every benchmark, the total budget of memory depends on the phase number N. For example, on CIFAR-100, the total memory budget is set as $7,000$ samples when N=5 ($7,000$ samples = 10 classes/phase $\times$ 500 samples/class + $2,000$ samples). Please note that $2,000$ is a bounded memory budget allocated since the 0-th phase for saving exemplars. For a fair comparison, we re-implement related methods and report the results in Table 4.1 if their original results (in the respective papers) were obtained in a different setting of memory budget.

4.4.2 Results and Analyses

Table 4.1 presents the results of two state-of-the-art methods (LUCIR+AANets and POD+AANets [LSS21a]) *w/* and *w/o* our RMM plugged in, and some recent CIL work [DCO+20, HPL+19, LH18, LSL+20, RKSL17]. Table 4.2 shows the ablation study in 6 settings. Figure 4.4 plots the changes of the average number of exemplars per old/new class for the incremental phases.

Comparing to the State-of-the-Art. From Table 4.1, we make the following observations. 1) Our RMM consistently improves the two top baselines LUCIR+AANets and POD+AANets [LSS21a] in all settings. E.g., LUCIR-AANets *w/* RMM and POD-AANets *w/* RMM respectively get 2.7% and 3.1% average improvements on the ImageNet-Subset. 2) Our POD-AANets *w/* RMM achieves the best performances. Interestingly, we find that our RMM can boost performance more when the number of phases is larger. For example, when N=25, RMM improves POD-AANets by 3.6% and 4.4% on CIFAR-100 and ImageNet-Subset,

Ablation Setting	*CIFAR-100*						*ImagNet-Subset*					
	N=5		10		25		5		10		25	
	Avg	Last	Avg	Last	Avg	Last	Avg	Last	Avg	Last	Avg	Last
1 BaseRow	66.61	57.81	64.61	55.70	62.63	52.53	77.36	70.02	75.83	68.97	72.18	63.89
2 One-level RL	67.92	58.61	66.94	58.31	65.95	56.44	78.50	72.00	78.15	71.00	75.47	67.47
3 Two-level RL (Used)	68.86	59.00	67.61	59.03	66.21	56.50	79.52	73.80	78.47	71.40	76.54	68.84
margin	+2.3	+1.2	+3	+3.3	+3.6	+4	+2.1	+3.8	+2.6	+2.4	+4.4	+5
4 Two-level RL (T.P.)	68.62	59.40	67.22	58.20	65.82	56.20	78.81	72.42	77.68	70.77	75.29	68.81
margin	+2	+1.6	+2.6	+2.5	+3.2	+3.7	+1.5	+2.4	+1.9	+1.8	+3.1	+4.9
5 UpperBound RL	70.00	61.12	68.36	60.00	66.56	56.74	80.01	74.31	78.95	71.97	76.99	69.14
6 CrossVal Fixed	67.50	58.48	66.69	57.19	65.73	55.51	77.96	70.31	76.70	69.08	74.18	66.10

Table 4.2: The evaluation results in the ablation study (%). "T.P." denotes our results using the **P**olicy functions **T**ransferred from another dataset. "Avg", "Last", and "Used" denote the average accuracy over all phases, the last-phase accuracy, and the results used as ours in Table 4.1, respectively. BaseRow is from the sota method POD-AANets [LSS21a]. Row 2 is for learning Level-1 policy. Row 3 is for learning Level-1 and Level-2 policies in a hierarchical way. Row 4 is for using Transferred Policies (from the other dataset in the table), when RL is costly or impossible on target CIL tasks. The bottom lines are two oracles: training the RL model on the target CIL task (Row 5) and using cross-validation to find the best fixed memory allocation between old and new classes (Row 6).

respectively. These two numbers are 2.3% and 2.1% when N=5. This indicates that the superiority of our RMM is more obvious in challenging settings (where the forgetting problem is more serious due to the more frequent model re-training through phases).

Ablation Settings. Table 4.2 shows the results of our ablation study. Row 1 is for the baseline method POD-AANets [LSS21a]. Row 2 is for learning only Level-1 policy π_η (where each class gets an even split of the memory). Row 3 is for learning both Level-1 policy π_η and Level-2 policy π_ϕ in our proposed hierarchical method, and its results are used in Table 4.1 as "ours". Row 4 is for using Policy functions Transferred from another dataset (T.P.), which means on the target CIL dataset there is no training of RMM. Here, for CIFAR-100, we use the policy functions learned on ImageNet-Subset, and vice versa. On the last two rows, we show two oracle settings. Row 5 is the upper bound that assumes all past and future data are accessible during training RMM on the target CIL dataset. Row 6 is for using cross-validation (i.e., all past, future, and validation data are accessible) to find the best fixed memory split between old and new class data, e.g., $\frac{\text{old}}{\text{new}} = 0.7$ is chosen and then used in all phases.

Ablation Results. ***Hierarchical***: In Table 4.2, when comparing Row 2 to Row 1, it is clear that leveraging reinforcement learning yields better results as it can derive adaptive memory allocation between old and new data. Using class-specific memory management further increases the model performance (i.e., comparing Row 3 to Row 2), even though we divide the classes into only two groups. ***T.P.*** (**T**ransferred **P**olicy functions): Comparing Row 4 to Row 3, we can see that using transferred policy functions (trained on another dataset) achieves comparable performance, and Row 4 does not require any reinforcement learning

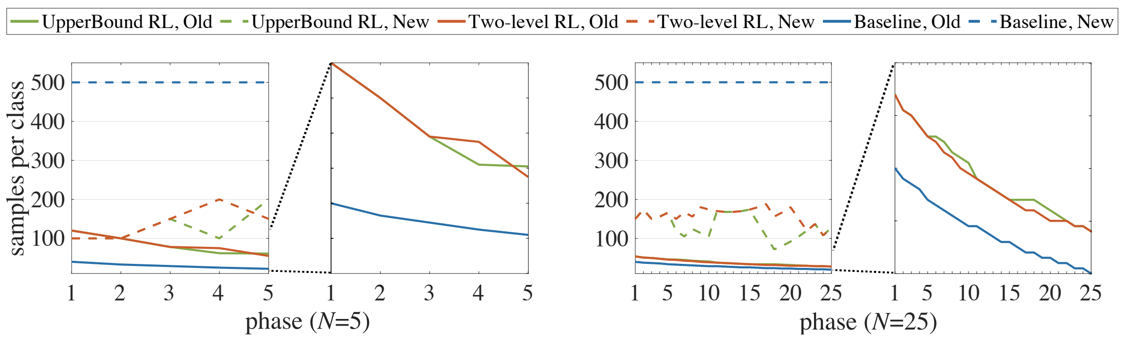


Figure 4.4: The memory allocated for “Old” and “New” across different phases on CIFAR-100. The second and fourth plots are enlarged versions of the first and third plots, respectively. Solid and dashed lines denote old and new classes, respectively. The baseline is POD-AANets [LSS21a]. “Two-level RL” and “UpperBound RL” correspond to Row 3 and Row 5 in Table 4.2, respectively.

on the target CIL dataset. ***Oracle***: Comparing Row 3 to Row 5, we see that learning RMM on pseudo CIL tasks is comparable to the upper bound case where all training and validation data are accessible, given the fact that the latter needs higher computational overhead and violates the standard CIL protocol. Row 6 results are consistently lower than ours in Row 3, although cross-validation has access to all past, future, and validation data.

Allocated Memory. Figure 4.4 shows the change of the average number of samples per class in three ablative settings. Solid and dashed lines represent old and new classes, respectively. From the plots, we have two observations. 1) Learning RMM on the pseudo or target CIL tasks (green and orange lines), we can obtain similar memory management results (i.e., actions). This means the learned policy is transferrable in non-stationary continuous environments. This matches the conclusion of continuous adaptation in [ABB+18]. 2) Using our RMM method achieved more balanced memory sizes between exemplars and new data. For example, in the 1-st phase of the 5-phase setting, “UpperBound RL” and “Two-level RL” allocate around 100 samples for both exemplars and new data. While the baseline setting has 40 and 500 samples for them, respectively. It thus addresses the data imbalance problem for CIL in a learnable way.

4.5 CONCLUSION

We propose the reinforced memory management (RMM) method specially for tackling CIL tasks. The hierarchical reinforcement learning (RL) framework (two levels) in RMM is capable of making more adaptive memory allocation actions than using standard RL (one level). Using the generated pseudo tasks in RMM solves the issue of data incompatibility between CIL and RL. Corresponding experimental results show that the policy trained on these pseudo tasks can be directly applied to target tasks without any computational overhead. Our overall method of RMM is generic, and its trained policy (with or without using an in-domain dataset) can be easily incorporated into exemplar replaying-based CIL methods to boost performance.

5 LEARNING TO AGGREGATE NEURAL NETWORKS ADAPTIVELY

Contents

An inherent problem in class-incremental learning (CIL) is the stability-plasticity dilemma between the learning of old and new classes, i.e., high-plasticity models easily forget old classes, but high-stability models are weak to learn new classes. We alleviate this issue by proposing a novel network architecture called Adaptive Aggregation Networks (AANets) in which we explicitly build two types of residual blocks at each residual level (taking ResNet as the baseline architecture): a stable block and a plastic block. We aggregate the output feature maps from these two blocks and then feed the results to the next-level blocks. We adapt the aggregation weights in order to balance these two types of blocks, i.e., to balance stability and plasticity, dynamically. We conduct extensive experiments on three CIL benchmarks: CIFAR-100, ImageNet-Subset, and ImageNet, and show that many existing CIL methods can be straightforwardly incorporated into the architecture of AANets to boost their performance.

This chapter is based on [LSS21a]. As the first author of [LSS21a], Yaoyao Liu conducted all experiments and was the main writer. This work has been cited more than 70 times and used as a baseline framework for many papers published in top-tier conferences, e.g., [JKK$^+$22, SZL$^+$22]. A CVPR 2022 paper [JKK$^+$22] especially thanks our work in its acknowledgments.

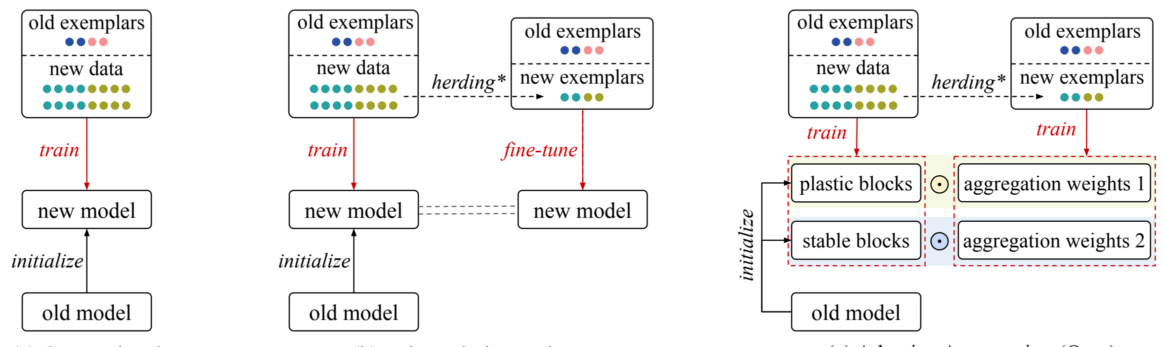
5.1 INTRODUCTION

AI systems are expected to work in an incremental manner when the amount of knowledge increases over time. They should be capable of learning new concepts while maintaining the ability to recognize previous ones. However, deep-neural-network-based systems often suffer from serious forgetting problems (called "catastrophic forgetting") when they are continuously updated using new coming data. This is due to two facts: (i) the updates can override the knowledge acquired from the previous data [MC89, MH93, Rat90, SLKK17, KMA+18], and (ii) the model can not replay the entire previous data to regain the old knowledge.

To encourage solving these problems, [RKSL17] defined a class-incremental learning (CIL) protocol for image classification where the training data of different classes gradually come phase-by-phase. In each phase, the classifier is re-trained on new class data, and then evaluated on the test data of both old and new classes. To prevent trivial algorithms such as storing all old data for replaying, there is a strict memory budget due to which a tiny set of exemplars of old classes can be saved in the memory. This memory constraint causes a serious data imbalance problem between old and new classes, and indirectly causes the main problem of CIL – the stability-plasticity dilemma [MBB13]. In particular, higher plasticity results in the forgetting of old classes [MC89], while higher stability weakens the model from learning the data of new classes (that contain a large number of samples). Existing CIL works try to balance stability and plasticity using data strategies. For example, as illustrated in Figure 5.1 (a) and (b), some early methods train their models on the imbalanced dataset where there is only a small set of exemplars for old classes [RKSL17, LH18], and recent methods include a fine-tuning step using a balanced subset of exemplars sampled from all classes [CMG+18, HPL+19, DCO+20]. However, these data strategies are still limited in terms of effectiveness. For example, when using the models trained after 25 phases, LUCIR [HPL+19] and Mnemonics [LSL+20] "forget" the initial 50 classes by 30% and 20%, respectively, on the ImageNet dataset [RDS+15].

In this chapter, we address the stability-plasticity dilemma by introducing a novel network architecture called Adaptive Aggregation Networks (AANets). Taking the ResNet [HZRS16] as an example of baseline architectures, we explicitly build two residual blocks (at each residual level) in AANets: one for maintaining the knowledge of old classes (i.e., the stability) and the other for learning new classes (i.e., the plasticity), as shown in Figure 5.1 (c). We achieve these by allowing these two blocks to have different levels of learnability, i.e., fewer learnable parameters in the stable block but more in the plastic one. We apply aggregation weights to the output feature maps of these blocks, sum them up, and pass the result maps to the next residual level. In this way, we are able to dynamically balance the usage of these blocks by updating their aggregation weights. To achieve auto-updating, we take the weights as hyperparameters and optimize them in an end-to-end manner [FAL17, WCW+19, LSL+20].

Technically, the overall optimization of AANets is bilevel. Level-1 is to learn the network parameters for two types of residual blocks, and level-2 is to adapt their aggregation weights. More specifically, level-1 is the standard optimization of network parameters, for which we use all the data available in the phase. Level-2 aims to balance the usage of the two types of blocks, for which we optimize the aggregation weights using a balanced subset (by downsampling the data of new classes), as illustrated in Figure 5.1 (c). We formulate these two levels in a bilevel optimization program (BOP) [SMD18] that solves two optimization problems alternatively, i.e., update network parameters with aggregation weights fixed, and then switch. For evaluation, we conduct CIL experiments on three widely-used benchmarks, CIFAR-100, ImageNet-Subset, and ImageNet. We find that many existing CIL methods, e.g., iCaRL [RKSL17], LUCIR [HPL+19], Mnemonics Training [LSL+20], and PODNet [DCO+20], can be directly incorporated in the architecture of AANets, yielding consistent performance



(a) Conventional (b) Balanced Fine-tuning (c) Adaptive Aggregation (Ours)

Figure 5.1: Conceptual illustrations of different CIL methods. (a) Conventional methods use all available data (which are imbalanced among classes) to train the model [RKSL17, HPL+19] (b) Recent methods [CMG+18, HPL+19, DCO+20, LSL+20] follow this convention but add a fine-tuning step on a balanced subset of all classes. (c) The proposed Adaptive Aggregation Networks (AANets) is a new architecture and it applies a different data strategy: using all available data to update the parameters of plastic and stable blocks, and the balanced set of exemplars to adapt the aggregation weights for these blocks. Our key lies in that adapted weights can balance the usage of the plastic and stable blocks, i.e., balance between plasticity and stability. *: *herding* is the method to choose exemplars [Wel09], and can be replaced by others, e.g., *mnemonics training* in [LSL+20].

improvements. We observe that a straightforward plug-in causes memory overheads, e.g., 26% and 15% respectively for CIFAR-100 and ImageNet-Subset. For a fair comparison, we conduct additional experiments under the settings of zero overhead (e.g., by reducing the number of old exemplars for training AANets), and validate that our approach still achieves top performance across all datasets.

Our contribution is three-fold: 1) a novel and generic network architecture called AANets specially designed for tackling the stability-plasticity dilemma in CIL tasks; 2) a BOP-based formulation and an end-to-end training solution for optimizing AANets; and 3) extensive experiments on three CIL benchmarks by incorporating four baseline methods in the architecture of AANets.

5.2 RELATED WORK

In this section, we discuss related works on bi-level optimization problems. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Bi-level Optimization Problem can be used to optimize hyperparameters of deep models. Technically, the network parameters are updated at one level and the key hyperparameters are updated at another level [VSV52, WZTE18, GPAM+14, ZCLS20, LSS20, LSL+19]. Recently, a few bilevel-optimization-based approaches have emerged for tackling incremental learning tasks. Wu et al. [WCW+19] learned a bias correction layer for incremental learning models using a bilevel optimization framework. Rajasegaran et al. [RKH+20] incrementally learned new tasks while learning a generic model to retain the knowledge from all tasks. Riemer et al. [RCA+19] learned network updates that are well-aligned with previous phases, such as to avoid learning towards any distracting directions. In our work, we apply the bilevel optimization program to update the aggregation weights in our AANets.

5.3 METHODOLOGY

Class-Incremental Learning (CIL) usually assumes $(N+1)$ learning phases in total, i.e., one initial phase and N incremental phases during which the number of classes gradually increases [HPL+19, LSL+20, DCO+20, HTM+21]. In the initial phase, data $\mathcal{D}_0$ is available to train the first model Θ_0. There is a strict memory budget in CIL systems, so after the phase, only a small subset of $\mathcal{D}_0$ (exemplars denoted as $\mathcal{E}_0$) can be stored in the memory and used as replay samples in later phases. Specifically in the i-th ($i \geq 1$) phase, we load the exemplars of old classes $\mathcal{E}_{0:i-1} = \{\mathcal{E}_0, \dots, \mathcal{E}_{i-1}\}$ to train model Θ_i together with new class data $\mathcal{D}_i$. Then, we evaluate the trained model on the test data containing both old and new classes. We repeat such training and evaluation through all phases.

The key issue of CIL is that the models trained at new phases easily "forget" old classes. To tackle this, we introduce a novel architecture called AANets. AANets is based on a ResNet-type architecture, and each of its residual levels is composed of two types of residual blocks: a plastic one to adapt to new class data and a stable one to maintain the knowledge learned from old classes. The details of this architecture are elaborated in Section 5.3.1. The steps for optimizing AANets are given in Section 5.3.2.

5.3.1 Architecture Details

In Figure 5.2, we provide an illustrative example of our AANets with three residual levels. The inputs $x^{[0]}$ are the images and the outputs $x^{[3]}$ are the features used to train classifiers. Each of our residual "levels" consists of two parallel residual "blocks" (of the original ResNet [HZRS16]): the orange one (called plastic block) will have its parameters fully adapted to new class data, while the blue one (called stable block) has its parameters partially fixed in order to maintain the knowledge learned from old classes. After feeding the inputs to Level 1, we obtain two sets of feature maps respectively from two blocks, and aggregate them after applying the aggregation weights $\alpha^{[1]}$. Then, we feed the resulting maps to Level 2 and repeat the aggregation. We apply the same steps for Level 3. Finally, we pool the resulting maps obtained from Level 3 to train classifiers. Below we elaborate on the details of this dual-branch design as well as the steps for feature extraction and aggregation.

Stable and Plastic Blocks. We deploy a pair of stable and plastic blocks at each residual level, aiming to balance between the plasticity, i.e., for learning new classes, and stability, i.e., for not forgetting the knowledge of old classes. We achieve these two types of blocks by allowing different levels of learnability, i.e., fewer learnable parameters in the stable block but more in the plastic. We detail the operations in the following. In any CIL phase, Let η and ϕ represent the learnable parameters of plastic and stable blocks, respectively. η contains all the convolutional weights, while ϕ contains only the neuron-level scaling weights [SLCS19]. Specifically, these scaling weights are applied on the model θ_{base} obtained in the 0-th phase[1]. As a result, the number of learnable parameters ϕ is much less than that of η. For example, when using 3×3 neurons in θ_{base}, the number of learnable parameters ϕ is only $\frac{1}{3\times3}$ of the number of full network parameters (while η has the full network parameters). We further

[1]Related work [HPL+19, DCO+20, LSL+20] learned Θ_0 in the 0-th phase using half of the total classes. We follow the same way to train Θ_0 and freeze it as θ_{base}.

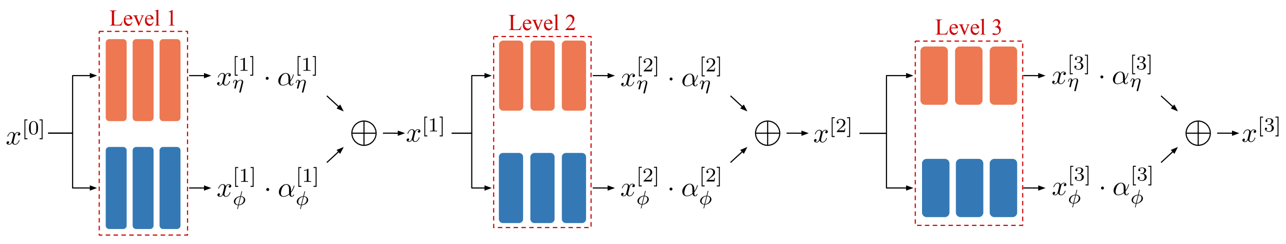

Level 1
Level 2
Level 3
$x^{[0]}$
$x_\eta^{[1]} \cdot \alpha_\eta^{[1]}$
$x_\phi^{[1]} \cdot \alpha_\phi^{[1]}$
$x^{[1]}$
$x_\eta^{[2]} \cdot \alpha_\eta^{[2]}$
$x_\phi^{[2]} \cdot \alpha_\phi^{[2]}$
$x^{[2]}$
$x_\eta^{[3]} \cdot \alpha_\eta^{[3]}$
$x_\phi^{[3]} \cdot \alpha_\phi^{[3]}$
$x^{[3]}$

Figure 5.2: An example architecture of AANets with three levels of residual blocks. At each level, we compute the feature maps from a stable block ($\phi \odot \theta_{\text{base}}$, blue) as well as a plastic block (η, orange), respectively, aggregate the maps with adapted weights, and feed the result maps to the next level. The outputs of the final level are used to train classifiers. **We highlight that this is a logical architecture of AANets, and in real implementations, we strictly control the memory (i.e., the sizes of input data and residual blocks) within the same budget as related works which deploy plain ResNets. Please refer to the details in the section on experiments.**

elaborate on these in the following paragraph.

Neuron-level Scaling Weights. For stable blocks, we learn their neuron parameters in the 0-th phase and freeze them in the other N phases. In these N phases, we apply a small set of scaling weights ϕ at the neuron-level, i.e., each weight for scaling one neuron in θ_{base}. We aim to preserve the structural pattern within the neuron and slowly adapt the knowledge of the whole blocks to new class data. Specifically, we assume the q-th layer of θ_{base} contains R neurons, so we have R neuron weights as $\{W_{q,r}\}_{r=1}^{R}$. For conciseness, we denote them as W_q. For W_q, we learn R scaling weights denoted as ϕ_q Let X_{q-1} and X_q be the input and output feature maps of the q-th layer, respectively. We apply ϕ_q to W_q as follows,

$$X_q = (W_q \odot \phi_q) X_{q-1}, \tag{5.1}$$

where $\odot$ donates the element-wise multiplication. Assuming there are Q layers in total, the overall scaling weights can be denoted as $\phi = \{\phi_q\}_{q=1}^{Q}$.

Feature Extraction and Aggregation. We elaborate on the process of feature extraction and aggregation across all residual levels in the AANets, as illustrated in Figure 5.2. Let $\mathcal{F}_\mu^{[k]}(\cdot)$ denote the transformation function of the residual block parameterized as μ at the Level k. Given a batch of training images $x^{[0]}$, we feed them to AANets to compute the feature maps at the k-th level (through the stable and plastic blocks respectively) as follows,

$$x_\phi^{[k]} = \mathcal{F}^{[k]}_{\phi \odot \theta_{\text{base}}}(x^{[k-1]}); \quad x_\eta^{[k]} = \mathcal{F}_\eta^{[k]}(x^{[k-1]}). \tag{5.2}$$

The transferability (of the knowledge learned from old classes) is different at different levels of neural networks [YCBL14]. Therefore, it makes more sense to apply different aggregation weights for different levels of residual blocks. Let $\alpha_\phi^{[k]}$ and $\alpha_\eta^{[k]}$ denote the aggregation weights of the stable and plastic blocks, respectively, at the k-th level. Then, the weighted sum of $x_\phi^{[k]}$ and $x_\eta^{[k]}$ can be derived as follows,

$$x^{[k]} = \alpha_\phi^{[k]} \cdot x_\phi^{[k]} + \alpha_\eta^{[k]} \cdot x_\eta^{[k]}. \tag{5.3}$$

In our illustrative example in Figure 5.2, there are three pairs of weights to learn at each phase. Hence, it becomes increasingly challenging to choose these weights manually if multiple phases are involved. In this chapter, we propose a learning strategy to automatically adapt these weights, i.e., optimizing the weights for different blocks in different phases. See details in Section 5.3.2.

5.3.2 Optimization Steps

In each incremental phase, we optimize two groups of learnable parameters in AANets: (a) the neuron-level scaling weights ϕ for the stable blocks and the convolutional weights η on the plastic blocks; (b) the feature aggregation weights α. The former is for network parameters and the latter is for hyperparameters. In this chapter, we formulate the overall optimization process as a bilevel optimization program (BOP) [GPAM+14, LSL+20].

The Formulation of BOP. In AANets, the network parameters $[\phi, \eta]$ are trained using the aggregation weights α as hyperparameters. In turn, α can be updated when temporarily fixing network parameters $[\phi, \eta]$. In this way, the optimality of $[\phi, \eta]$ imposes a constraint on α and vise versa. Ideally, in the i-th phase, the CIL system aims to learn the optimal α_i and $[\phi_i, \eta_i]$ that minimize the classification loss on all training samples seen so far, i.e., $\mathcal{D}_i \cup \mathcal{D}_{0:i-1}$, so the ideal BOP can be formulated as,

$$\min_{\alpha_i} \mathcal{L}(\alpha_i, \phi_i^*, \eta_i^*; \mathcal{D}_{0:i-1} \cup \mathcal{D}_i) \tag{5.4a}$$

$$\text{s.t. } [\phi_i^*, \eta_i^*] = \arg\min_{[\phi_i, \eta_i]} \mathcal{L}(\alpha_i, \phi_i, \eta_i; \mathcal{D}_{0:i-1} \cup \mathcal{D}_i), \tag{5.4b}$$

where $\mathcal{L}(\cdot)$ denotes the loss function, e.g., cross-entropy loss. Please note that for the conciseness of the formulation, we use ϕ_i to represent $\phi_i \odot \theta_{\text{base}}$ (same in the following equations). We call Problem 5.4a and Problem 5.4b the *upper-level* and *lower-level* problems, respectively.

Data Strategy. To solve Problem 5.4, we need to use $\mathcal{D}_{0:i-1}$. However, in the setting of CIL [RKSL17, HPL+19, DCO+20], we cannot access $\mathcal{D}_{0:i-1}$ but only a small set of exemplars $\mathcal{E}_{0:i-1}$, e.g., 20 samples of each old class. Directly replacing $\mathcal{D}_{0:i-1} \cup \mathcal{D}_i$ with $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ in Problem 5.4 will lead to the forgetting problem for the old classes. To alleviate this issue, we propose a new data strategy in which we use different training data splits to learn different groups of parameters: 1) in the *upper-level* problem, α_i is used to balance the stable and the plastic blocks, so we use the balanced subset to update it, i.e., learning α_i on $\mathcal{E}_{0:i-1} \cup \mathcal{E}_i$ adaptively; 2) in the *lower-level* problem, $[\phi_i, \eta_i]$ are the network parameters used for feature extraction, so we leverage all the available data to train them, i.e., base-training $[\phi_i, \eta_i]$ on $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. Based on these, we can reformulate the ideal BOP in Problem 5.4 as a solvable BOP as follows,

$$\min_{\alpha_i} \mathcal{L}(\alpha_i, \phi_i^*, \eta_i^*; \mathcal{E}_{0:i-1} \cup \mathcal{E}_i) \tag{5.5a}$$

$$\text{s.t. } [\phi_i^*, \eta_i^*] = \arg\min_{[\phi_i, \eta_i]} \mathcal{L}(\alpha_i, \phi_i, \eta_i; \mathcal{E}_{0:i-1} \cup \mathcal{D}_i), \tag{5.5b}$$

where Problem 5.5a is the *upper-level* problem and Problem 5.5b is the *lower-level* problem we are going to solve.

Updating Parameters. We solve Problem 5.5 by alternatively updating two groups of parameters (α_i and $[\phi, \eta]$) across epochs, e.g., if α_i is updated in the j-th epoch, then $[\phi, \eta]$

will be updated in the $(j+1)$-th epoch, until both of them converge. Taking the i-th phase as an example, we initialize α_i, ϕ_i, η_i with $\alpha_{i-1}, \phi_{i-1}, \eta_{i-1}$, respectively. Please note that ϕ_0 is initialized with ones, following [SLCS19, SLC+22]; η_0 is initialized with θ_{base}; and α_0 is initialized with 0.5. Based on our **Data Strategy**, we use all available data in the current phase to solve the *lower-level* problem, i.e., training $[\phi_i, \eta_i]$ as follows,

$$[\phi_i, \eta_i] \leftarrow [\phi_i, \eta_i] - \gamma_1 \nabla_{[\phi_i, \eta_i]} \mathcal{L}(\alpha_i, \phi_i, \eta_i; \mathcal{E}_{0:i-1} \cup \mathcal{D}_i). \tag{5.6}$$

Then, we use a balanced exemplar set to solve the *upper-level* problem, i.e., training α_i as follows,

$$\alpha_i \leftarrow \alpha_i - \gamma_2 \nabla_{\alpha_i} \mathcal{L}(\alpha_i, \phi_i, \eta_i; \mathcal{E}_{0:i-1} \cup \mathcal{E}_i), \tag{5.7}$$

where γ_1 and γ_2 are the *lower-level* and *upper-level* learning rates, respectively.

Algorithm 3: AANets (in the i-th phase)

```
Input:  New class data $\mathcal{D}_i$; old class exemplars $\mathcal{E}_{0:i-1}$; old parameters $\alpha_{i-1}$, $\phi_{i-1}$, $\eta_{i-1}$;
        base model $\theta_{\text{base}}$.
Output: new parameters $\alpha_i$, $\phi_i$, $\eta_i$; new class exemplars $\mathcal{E}_i$.
1  Get $\mathcal{D}_i$ and load $\mathcal{E}_{0:i-1}$ from memory;
2  Initialize $[\phi_i, \eta_i]$ with $[\phi_{i-1}, \eta_{i-1}]$;
3  Initialize $\alpha_i$ with $\alpha_{i-1}$;
4  Select exemplars $\mathcal{E}_i \subsetneq \mathcal{D}_i$, e.g. by herding [RKSL17, HPL+19] or mnemonics
     training [LSL+20];
5  for epochs do
6      for mini-batches in $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ do
7          Train $[\phi_i, \eta_i]$ on $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ by Eq. 5.6;
8      end
9      for mini-batches in $\mathcal{E}_{0:i-1} \cup \mathcal{E}_i$ do
10         Learn $\alpha_i$ on $\mathcal{E}_{0:i-1} \cup \mathcal{E}_i$ by Eq. 5.7;
11     end
12 end
13 Update exemplars $\mathcal{E}_i$, e.g. by herding [RKSL17, HPL+19] or mnemonics
     training [LSL+20];
14 Replace $\mathcal{E}_{0:i-1}$ with $\mathcal{E}_{0:i-1} \cup \mathcal{E}_i$ in the memory.
```

5.3.3 Algorithm

In Algorithm 3, we summarize the overall training steps of the proposed AANets in the i-th incremental learning phase (where $i \in [1, ..., N]$). Lines 1-4 show the preprocessing including loading new data and old exemplars (Line 1), initializing the two groups of learnable parameters (Lines 2-3), and selecting the exemplars for new classes (Line 4). Lines 5-12 optimize alternatively between the network parameters and the Adaptive Aggregation weights. In specific, Lines 6-8 and Lines 9-11 execute the training for solving the *upper-level* and *lower-level* problems, respectively. Lines 13-14 update the exemplars and save them to the memory.

5.4 EXPERIMENTS

We evaluate the proposed AANets on three CIL benchmarks, i.e., CIFAR-100 [KH+09], ImageNet-Subset [RKSL17] and ImageNet [RDS+15]. We incorporate AANets into four baseline methods and boost their model performances consistently for all settings. Below we describe the datasets and implementation details (Section 5.4.1), followed by the results and analyses (Section 5.4.2), which include a detailed ablation study, extensive comparisons to related methods, and some visualization of the results.

5.4.1 Datasets and Implementation Details

Datasets. We conduct CIL experiments on two datasets, CIFAR-100 [KH+09] and ImageNet [RDS+15], following closely related work [HPL+19, LSL+20, DCO+20]. CIFAR-100 contains $60,000$ samples of 32×32 color images for 100 classes. There are 500 training and 100 test samples for each class. ImageNet contains around 1.3 million samples of 224×224 color images for 1000 classes. There are approximately $1,300$ training and 50 test samples for each class. ImageNet is used in two CIL settings: one based on a subset of 100 classes (ImageNet-Subset) and the other based on the full set of $1,000$ classes. The 100-class data for ImageNet-Subset are sampled from ImageNet in the same way as [HPL+19, DCO+20].

Architectures. Following the exact settings in [HPL+19, LSL+20], we deploy a 32-layer ResNet as the baseline architecture (based on which we build the AANets) for CIFAR-100. This ResNet consists of 1 initial convolution layer and 3 residual blocks (in a single branch). Each block has 10 convolution layers with 3×3 kernels. The number of filters starts from 16 and is doubled every next block. After these 3 blocks, there is an average-pooling layer to compress the output feature maps to a feature embedding. To build AANets, we convert these 3 blocks into three levels of blocks and each level consists of a stable block and a plastic block, referring to Section 5.3.1. Similarly, we build AANets for ImageNet benchmarks but taking an 18-layer ResNet [HZRS16] as the baseline architecture [HPL+19, LSL+20]. Please note that there is no architecture change applied to the classifiers, i.e., using the same FC layers as in [HPL+19, LSL+20].

Hyperparameters and Configuration. The learning rates γ_1 and γ_2 are initialized as 0.1 and 1×10^{-8}, respectively. We impose a constraint on each pair of α_η and α_ϕ to make sure $\alpha_\eta + \alpha_\phi = 1$. For a fair comparison, our training hyperparamters are almost the same as in [DCO+20, LSL+20]. Specifically, on the CIFAR-100 (ImageNet), we train the model for 160 (90) epochs in each phase, and the learning rates are divided by 10 after 80 (30) and then after 120 (60) epochs. We use an SGD optimizer with a momentum 0.9 and a batch size 128 to train the models in all settings.

Memory Budget. By default, we follow the same data replay settings used in [RKSL17, HPL+19, LSL+20, DCO+20], where each time reserves 20 exemplars per old class. In our "strict memory budget" settings, we strictly control the memory budget shared by the exemplars and the model parameters. For example, if we incorporate AANets to LUCIR [HPL+19], we need to reduce the number of exemplars to balance the additional memory used by model parameters (as AANets take around 20% more parameters than plain ResNets). As a result, we reduce the numbers of exemplars for AANets from 20 to 13, 16, and 19, respectively, for CIFAR-100, ImageNet-Subset, and ImageNet, in the "strict memory budget" setting. For example, on CIFAR-100, we use $530k$ additional parameters,

Row	Ablation Setting	*CIFAR-100* (acc.%)					
		Memory	FLOPs	#Param	N=5	10	25
1	single-branch "all" [HPL+19]	7.64MB	70M	469K	63.17	60.14	57.54
2	"all" + "all"	9.43MB	140M	938K	64.49	61.89	58.87
3	"all" + "scaling"	9.66MB	140M	530K	**66.74**	**65.29**	**63.50**
4	"all" + "frozen"	9.43MB	140M	469K	65.62	64.05	63.67
5	"scaling" + "frozen"	9.66MB	140M	60K	64.71	63.65	62.89
6	*w/o* balanced $\mathcal{E}$	9.66MB	140M	530K	65.91	64.70	63.08
7	*w/o* adapted α	9.66MB	140M	530K	65.89	64.49	62.89
8	strict memory budget	7.64MB	140M	530K	66.46	65.38	61.79

Row	Ablation Setting	*ImageNet-Subset* (acc.%)					
		Memory	FLOPs	#Param	N=5	10	25
1	single-branch "all" [HPL+19]	330MB	1.82G	11.2M	70.84	68.32	61.44
2	"all" + "all"	372MB	3.64G	22.4M	69.72	66.69	63.29
3	"all" + "scaling"	378MB	3.64G	12.6M	72.55	69.22	67.60
4	"all" + "frozen"	372MB	3.64G	11.2M	71.71	69.87	67.92
5	"scaling" + "frozen"	378MB	3.64G	1.4M	**73.01**	**71.65**	**70.30**
6	*w/o* balanced $\mathcal{E}$	378MB	3.64G	12.6M	70.30	69.92	66.89
7	*w/o* adapted α	378MB	3.64G	12.6M	70.31	68.71	66.34
8	strict memory budget	330MB	3.64G	12.6M	72.21	69.10	67.10

Table 5.1: Ablation study. The baseline (Row 1) is LUCIR [HPL+19]. "all", "scaling", and "frozen" denote three types of blocks and they have different numbers of learnable parameters, e.g., "all" means all convolutional weights and biases are learnable. If we name them as A, B, and C, we use A+B in the table to denote the setting of using A-type and B-type blocks respectively as plastic and stable blocks. See more details in Section 5.4.2 **Ablation settings**. Adapted α are applied on Rows 3-8. "all"+"scaling" is the default setting of Rows 6-8. "#Param" indicates the number of learnable parameters. "Memory" denotes the peak memory for storing the exemplars and the learnable & frozen network parameters during the model training through all phases.

so we need to reduce $530k$floats $\times$ 4bytes/float $\div$ $(32 \times 32 \times 3$bytes/image$)$ $\div$ 100classes $\approx$ 7images/class.

Benchmark Protocol. We follow the common protocol used in [HPL+19, LSL+20, DCO+20]. Given a dataset, the model is trained on half of the classes in the 0-th phase. Then, it learns the remaining classes evenly in the subsequent N phases. For N, there are three options as 5, 10, and 25, and the corresponding settings are called "N-phase". In each phase, the model is evaluated on the test data for all seen classes. The average accuracy (over all phases) is reported. For each setting, we run the experiment three times and report averages and 95% confidence intervals.

5.4.2 Results and Analyses

Table 5.1 summarizes the statistics and results in 8 ablative settings. Table 5.2 presents the results of 4 state-of-the-art methods *w/* and *w/o* AANets as a plug-in architecture, and the reported results from some other comparable work. Figure 5.3 compares the activation maps (by Grad-CAM [SCD+17]) produced by different types of residual blocks and for the classes seen in different phases. Figure 5.4 shows the changes of values of α_η and α_ϕ across 10 incremental phases.

Ablation Settings. Table 5.1 shows the ablation study. By differentiating the numbers of learnable parameters, we can have 3 block types: 1) "all" for learning all the convolutional weights and biases; 2) "scaling" for learning neuron-level scaling weights [SLCS19] on the top of a frozen base model θ_{base}; and 3) "frozen" for using only θ_{base} (always frozen). In Table 5.1, the pattern of combining blocks is A+B where A and B stand for the plastic and the stable blocks, respectively. Row 1 is the baseline method LUCIR [HPL+19]. Row 2 is a double-branch version for LUCIR without learning any aggregation weights. Rows 3-5 are our AANets using different combinations of blocks. Row 6-8 use "all"+"scaling" under an additional setting as follows. 1) Row 6 uses imbalanced data $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ to train α adaptively. 2) Row 7 uses fixed weights $\alpha_\eta = \alpha_\phi = 0.5$ at each residual level. 3) Row 8 is under the "strict memory budget" setting, where we reduce the numbers of exemplars to 14 and 17 for CIFAR-100 and ImageNet-Subset, respectively.

Ablation Results. In Table 5.1, if comparing the second block (ours) to the first block (single-branch and double-branch baselines), it is obvious that using AANets can clearly improve the model performance, e.g., "scaling"+"frozen" gains an average of 4.8% over LUCIR for the ImageNet-Subset, by optimizing $1.4M$ parameters during CIL — only 12.6% of that in LUCIR. Among Rows 3-5, we can see that for the ImageNet-Subset, models with the fewest learnable parameters ("scaling"+"frozen") work the best. We think this is because we use shallower networks for learning larger datasets (ResNet-32 for CIFAR-100; ResNet-18 for ImageNet-Subset), following the **Benchmark Protocol**. In other words, θ_{base} is quite well-trained with the rich data of half ImageNet-Subset (50 classes in the 0-th phase), and can offer high-quality features for later phases. Comparing Row 6 to Row 3 shows the efficiency of using a balanced subset to optimize α. Comparing Row 7 to Row 3 shows the superiority of learning α (which is dynamic and optimal) over manually choosing α.

About the Memory Usage. By comparing Row 3 to Row 1, we can see that AANets can clearly improve the model performance while introducing small overheads for the memory, e.g., 26% and 14.5% on the CIFAR-100 and ImageNet-Subset, respectively. If comparing Row 8 to Row 3, we find that though the numbers of exemplars are reduced (for Row 8), the model performance of AANets has a very small drop, e.g., only 0.3% for the 5-Phase CIL

Dataset: CIFAR-100	N=5	10	25
BiC [WCW+19]	59.36	54.20	50.00
TPCIL [TCH+20]	65.34	63.58	–
iCaRL [RKSL17]	57.12$_{\pm 0.50}$	52.66$_{\pm 0.89}$	48.22$_{\pm 0.76}$
w/ AANets (ours)	64.22$_{\pm 0.42}$	60.26$_{\pm 0.73}$	56.43$_{\pm 0.81}$
LUCIR [HPL+19]	63.17$_{\pm 0.87}$	60.14$_{\pm 0.73}$	57.54$_{\pm 0.43}$
w/ AANets (ours)	66.74$_{\pm 0.37}$	65.29$_{\pm 0.43}$	**63.50**$_{\pm 0.61}$
Mnemonics [LSL+20]	63.34$_{\pm 0.62}$	62.28$_{\pm 0.43}$	60.96$_{\pm 0.72}$
w/ AANets (ours)	**67.59**$_{\pm 0.34}$	**65.66**$_{\pm 0.61}$	63.35$_{\pm 0.72}$
PODNet-CNN [DCO+20]	64.83$_{\pm 1.11}$	63.19$_{\pm 1.31}$	60.72$_{\pm 1.54}$
w/ AANets (ours)	66.31$_{\pm 0.87}$	64.31$_{\pm 0.90}$	62.31$_{\pm 1.02}$

Dataset: ImageNet-Subset	N=5	10	25
BiC [WCW+19]	70.07	64.96	57.73
TPCIL [TCH+20]	76.27	74.81	–
iCaRL [RKSL17]	65.44$_{\pm 0.35}$	59.88$_{\pm 0.83}$	52.97$_{\pm 1.02}$
w/ AANets (ours)	73.45$_{\pm 0.51}$	71.78$_{\pm 0.64}$	69.22$_{\pm 0.83}$
LUCIR [HPL+19]	70.84$_{\pm 0.69}$	68.32$_{\pm 0.81}$	61.44$_{\pm 0.91}$
w/ AANets (ours)	72.55$_{\pm 0.67}$	69.22$_{\pm 0.72}$	67.60$_{\pm 0.39}$
Mnemonics [LSL+20]	72.58$_{\pm 0.85}$	71.37$_{\pm 0.56}$	69.74$_{\pm 0.39}$
w/ AANets (ours)	72.91$_{\pm 0.53}$	71.93$_{\pm 0.37}$	70.70$_{\pm 0.45}$
PODNet-CNN [DCO+20]	75.54$_{\pm 0.29}$	74.33$_{\pm 1.05}$	68.31$_{\pm 2.77}$
w/ AANets (ours)	**76.96**$_{\pm 0.53}$	**75.58**$_{\pm 0.74}$	**71.78**$_{\pm 0.81}$

Dataset: ImageNet	N=5	10	25
BiC [WCW+19]	70.07	64.96	57.73
TPCIL [TCH+20]	76.27	74.81	–
iCaRL [RKSL17]	65.44$_{\pm 0.35}$	59.88$_{\pm 0.83}$	52.97$_{\pm 1.02}$
w/ AANets (ours)	73.45$_{\pm 0.51}$	71.78$_{\pm 0.64}$	69.22$_{\pm 0.83}$
LUCIR [HPL+19]	70.84$_{\pm 0.69}$	68.32$_{\pm 0.81}$	61.44$_{\pm 0.91}$
w/ AANets (ours)	72.55$_{\pm 0.67}$	69.22$_{\pm 0.72}$	67.60$_{\pm 0.39}$
Mnemonics [LSL+20]	72.58$_{\pm 0.85}$	71.37$_{\pm 0.56}$	69.74$_{\pm 0.39}$
w/ AANets (ours)	72.91$_{\pm 0.53}$	71.93$_{\pm 0.37}$	70.70$_{\pm 0.45}$
PODNet-CNN [DCO+20]	75.54$_{\pm 0.29}$	74.33$_{\pm 1.05}$	68.31$_{\pm 2.77}$
w/ AANets (ours)	**76.96**$_{\pm 0.53}$	**75.58**$_{\pm 0.74}$	**71.78**$_{\pm 0.81}$

Table 5.2: Average incremental accuracies (%) of four state-of-the-art methods *w/* and *w/o* our AANets as a plug-in architecture. In the upper block, we present some comparable results reported in some other related works. Please note 1) [DCO+20] didn't report the results for N=25 on the ImageNet, and we produce the results using their public code; 2) [LSL+20] updated their results on arXiv (after fixing a bug in their code), different from its conference version; 3) for "*w/* AANets", we use "all"+"scaling" blocks corresponding to Row 3 of Table 5.1.

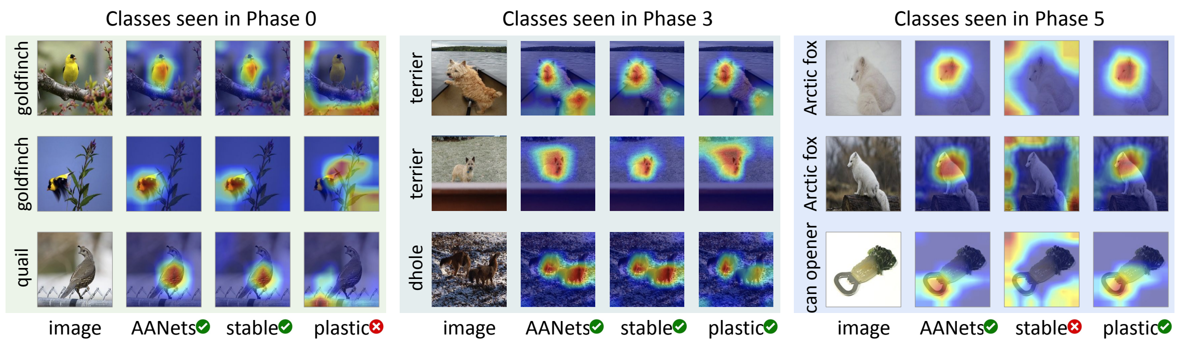


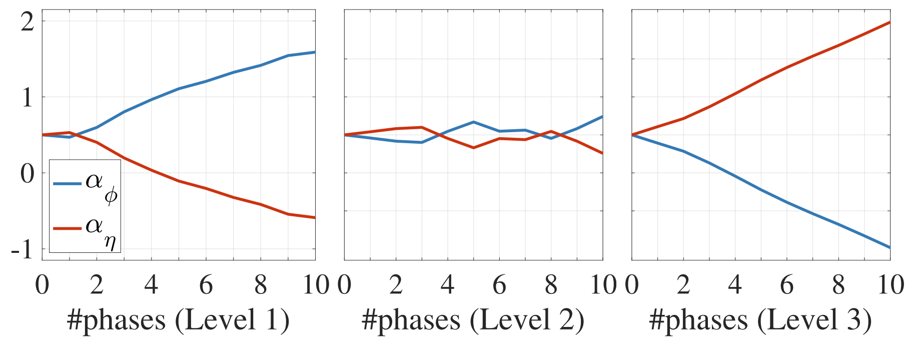
Figure 5.3: The activation maps using Grad-CAM [SCD+17] for the 5-th phase (the last phase) model on ImageNet-Subset (N=5). Samples are selected from the classes coming in the 0-th phase (left), the 3-rd phase (middle), and the 5-th phase (right), respectively. Green tick (red cross) means the discriminative features are activated on the object regions successfully (unsuccessfully). $\bar{\alpha}_\eta = 0.428$ and $\bar{\alpha}_\phi = 0.572$.

models of CIFAR-100 and ImageNet-Subset. Therefore, we can conclude that AANets can achieve rather satisfactory performance under strict memory control — a desirable feature needed in class-incremental learning systems.

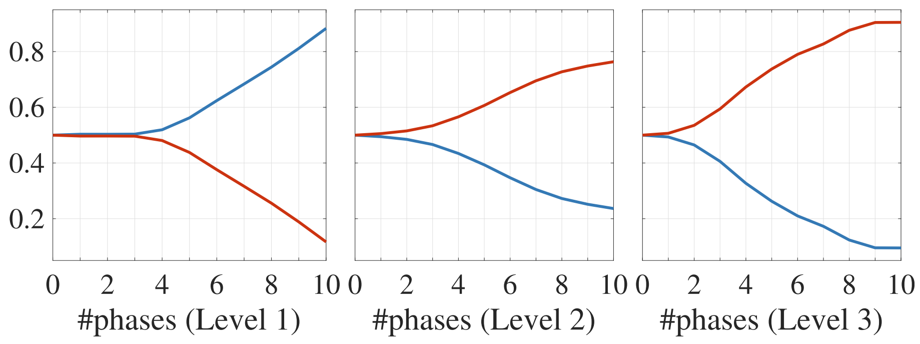
Comparing to the State-of-the-Art. Table 5.2 shows that taking our AANets as a plug-in architecture for 4 state-of-the-art methods [RKSL17, HPL+19, LSL+20, DCO+20] consistently improves their model performances. E.g., for CIFAR-100, LUCIR *w/* AANets and Mnemonics *w/* AANets respectively gains 4.9% and 3.3% improvements on average. From Table 5.2, we can see that our approach of using AANets achieves top performances in all settings. Interestingly, we find that AANets can boost more performance for simpler baseline methods, e.g., iCaRL. iCaRL *w/* AANets achieves mostly better results than those of LUCIR on three datasets, even though the latter method deploys various regularization techniques.

Visualizing Activation Maps. Figure 5.3 demonstrates the activation maps visualized by Grad-CAM for the final model (obtained after 5 phases) on ImageNet-Subset (N=5). The visualized samples from left to right are picked from the classes coming in the 0-th, 3-rd, and 5-th phases, respectively. For the 0-th phase samples, the model makes the prediction according to foreground regions (right) detected by the stable block and background regions (wrong) by the plastic block. This is because, through multiple phases of full updates, the plastic block forgets the knowledge of these old samples while the stable block successfully retains it. This situation is reversed when using that model to recognize the 5-th phase samples. The reason is that the stable block is far less learnable than the plastic block, and may fail to adapt to new data. For all shown samples, the model extracts features as informative as possible in two blocks. Then, it aggregates these features using the weights adapted from the balanced dataset, and thus can make a good balance of the features to achieve the best prediction.

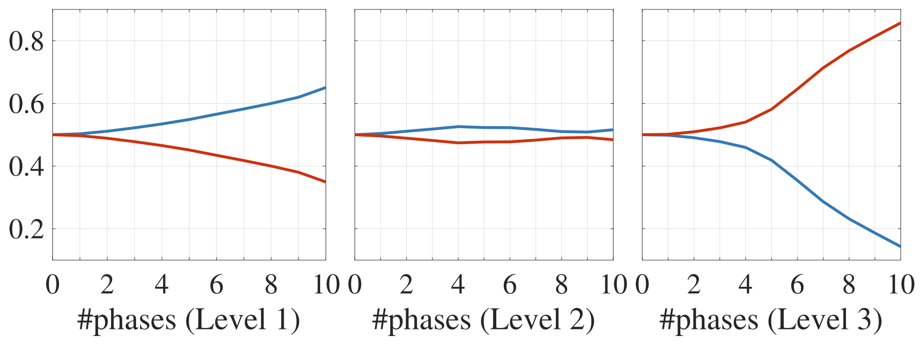
Aggregation Weights (α_η and α_ϕ). Figure 5.4 shows the values of α_η and α_ϕ learned during training 10-phase models. Each row displays three plots for three residual levels of AANets, respectively. Comparing among columns, we can see that Level 1 tends to get larger values of α_ϕ, while Level 3 tends to get larger values of α_η. This can be interpreted as lower-level residual blocks learning to stay stable, which is intuitively correct in deep models. With respect to the learning activity of CIL models, it is to continuously transfer the learned



(a) CIFAR-100 (N=10)



(b) ImageNet-Subset (N=10)



(c) ImageNet (N=10)

Figure 5.4: The values of α_η and α_ϕ adapted *for each residual level* and *in each incremental phase*. All curves are smoothed with a rate of 0.8 for better visualization.

knowledge to subsequent phases. The features at different resolutions (levels in our case) have different transferabilities [YCBL14]. Level 1 encodes low-level features that are more stable and shareable among all classes. Level 3 nears the classifiers, and tends to be more plastic such as to fast to adapt to new classes.

5.5 CONCLUSIONS

We introduce a novel network architecture AANets specially for CIL. Our main contribution lies in addressing the issue of stability-plasticity dilemma in CIL by a simple modification on plain ResNets — applying two types of residual blocks to respectively and specifically learn stability and plasticity at each residual level, and then aggregating them as a final representation. To achieve efficient aggregation, we adapt the level-specific and phase-specific weights in an end-to-end manner. Our overall approach is *generic* and can be easily incorporated into existing CIL methods to boost their performance.

6 LEARNING TO OPTIMIZE THE HYPERPARAMETER ONLINE

Contents

In Chapter 5, we show a dynamic architecture to address the stability-plasticity trade-off in class-incremental learning (CIL). However, none of the existing CIL models can achieve the optimal trade-off in different data-receiving settings—where typically the training-from-half (TFH) setting needs more stability, but the training-from-scratch (TFS) needs more plasticity. To this end, we design an online learning method that can adaptively optimize the tradeoff without knowing the setting as a priori. Specifically, we first introduce the key hyperparameters that influence the tradeoff, e.g., knowledge distillation (KD) loss weights, learning rates, and classifier types. Then, we formulate the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose a specific algorithm to solve it. We apply local estimated rewards and a classic bandit algorithm Exp3 [ACBFS02] to address the issues when applying online MDP methods to the CIL protocol. Our method consistently improves top-performing CIL methods in both TFH and TFS settings, e.g., boosting the average accuracy of TFH and TFS by 2.2 percentage points on ImageNet-Full, compared to the state-of-the-art [LSS21b].

This chapter is based on [LLSS23a]. As the first author, Yaoyao Liu conducted all experiments and was the main writer.

6.1 INTRODUCTION

Real-world problems are ever-changing, with new concepts and new data being continuously observed. Ideal AI systems should have the ability to learn new concepts from the new data, also known as *plasticity*, while maintaining the ability to recognize old concepts, also

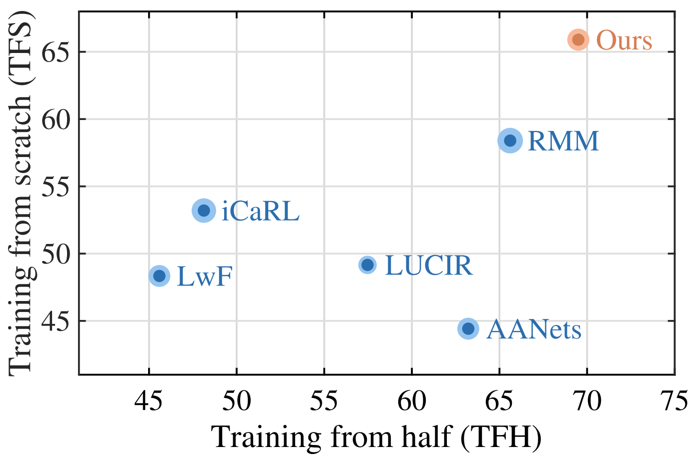


Figure 6.1: Average accuracy (%) on CIFAR-100 25-phase, using two data-receiving settings: 1) *training-from-half* (TFH): a large amount of data is available beforehand to pre-train the encoder; 2) *training-from-scratch* (TFS): classes come evenly in each phase. **Dark blue** and **orange** indicate the baselines and our method, respectively. Light-color circles are confidence intervals. Notice that methods with strong KD losses, e.g., LUCIR [HPL+19], AANets [LSS21a], and RMM [LSS21b], tend to provide worse performance in TFS than TFH, while methods with weak KD losses, e.g., iCaRL [RKSL17] and LwF [LH18], tend to provide worse performance in TFH than TFS. Our method uses an online learning algorithm to produce the key hyperparameters, e.g., KD loss weights that control which KD losses are used. Thus, our method achieves the highest performance in both TFS and TFH.

known as *stability*. However, there is a fundamental *tradeoff* between plasticity and stability: too much plasticity may result in the *catastrophic forgetting* of old concepts, while too much stability restricts the ability to adapt to new concepts [MC89, MH93, Rat90]. To encourage related research, [RKSL17] defined the class-incremental learning (CIL) protocol, where the training samples of different classes come to the model phase-by-phase and most of the past data are removed from the memory.

Recently, many methods have been proposed to balance the stability-plasticity tradeoff for *different data-receiving settings* of CIL. For example, the strong feature knowledge distillation (KD) loss function is usually adopted when a large amount of data is available beforehand (e.g., the *training-from-half* (TFH) setting) since it encourages stability [HPL+19, LSS21a, LSS21b]; while the weak logit KD loss function is popular when data and classes are received evenly in each phase (e.g., the *training-from-scratch* (TFS) setting) since it provides more plasticity [RKSL17, BP19, LH18].

Most CIL algorithms pre-fix the tradeoff balancing methods, usually according to which data-receiving setting will be used in the experiments. However, in real-world scenarios, it is difficult to anticipate how data will be received in the future. Hence, a pre-fixed method is no longer proper for balancing the stability and plasticity of the actual data stream, thus generating worse performance. This can also be validated in Figure 6.1. Notice that the methods with weak KD, e.g., iCaRL [RKSL17] and LwF [LH18], provide worse performance in TFH than in TFS since weak KD provides too much plasticity for TFH, while the methods with strong KD, e.g., LUCIR [HPL+19], AANets [LSS21a], and RMM [LSS21b], perform worse in TFS than in TFH due to too much stability.

Therefore, a natural question is: how to design an *adaptive trade-off balancing method* to achieve good performance *without knowing* how data will be received *beforehand*? To tackle this, we propose an online-learning-inspired method to adaptively adjust key hyperparameters that affect the trade-off balancing performance in CIL. In our method, we introduce hyperparameters to control the choice of KD loss functions, learning rates, and classifier types, which are key algorithm choices that affect the tradeoff balancing performance.[1] In this way, deciding this choice is transformed into a hyperparameter optimization (HO) problem.

This HO problem cannot be directly solved because future data are not available. Thus, we borrow ideas from online learning, which is a widely adopted approach to adaptively tune the decisions without knowing the future data a priori while still achieving good performance in hindsight.

However, in CIL, our decisions affect not only the next phase but also all the future phases, which is different from the standard online learning setting [And08]. To capture the dependence across phases, we formulate the HO problem in CIL as an online MDP, which is a generalized version of online learning. Further, we propose a new algorithm based on [EDKM09] to solve this online MDP problem. Our algorithm differs from the standard online MDP algorithm in [EDKM09] in two aspects:

- In CIL, we cannot directly observe the reward (i.e., validation accuracy) because the validation data is not accessible during training. To address this issue, we estimate the reward by rebuilding local training and validation sets during policy learning in each phase, and computing the estimated reward on the local validation sets.

- In CIL, we only have access to the model generated by the selected hyperparameters instead of the models generated by other hyperparameters. In other words, we only have bandit feedback instead of full feedback as assumed in [EDKM09]. To address this, we revise the algorithm in [EDKM09] by combining it with a classic bandit algorithm, Exp3 [ACBFS02].

Empirically, we find our method performs well consistently. We conduct extensive CIL experiments by plugging our method into three top-performing methods (LUCIR [HPL+19], AANets [LSS21a], and RMM [LSS21b]) and testing them on three benchmarks (i.e., CIFAR-100, ImageNet-Subset, and ImageNet-Full). Our results show the consistent improvements of the proposed method, e.g., boosting the average accuracy of TFH and TFS by 2.2 percentage points on ImageNet-Full, compared to the state-of-the-art [LSS21b].

Lastly, it is worth mentioning that our method can also be applied to optimize other key hyperparameters in CIL, e.g., memory allocation [LSS21b].

Summary of our contributions. Our contributions are three-fold: 1) an online MDP formulation that allows online updates of hyperparameters that affect the balance of the stability and plasticity in CIL; 2) an Exp3-based online MDP algorithm to generate adaptive hyperparameters using bandit and estimated feedback; 3) extensive comparisons and visualizations for our method in three CIL benchmarks, taking top-performing methods as baselines.

[1]The impact of KD losses has been discussed before. Learning rates naturally affect how fast the model learns new concepts. We also adjust the classifier type because empirical results [RKSL17, HPL+19] show that the nearest class mean (NCM) and fully-connected (FC) classifiers perform better under more plasticity and stability, respectively.

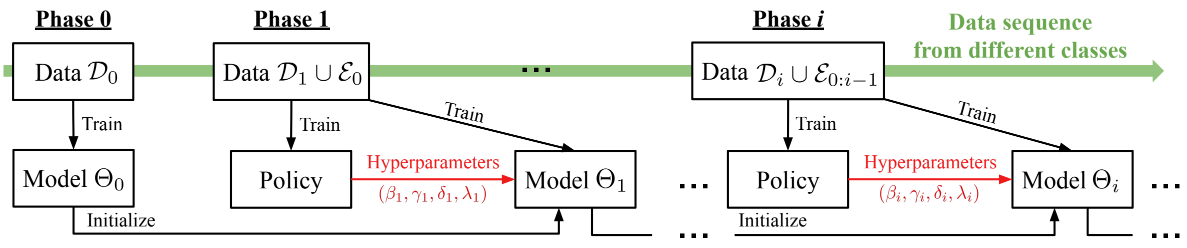

Phase 0
Phase 1
Phase i
Data sequence from different classes
Data $\mathcal{D}_0$
Data $\mathcal{D}_1 \cup \mathcal{E}_0$
Data $\mathcal{D}_i \cup \mathcal{E}_{0:i-1}$
Train
Model Θ_0
Policy
Hyperparameters
$(\beta_1, \gamma_1, \delta_1, \lambda_1)$
Model Θ_1
$(\beta_i, \gamma_i, \delta_i, \lambda_i)$
Model Θ_i
Initialize

Figure 6.2: The computing flow of our method. We formulate the CIL task as an online MDP: each phase in CIL is a stage in the MDP, and the CIL models are the states. We train the policy to produce actions, which contain the hyperparameters we use in the CIL training. We illustrate the training process of each phase in Figure 6.3.

6.2 RELATED WORK

In this section, we discuss related works on reinforcement learning, online learning, and hyperparameter optimization. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Reinforcement learning (RL) aims to learn a policy in an environment, which is typically formulated as an MDP. Some CIL papers also deploy RL algorithms in their frameworks. [XZ18] used RL to expand its backbone network when a new task arrives adaptively. [LSS21b] used RL to learn a policy to adjust the memory allocation between old and new class data dynamically along with the learning phases. Our method focuses on learning a policy to produce key hyperparameters. Besides, the existing methods need to solve the complete MDP, which is time-consuming. Here, we formulate the CIL task as an online MDP. Thus, our method is more time-efficient.

Online learning observes a stream of samples and makes a prediction for each element in the stream. There are mainly two settings in online learning: full feedback and bandit feedback. *Full feedback* means that the full reward function is given at each stage. It can be solved by Best-Expert algorithms [EDKM05]. *Bandit feedback* means that only the reward of the implemented decision is revealed. If the rewards are independently drawn from a fixed and unknown distribution, we may use e.g., Thompson sampling [AG12] and UCB [AO10] to solve it. If the rewards are generated in a non-stochastic version, we can solve it by e.g., Exp3 [ACBFS02]. *Online MDP* is an extension of online learning. Many studies [EDKM09, LZQL19] aim to solve it by converting it to online learning. In our case, we formulate the CIL as an online MDP and convert it into a classic online learning problem. The rewards in our MDP are non-stochastic because the training and validation data change in each phase. Therefore, we design our algorithm based on Exp3 [ACBFS02].

Hyperparameter optimization (HO). There are mainly two popular lines of HO methods: gradient-based and meta-learning-based. Gradient-based HO methods [BPRS18] make it possible to tune the entire weight vectors associated with a neural network layer as hyperparameters. Meta-learning-based HO methods [FFS$^+$18] use a bilevel program to optimize the hyperparameters. However, all these methods only consider time-invariant environments. Our online method learns hyperparameters that adapt to the time-varying environments in CIL.

6.3 METHODOLOGY

As illustrated in Figures 6.2 and 6.3, we formulate CIL as an online MDP and learn a policy to produce the hyperparameters in each phase. In this section, we first elaborate on the CIL denotations. Then, we introduce the online MDP formulation, show optimizable hyperparameters, and provide an online learning algorithm to train the policy.

6.3.1 Denotations for CIL

The general CIL pipeline is as follows. There are multiple phases during which the number of classes gradually increases to the maximum [DCO+20, HPL+19, HTM+21, LSL+20]. In the 0-th phase, we observe data $\mathcal{D}_0$, and use it to learn an initial model Θ_0. After this phase, we can only store a small subset of $\mathcal{D}_0$ (i.e., exemplars denoted as $\mathcal{E}_0$) in memory used as replay samples in later phases. In the i-th phase ($i \geq 1$), we get new class data $\mathcal{D}_i$ and load exemplars $\mathcal{E}_{0:i-1}=\mathcal{E}_0 \cup \cdots \cup \mathcal{E}_{i-1}$ from the memory. Then, we initialize Θ_i with Θ_{i-1}, and train it using $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. We evaluate the model Θ_i on a test set $\mathcal{Q}_{0:i}$ for all classes observed so far. After that, we select exemplars $\mathcal{E}_{0:i}$ from $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ and save them in the memory.

Existing work mainly focus on two data-receiving settings: *training-from-half* (TFH) [HPL+19] and *training-from-scratch* (TFS) [RKSL17] settings. In TFH, we are given half of all classes in the 0-th phase, and observe the remaining classes evenly in the subsequent N phases. In TFS, we are given the same number of classes in all N phases.

6.3.2 An Online MDP Formulation for CIL

Optimizing hyperparameters in CIL should be online inherently: training and validation data changes in each phase, so the hyperparameters should be adjusted accordingly. Thus, it is intuitive to formulate the CIL as an online MDP. In the following, we provide detailed formulations.

Stages. Each phase in the CIL task can be viewed as a stage in the online MDP.

States. The state should define the current situation of the intelligent agent. In CIL, we use the model Θ_i as the state of the i-th phase/stage. We use $\mathbb{S}$ to denote the state space.

Actions. We use a vector consisting of the hyperparameters in the i-th phase as the action $\mathbf{a}_i$. When we take the action $\mathbf{a}_i$, we deploy the corresponding hyperparameters. We denote the action space as $\mathbb{A}$. Please refer to the next subsection, *Optimizable Hyperparameters*, for more details.

Policy $\mathbf{p}=\{p(\mathbf{a}|\Theta_i)\}_{\mathbf{a}\in\mathbb{A}}$ is a probability distribution over the action space $\mathbb{A}$, given the current state Θ_i.

Environments. We define the training and validation data in each phase as the environment. In the i-th phase, the environment is $\mathcal{H}_i=(\mathcal{E}_{0:i-1} \cup \mathcal{D}_i, \mathcal{Q}_{0:i})$, where $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$ is the training data and $\mathcal{Q}_{0:i}$ is the corresponding validation data. The environment is time-varying because we are given different training and validation data in each phase.

Rewards. CIL aims to train a model that is efficient in recognizing all classes seen so far. Therefore, we use the validation accuracy as the reward in each phase. Our objective is to maximize a cumulative reward, i.e., $R = \sum_{i=1}^{N} r_{\mathcal{H}_i}(\Theta_i, \mathbf{a}_i)$, where $r_{\mathcal{H}_i}(\Theta_i, \mathbf{a}_i)$ denotes the i-th phase reward, i.e., the validation accuracy of Θ_i. The reward function $r_{\mathcal{H}_i}$ changes with $\mathcal{H}_i$, so it is time-varying.

6.3.3 Optimizable Hyperparameters

In this part, we introduce the optimizable hyperparameters, and how to define the actions and action space based on the hyperparameters. We consider three kinds of hyperparameters that significantly affect the stability and plasticity: 1) KD loss weights, 2) learning rates, and 3) classified types.

1) KD loss weights. We first introduce two KD losses (i.e., logit and feature KD losses) and then show how to use the KD loss weights to balance them.

Logit KD loss is proposed in [HVD15] and widely applied in CIL methods [LH18, RKSL17, LSS21a]. Its motivation is to make the current model Θ_i mimic the prediction logits of the old model Θ_{i-1}:

$$\mathcal{L}_{\text{logi}} = -\sum_{k=1}^{K} \eta_k(\mu(x;\Theta_{i-1})) \log \eta_k(\mu(x;\Theta_i)), \tag{6.1}$$

where $\mu(x;\Theta)$ is a function that maps the input mini-batch (x, y) to the prediction logits using the model Θ. $\eta_k(v) = v_k^{1/\tau} / \sum_j v_j^{1/\tau}$ is a re-scaling function for the k-th class prediction logit and τ is a scalar set to be greater than 1.

Feature KD loss [HPL$^+$19, DCO$^+$20] aims to enforce a stronger constraint on the previous knowledge by minimizing the cosine similarity between the features from the current model Θ_i and the old model Θ_{i-1}. It can be computed as follows,

$$\mathcal{L}_{\text{feat}} = 1 - S_c(f(x;\Theta_i), f(x;\Theta_{i-1})), \tag{6.2}$$

where $f(x;\Theta)$ denotes a function that maps the input image x to the features using the model Θ. $S_c(v_1, v_2)$ denotes the cosine similarity between v_1 and v_2.

The overall loss in the i-th phase is a weighted sum of the classification and different KD losses:

$$\mathcal{L}_{\text{overall}} = \mathcal{L}_{\text{CE}} + \beta_i \mathcal{L}_{\text{logi}} + \gamma_i \mathcal{L}_{\text{feat}}, \tag{6.3}$$

where β_i and γ_i are the weights of the logit and feature KD losses, respectively. $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy classification loss. Existing methods [LH18, RKSL17, HPL$^+$19, LSS21a, LSS21b] can be viewed as using fixed heuristic KD loss weights, e.g., $\beta_i \equiv 1$ and $\gamma_i \equiv 0$ in iCaRL [LH18, RKSL17]. Instead, our method optimizes β_i and γ_i online. Thus, we can balance the model's stability and plasticity by adjusting β_i and γ_i. We apply different weights for the logit and feature KD losses so that we can achieve fine-grained control over the intensity of knowledge distillation.

2) Learning rate is another important hyperparameter that affects the model's stability and plasticity. We empirically find that a lower learning rate makes the CIL model more stable, while a higher learning rate makes the CIL model more plastic. If we use λ_i to denote the learnable learning rate in the i-th phase, we update the CIL model as follows,

$$\Theta_i \leftarrow \Theta_i - \lambda_i \nabla_{\Theta_i} \mathcal{L}_{\text{overall}}. \tag{6.4}$$

Another hyperparameter, the number of training epochs, has similar properties to the learning rate. We choose to optimize the learning rate and fix the number of epochs because it empirically works better with our online learning algorithm.

3) Classifier type. Motivated by the empirical analysis, we consider two classifier types in our study: *nearest class mean* (NCM) [RKSL17, SSZ17] and *fully-connected* (FC) [HPL$^+$19, LSL$^+$20] classifiers. For the NCM classifier, we first compute the mean feature for each class using the

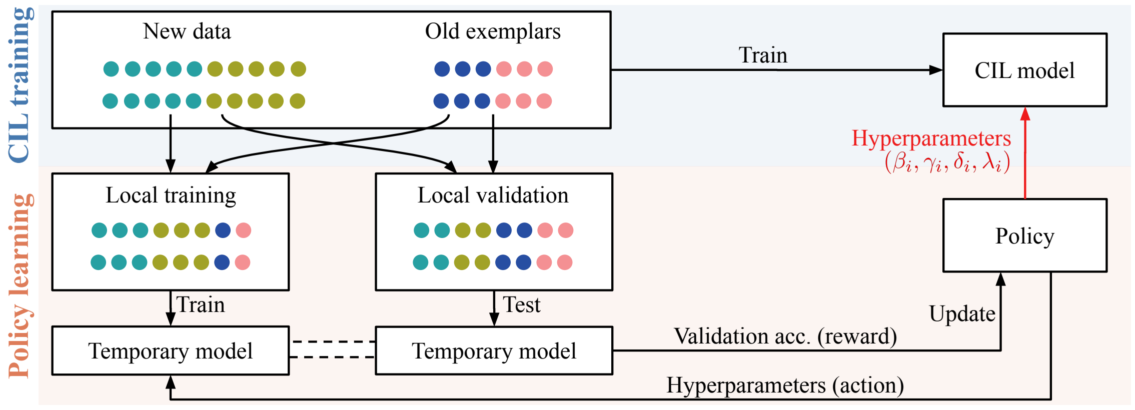


Figure 6.3: The training process of our online learning method in the i-th phase. It includes *policy learning* and *CIL training*. **(a) Policy learning**. 1) We construct a class-balanced subset from all training data as the local validation set and use the remaining data as the local training set. 2) We initialize the temporary model with Θ_{i-1}. 3) We sample an action using the current policy and deploy the hyperparameters on the temporary model according to the action. 4) We train it on the local training set for M_1 epochs, and evaluate it on the local test set. 5) We use the validation accuracy as the reward and update the policy. We update the policy for T iterations by repeating Steps 2-5. **(b) CIL training**. We sample an action using the learned policy and deploy the hyperparameters on the CIL model. Then, we train the CIL model on all training data for M_2 epochs. We set $M_1 = \lceil 0.1 M_2 \rceil$ to speed up the policy learning.

new data and old exemplars. Then we perform a nearest neighbor search using the Euclidean distance on the L_2 normalized mean features to get the final predictions. It is observed empirically that the NCM classifier tends to work better on the models with high plasticity, while the FC classifier performs better on the models with high stability [RKSL17, HPL$^+$19]. Thus, we propose to use a hyperparameter, classifier type indicator δ_i, to control the final predictions during the evaluation:

$$\mu(x;\Theta_i) = \mu_{\text{ncm}}(x;\Theta_i)[\delta_i = 1] + \mu_{\text{fc}}(x;\Theta_i)[\delta_i = 0], \tag{6.5}$$

where $\delta_i \in \{0,1\}$, μ_{ncm} and μ_{fc} are the predictions on the input image x using the NCM and FC classifiers, respectively.

Summary: actions and action space. In summary, we define the action as $\mathbf{a}_i{=}(\beta_i, \gamma_i, \lambda_i, \delta_i,)$, which consists of the following hyperparameters: KD loss weights β_i and γ_i, learning rate λ_i, and classifier type indicator δ_i. For the hyperparameters that may vary in a continuous range, we *discretize* them to define a *finite* action space.[2] In the next subsection, we show how to learn the policy in each phase.

6.3.4 Policy Learning

A common approach to solving an online MDP is to approximate it as an online learning problem and solve it using online learning algorithms [EDKM05, AG12, ACBFS02]. We also

[2]Though discretization suffers the curse of dimensionality, our experiments show that with a coarse grid, we already have significant improvements over pre-fixed hyperparameters.

take this approach, and our approximation follows [EDKM09], which achieves the optimal regret. In their paper, [EDKM09] relax the Markovian assumption of the MDP by decoupling the cumulative reward function and letting it be time-dependent so that they can solve the online MDP by standard online learning algorithms. Such a decoupling requires the following assumptions. 1) Fast mixing: in CIL, the hyperparameters in an early phase do not have much impact on the test accuracy of the classes observed in the current phase. 2) The algorithm changes the hyperparameters slowly (this can be observed in ***Experiments*** & Figure 6.5). Thus, these assumptions fit our CIL problem.

However, we cannot directly apply the algorithms proposed in [EDKM09] to our problem. It is because their paper assumes *full feedback*, i.e., we can observe the rewards of all actions in each phase. Therefore, its online learning problem could be solved by Best Expert algorithms [EDKM05]. In CIL, we cannot observe any reward (i.e., validation accuracy) because the validation data is not accessible during training. To address this issue, we rebuild the local training and validation sets in each phase. In this way, our problem has *bandit feedback*: we can compute the reward of the implemented action. Therefore, we can solve our online learning problem based on Exp3 [ACBFS02], a famous bandit algorithm.

In the following, we show how to rebuild the local training and validation sets, compute the decoupled cumulative reward, and learn the policy with Exp3.

Rebuilding local datasets. In the i-th phase, we need to access the validation set $\mathcal{Q}_{0:i}$ to compute the reward (i.e., the validation accuracy). However, we are not allowed to use $\mathcal{Q}_{0:i}$ during training because it violates the CIL benchmark protocol. Therefore, we replace $\mathcal{Q}_{0:i}$ with a class-balanced subset $\mathcal{B}_{0:i}$ sampled from the training data $\mathcal{E}_{0:i-1} \cup \mathcal{D}_i$. $\mathcal{B}_{0:i}$ contains the same number of samples for both the old and new classes. In this way, we can rebuild the local training and validation sets, and obtain the local environment $h_i = ((\mathcal{E}_{0:i-1} \cup \mathcal{D}_i) \setminus \mathcal{B}_{0:i}, \mathcal{B}_{0:i})$.

Decoupled cumulative reward. We create the decoupled cumulative reward function $\hat{R}$ based on the original cumulative reward function $R = \sum_{j=1}^{N} r_{\mathcal{H}_j}(\Theta_j, \mathbf{a}_j)$. In the i-th phase, we compute $\hat{R}$ as follows,

$$\hat{R}(\mathbf{a}_i, h_i) = \underbrace{\sum_{j=1}^{i-1} r_{\mathcal{H}_j}(\Theta_j, \mathbf{a}_j)}_{\text{Part I}} + \underbrace{\sum_{j=i}^{i+n} r_{h_i}(\Theta_j, \mathbf{a}_i)}_{\text{Part II}}, \tag{6.6}$$

where Part I is the historical rewards from the 1-st phase to the (i-1)-th phase. It is a constant and doesn't influence policy optimization. Part II is the long-term reward of a time-invariant local MDP based on the local environment h_i. We use Part II as an estimation of the future rewards, following [EDKM09]. Because we don't know the total number of phases N during training, we assume there are n phases in the future. Furthermore, we fix the action $\mathbf{a}_i$ in Part II to simplify the training process. Thus, $\hat{R}$ can be reviewed as a function of $\mathbf{a}_i$ and h_i.

Training policy with Exp3. Exp3 [ACBFS02] introduces an auxiliary variable $\mathbf{w} = \{w(\mathbf{a})\}_{\mathbf{a}\in\mathbb{A}}$. After updating $\mathbf{w}$, we can determine the policy $\mathbf{p}=\{p(\mathbf{a}|\Theta_i)\}_{\mathbf{a}\in\mathbb{A}}$ by

$$\mathbf{p} = \mathbf{w}/||\mathbf{w}||, \tag{6.7}$$

The updating rule of w is provided below. In the 1-st phase, we initialize $\mathbf{w}$ as $\{1, \dots, 1\}$. In each phase i ($i \geq 1$), we update $\mathbf{w}$ for T iterations. In the t-th iteration, we sample an action $\mathbf{a}_t \sim \mathbf{p}$, apply the action $\mathbf{a}_t$ to the CIL system, and compute the decoupled cumulative reward

$\hat{R}(\mathbf{a}_t, h_i)$ using Eq. 6.6. After that, we update $w(\mathbf{a}_t)$ in $\mathbf{w}$ as,

$$w(\mathbf{a}_t) \leftarrow w(\mathbf{a}_t) \exp(\xi \hat{R}(\mathbf{a}_t, h_i) / p(\mathbf{a}_t | \Theta_i)), \tag{6.8}$$

where ξ is a constant, which can be regarded as the learning rate in Exp3.

Methods	**CIFAR-100**, N=5			**CIFAR-100**, N=25		
	TFH	TFS	Avg.	TFH	TFS	Avg.
LwF [LH18]	52.3	58.8	55.6	45.6	48.3	47.9
iCaRL [RKSL17]	58.1	64.0	61.0	48.1	53.2	50.7
PODNet [DCO+20]	64.7	63.6	64.2	60.3	45.3	52.8
DER [YXH21]	67.6	72.3	70.0	65.5	67.3	66.4
FOSTER [WZYZ22]	70.4	72.5	71.5	63.8	**70.7**	67.3
LUCIR [HPL+19]	63.1±0.7	63.0±0.6	63.1±0.7	57.5±0.4	49.2±0.5	53.4±0.5
w/ ours	63.9±0.6	64.9±0.5	64.4±0.6	59.3±0.5	52.4±0.5	55.9±0.5
	↑0.8	↑1.9	↑1.3	↑1.8	↑3.2	↑2.5
AANets [LSS21a]	65.3±0.4	63.1±0.3	64.2±0.4	63.2±0.3	44.4±0.4	53.8±0.4
w/ ours	67.0±0.3	65.1±0.3	66.1±0.3	64.1±0.4	50.3±0.5	57.2±0.5
	↑1.7	↑2.0	↑1.9	↑0.9	↑5.9	↑3.4
RMM [LSS21b]	67.6±0.7	70.4±0.8	69.0±0.8	65.6±0.6	58.4±0.6	62.0±0.6
w/ ours	**70.8**±0.7	**72.7**±0.6	**71.8**±0.7	**69.5**±0.8	65.9±0.7	**67.7**±0.8
	↑3.2	↑2.3	↑2.8	↑3.9	↑7.5	↑5.7

Methods	**ImageNet-Subset**, N=5			**ImageNet-Subset**, N=25		
	TFH	TFS	Avg.	TFH	TFS	Avg.
LwF [LH18]	55.1	62.0	58.6	44.3	43.9	44.1
iCaRL [RKSL17]	65.3	70.4	67.9	53.0	53.5	53.3
PODNet [DCO+20]	64.3	58.9	61.6	68.3	39.1	53.7
DER [YXH21]	78.4	76.9	77.7	75.4	71.0	73.2
FOSTER [WZYZ22]	80.2	78.3	79.3	69.3	72.9	71.1
LUCIR [HPL+19]	65.3±0.6	66.7±0.5	66.0±0.6	61.4±0.7	46.2±0.8	53.8±0.8
w/ ours	70.6±0.7	68.4±0.6	69.5±0.7	62.9±0.6	54.1±0.6	58.5±0.6
	↑5.3	↑1.7	↑3.5	↑1.5	↑7.9	↑4.7
AANets [LSS21a]	77.0±0.7	68.9±0.6	73.0±0.7	72.2±0.6	60.7±0.5	66.5±0.6
w/ ours	77.3±0.6	70.6±0.5	74.0±0.6	72.9±0.5	64.8±0.5	68.9±0.5
	↑0.3	↑1.7	↑1.0	↑0.7	↑4.1	↑2.4
RMM [LSS21b]	79.5±0.2	80.5±0.3	80.0±0.3	75.0±0.3	71.6±0.3	73.3±0.3
w/ ours	**81.0**±0.3	**82.2**±0.4	**81.6**±0.4	**76.1**±0.2	**73.2**±0.4	**74.7**±0.3
	↑1.5	↑1.7	↑1.6	↑1.1	↑1.6	↑1.4

Table 6.1: Average accuracy (%) across all phases on CIFAR-100 and ImageNet-Subset. The first block shows some recent CIL methods. The second block shows three top-performing baselines [HPL+19, LSS21a, LSS21b] *w/* and *w/o* our method plugged in. "TFH" and "TFS" denote the *training-from-half* and *training-from-scratch* settings, respectively. "Avg." shows the average of the "TFH" and "TFS" results. For "AANets" [LSS21a], we use its version based on PODNet [DCO+20]. We rerun the baselines [LH18, RKSL17, DCO+20, YXH21, WZYZ22] using their open-source code in a unified setting for a fair comparison.

6.4 EXPERIMENTS

We evaluate the proposed method on three CIL benchmarks: CIFAR-100 [KH+09], ImageNet-Subset [RKSL17], and ImageNet-Full [RDS+15]. We incorporate our method into three top-performing baseline methods (LUICR [HPL+19], AANets [LSS21a], and RMM [LSS21b]) and boost their performances consistently in all settings. Below we describe the datasets and implementation details, followed by the results and analyses, which include extensive comparisons to related work, ablation results, and visualizations.

6.4.1 Datasets and Implementation Details

Datasets. We employ CIFAR-100 [KH+09], ImageNet-Subset [RKSL17] (100 classes), and ImageNet-Full [RDS+15] (1000 classes) as the benchmarks. We use the same data splits and class orders as the related work [RKSL17, CMG+18, LSS21a, LSS21b] for a fair comparison.

Network architectures. We use a modified 32-layer ResNet [RKSL17] for CIFAR-100 and an 18-layer ResNet [HZRS16] for ImageNet, following [RKSL17, HPL+19, LSS21a, LSS21b, HTM+21]. We deploy the AANets [LSS21a] for the experiments based on AANets and RMM [LSS21b]. Further, we use a cosine normalized classifier without bias terms as the FC classifier, following [HPL+19, LSL+20].

Configurations. We discretize the hyperparameter search space into 50 actions, i.e., $\text{card}(\mathbb{A})$=50. We update the policy for 25 iterations in each phase, i.e., T=25. For other configurations, we follow the corresponding baselines.

Benchmark protocol. We run experiments in both TFH and TFS settings. 1) *Training-from-half* (TFH) [HPL+19, LSL+20, DCO+20, LSS21a]. It means the data of the half dataset are used to train the 0-th phase model. The rest classes come to the model evenly in subsequent N phases. 2) *Training-from-scratch* [RKSL17, ZXG+20, WCW+19] (TFS). It divides all classes evenly for N phases, i.e., the same number of classes coming to the model in each phase. Compared to TFH, this setting has poorer models in earlier phases. We have two options for the number of total phases, i.e., N=5/25. We report the average accuracy across all phases. We run each experiment three times and report the average results with 95% confidence intervals.

6.4.2 Results and Analyses

Tables 6.1 and 6.2 present the results of top-performing baselines *w/* and *w/o* our method and some recent related work. Table 3.2 summarizes the results in seven ablative settings.

Methods	**ImageNet-Full**, N=5		
	TFH	TFS	Avg.
LUCIR [HPL+19]	$64.5_{\pm 0.3}$	$62.7_{\pm 0.4}$	$62.0_{\pm 0.4}$
w/ ours	$65.8_{\pm 0.3}$ $\uparrow$1.3	$66.1_{\pm 0.3}$ $\uparrow$3.4	$66.0_{\pm 0.3}$ $\uparrow$2.4
RMM [LSS21b]	$69.0_{\pm 0.5}$	$66.1_{\pm 0.4}$	$67.6_{\pm 0.5}$
w/ ours	**70.7**$_{\pm 0.5}$ $\uparrow$1.7	**68.9**$_{\pm 0.5}$ $\uparrow$2.8	**69.8**$_{\pm 0.5}$ $\uparrow$2.2

Table 6.2: Average accuracy (%) on ImageNet-Full.

No.	Optimizing			N=5		N=25	
	(β, γ)	δ	λ	TFH	TFS	TFH	TFS
1	Baseline			63.11	62.96	57.47	49.16
2	✓			63.20	63.60	58.27	50.91
3	✓	✓		63.23	64.08	58.20	51.94
4	✓	✓	✓	**63.88**	**64.92**	**59.27**	**52.44**
5	Cross-val fixed			63.33	64.02	57.50	51.64
6	Offline RL [LSS21b]			63.42	63.88	58.12	51.53
7	Bilevel HO [FFS+18]			63.20	63.02	57.56	49.42

Table 6.3: Ablation results (average accuracy %) on CIFAR-100. (β, γ) are KD loss weights. λ and δ denote learning rates and classifier types, respectively. The baseline is LUCIR [HPL+19]. Row 4 shows our best result.

Figure 6.4 compares the activation maps (using Grad-CAM [SCD+17]) produced by diffident methods in TFH and TFS. Figure 6.5 shows the values of hyperparameters produced by our method.

Comparison with the state-of-the-art. Tables 6.1 and 6.2 show that taking our method as a plug-in module for the state-of-the-art [LSS21b] and other baselines [HPL+19, LSS21a] consistently improves their performance. For example, RMM [LSS21b] *w/* ours gains 4.3 and 2.2 percentage points on CIFAR-100 and ImageNet-Full, respectively. Interestingly, we find that we can surpass the baselines more when the number of phases N is larger. E.g., on CIFAR-100, our method improves RMM by 5.7 percentage points when N=25, while this number is 2.8 percentage points when N=5. Our explanation is that the forgetting problem is more serious when the number of phases is larger. Thus, we need better hyperparameters to balance the stability and plasticity.

Ablation study. Table 3.2 concerns eight ablative settings, and shows the results in both TFH and TFS for different numbers of phases (N=5/25). The detailed analyses are as follows.

1) First block. Row 1 shows the baseline [HPL+19].

2) Second block: optimizable hyperparameters. In our study, we optimize three kinds of hyperparameters that affect the model's stability and plasticity: KD loss weights (β, γ), learning rate λ, and classifier type indicator δ. Comparing Row 2 to Row 1, we can observe that optimizing the KD loss weights boosts the TFS accuracy more significantly. It is because the baseline, LUCIR [HPL+19], applies a strong regularization term (i.e., feature KD loss) which harms the TFH performance. Our method changes the regularization term by adjusting the KD loss weights, so it achieves better performance. Comparing Row 3 to Row 2, we can see that optimizing the classifier type indicator δ performs further improvements, especially in TFS. It is because the baseline deploys an FC classifier by default while our method learns to switch between different classifiers in different settings. Comparing Row 4 to Row 3, we can see the effectiveness of optimizing the learning rates in CIL. In summary, it is impressive that optimizing three kinds of hyperparameters together achieves the best results.

3) Third block: hyperparameter learning methods. For Row 5, we use cross-validation (i.e., all past, future, and validation data are accessible) to find a set of fixed hyperparameters and apply them to all phases. We can see that Row 5 results are consistently lower than

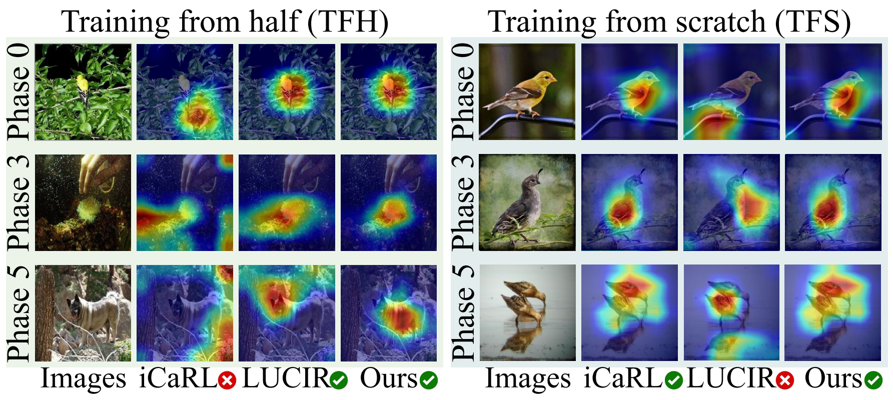


Figure 6.4: The activation maps using Grad-CAM [SCD+17] for the last phase model on ImageNet-Subset 5-phase. Samples are selected from the classes coming in the 0-th, 3-rd, and 5-th phases. Green ticks mean successful activation of discriminative features on object regions, while red crosses mean unsuccessful.

ours in Row 4, although it can access more data. It shows that we need to update the hyperparameters online in different phases. For Row 6, we use the policy pre-trained in the 0-th phase of the target CIL task using the framework proposed in [LSS21b] (compared to ours, it is offline). Comparing Row 6 with Row 4, we are happy to see that our online learning algorithm achieves better performance than the offline RL while we use much less training time. For Row 7, we use the bilevel hyperparameter optimization method [FFS+18]. Comparing Row 7 with Row 4, we can observe that our method achieves more significant performance improvements. The reason is that [FFS+18] is designed for time-invariant environments, while our online algorithm can adapt to the time-varying environments in CIL.

Visualizing activation maps. Figure 6.4 demonstrates the activation maps visualized by Grad-CAM [SCD+17] for the final model (obtained after five phases) on ImageNet-Subset 5-phase. The left and right sub-figures show the results for *training-from-half* (TFH) and *training-from-scratch* (TFS), respectively. We can observe: 1) LUCIR [HPL+19] makes predictions according to foreground (correct) and background (incorrect) regions in the TFH and TFS settings, respectively; 2) iCaRL [RKSL17] behaves opposite to LUCIR in the two settings; 3) our method always makes predictions according to foreground (correct) regions in both TFH and TFS. The reasons are as follows. LUCIR applies a strong (feature) KD by default, so it performs better in TFH. iCaRL applies a weak (logit) KD by default, so it performs better in TFS. Our method can adjust the hyperparameters to change between different KD losses, so it performs well in both settings.

Hyperparameter values. Figure 6.5 shows the hyperparameter values produced by our policy on CIFAR-100 25-phase.

1) KD loss weights β and γ. From the figure, we have two observations. a) The policy learns to produce a larger initial value for γ and β in TFH and TFS, respectively. Our explanation is as follows. In TFH, we already have a pre-trained model, so we need a strong regularization term (i.e., feature KD loss) to make the model more stable and avoid forgetting. In TFS, we start from random initialization, so we need a weak regularization term (i.e., logit KD loss) to improve the model's plasticity. b) Both β and γ increase in TFH, while β decreases and γ increases in TFS. It is because we need stronger regularization to maintain the knowledge

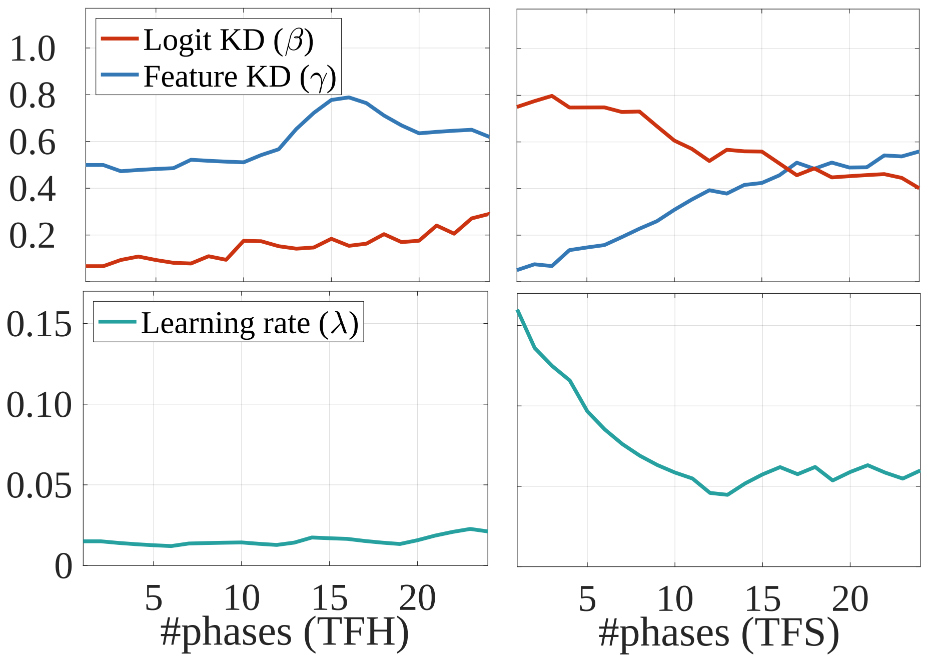


Figure 6.5: The hyperparameter values produced by our policy on CIFAR-100 25-phase for LUCIR [HPL+19] *w/* ours. We smooth all curves with a rate of 0.8 for better visualization.

when more data is observed. The policy achieves that in different ways: it assigns higher weights for both KD losses in TFH, while it transfers from logit KD to feature KD in TFS.
2) Learning rates λ. In Figure 6.5, we can observe that the learning rate in TFS is much higher than in TFH. It can be explained that 1) we need a higher learning rate in TFS as the model is trained from scratch and needs to capture more new knowledge; 2) we need a lower learning rate in TFH because we need to avoid forgetting the pre-trained model.
3) Classifier type indicator δ. On CIFAR-100 25-phase, our policy learned to choose the NCM and FC classifiers in TFS and TFH, respectively.

6.5 CONCLUSIONS

In this study, we introduce a novel framework that allows us to optimize hyperparameters online to balance the stability and plasticity in CIL. To achieve this, we formulate the CIL task as an online MDP and learn a policy to produce the hyperparameters. Our approach is generic, and it can be easily applied to existing methods to achieve large-margin improvements in both TFS and TFH settings. It is worth mentioning that our method can also be applied to optimize other key hyperparameters in CIL.

II

Few-shot Learning: Learning with Limited Data without Overfitting

While the previous parts focused on incremental learning, this part considers another learning scenario with imperfect data, few-shot learning for image classification. For this task, we study how to improve the generalization ability of the model and tackle the overfitting problem by learning to transfer knowledge and ensemble deep models. Specifically,

Chapter 7 tackles the over-fitting issue in few-shot learning by proposing a novel method called meta-transfer learning (MTL), which learns to adapt a deep neural network for few-shot learning tasks.

In Chapter 8, we improve the robustness of few-shot learning by meta-learning the ensemble of epoch-wise empirical Bayes models. *Epoch-wise* means that each training epoch has a Bayes model whose parameters are specifically learned and deployed. *Empirical* means that the hyperparameters, e.g., used for learning and ensembling the epoch-wise models, are generated by hyperprior learners conditional on task-specific data.

7 LEARNING TO TRANSFER KNOWLEDGE

Contents

Meta-learning has been proposed as a framework to address the challenging few-shot learning setting. The key idea is to leverage a large number of similar few-shot tasks in order to learn how to adapt a base-learner to a new task for which only a few labeled samples are available. As deep neural networks (DNNs) tend to overfit using a few samples only, typical meta-learning models use shallow neural networks, thus limiting their effectiveness. In order to achieve top performance, some recent works tried to use the DNNs pre-trained on large-scale datasets but mostly in straight-forward manners, e.g., (1) taking their weights as a warm start of meta-training, and (2) freezing their convolutional layers as the feature extractor of base-learners. In this chapter, we propose a novel approach called **meta-transfer learning (MTL)**, which learns to transfer the weights of a deep NN for few-shot learning tasks. Specifically, *meta* refers to training multiple tasks, and *transfer* is achieved by learning the scaling and shifting functions of DNN weights (and biases) for each task. To further boost the learning efficiency of MTL, we introduce the **hard task (HT) meta-batch** scheme as an effective learning curriculum of few-shot classification tasks. We conduct experiments for five-class few-shot classification tasks on three challenging benchmarks, *mini*ImageNet, *tiered*ImageNet, and Fewshot-CIFAR100 (FC100), in both supervised and semi-supervised settings. Extensive comparisons to related works validate that our MTL approach trained with the proposed HT meta-batch scheme achieves top performance. An ablation study also shows that both components contribute to fast convergence and high accuracy.

This chapter is based on [SLC+22]. As the co-first author of [LSL+20], Yaoyao Liu conducted most experiments and was the main writer. Figure 7.8 and the corresponding results were contributed by Zhaozheng Chen from Singapore Management University. The conference version of this paper, [SLCS19], is one of the Top 200 most cited CVPR papers in five years and has been cited more than 700 times. This work also inspired several real-world applications, e.g., super-resolution [SCC20] and COVID-19 Detection [SB20]. Furthermore, this work is reproduced in [LDT+21] and has shown to be one of the most effective few-shot learning baselines with the highest cross-domain transferability.

7.1 INTRODUCTION

Although deep learning systems have achieved great performance when sufficient amounts of labeled data are available [YYG15, HZRS16, SLD17], there has been growing interest in reducing the required amount of data. Few-shot learning tasks have been defined for this purpose. The aim is to learn new concepts from a handful of training examples, e.g., from 1 or 5 training images [LFP06, FAL17, SLCS19]. Humans tend to be highly effective in this context, often grasping the essential connection between new concepts and their own knowledge, but it remains challenging for machine learning models. For instance, on the CIFAR-100 dataset, a classification model trained in the fully supervised mode achieves 76% accuracy for the 100-class setting [CUH16], while the best-performing 1-shot model achieves only 45% in average for the much simpler 5-class setting [SLCS19]. On the other hand, in many real-world applications, we lack large-scale training data, as e.g., in the medical domain. It is thus desirable to improve machine learning models in order to handle few-shot settings.

Basically, the nature of few-shot learning with very scarce training data makes it difficult to train powerful machine learning models for new concepts. People explore a variety of methods in order to overcome this. A straightforward idea is to increase the amount of available data by data augmentation techniques [KBI+17]. Several methods were proposed to learn a data generator e.g. conditioned on Gaussian noises [MD17, SKS+18, WGHH18] or object attributes [XSSA19]. However, this data generator often under-performs when trained on few-shot data, which has been investigated by [BV18]. An alternative is to merge data from multiple tasks, which, however, is often ineffective due to high variances of the data across tasks [WGHH18].

In contrast to data augmentation methods, meta-learning is a task-level optimization-based method [BBCG92, NM92, TP98]. It aims to transfer experience from similar few-shot learning tasks [FAL17, FXL18, GFL+18, FFS+18, LC18, ZCG+18, SLCS19, AES19, LMRS19, HMX+20]. Related methods follow a unified training process that contains two loops. The inner-loop learns a base-learner for an individual task, and the outer-loop then uses the validation performance of the learned base-learner to optimize the meta-learner. A state-of-the-art representative method named Model-Agnostic Meta-Learning (MAML) learns to search for the optimal initialization state to fast adapt a base-learner to a new task [FAL17]. Its task-agnostic property makes it possible to generalize to few-shot supervised/semi-supervised learning as well as unsupervised reinforcement learning [FAL17, GFL+18, FXL18, AES19, ZCG+18, RRS+19, RTR+18, LSL+19]. However, in our view, there are two main limitations of this type of approach limiting their effectiveness: i) these methods usually require a large number of similar tasks for meta-training, which is costly; and ii) each task is typically modeled by a low-complexity base-learner, such as a shallow neural network (SNN), to avoid model overfitting to few-shot training data, thus being unable to deploy the deeper and more powerful architectures. For example, for the *mini*ImageNet dataset [VBL$^+$16], MAML uses a *shallow* CNN with only 4 CONV layers and its optimal performance was obtained by learning on 240k tasks (60k iterations in total and each meta-batch contains 4 tasks).

In this chapter, we propose a novel meta-learning method called **meta-transfer learning (MTL)** leveraging the advantages of both transfer learning and meta-learning. In a nutshell, MTL is a novel learning method that helps deeper neural networks based base-learners converge faster while reducing their probability to overfit when training on a few labeled data only. In particular, "transfer" means that DNN weights trained on large-scale data can be used in other tasks by two light-weight neuron operations: *Scaling* and *Shifting* (*SS*), i.e. $\alpha X + \beta$. "Meta" means that the parameters of these *SS* operations can be viewed as hyper-parameters learned with few-shot learning tasks [MY17, LZCL17, FFS$^+$18]. First, large-scale trained DNN weights offer a good initialization, enabling fast convergence of MTL with fewer tasks, e.g., only 8k tasks for *mini*ImageNet [VBL$^+$16], 30 times fewer than MAML [FAL17]. Second, light-weight operations on DNN neurons have less parameters to learn, e.g., less than $\frac{2}{49}$ if considering neurons of size 7×7 ($\frac{1}{49}$ for α and $< \frac{1}{49}$ for β), reducing the chance of overfitting to few-shot data. Third, these operations keep those trained DNN weights unchanged and thus avoid the problem of "catastrophic forgetting" which means forgetting general patterns when adapting to a specific task [LR17, MC89]. Finally, these operations are conducted on the convolutional layers mostly working for image feature extraction, thus can generalize well to a variety of few-shot learning models, e.g., MAML [FAL17], MatchingNet [VBL$^+$16], ProtoNet [SSZ17], RelationNet [SYZ$^+$18] and SIB [HMX$^+$20].

The second main contribution of this chapter is an effective meta-training curriculum. Curriculum learning [BLCW09] and hard negative mining [SGG16] both suggest that faster convergence and stronger performance can be achieved by better arrangements of training data, i.e., the few-shot training tasks in our case. Inspired by these ideas, we design our **hard task (HT) meta-batch** strategy to offer a challenging but effective learning curriculum. The conventional meta-batch contains a number of random tasks [FAL17], but our HT meta-batch online re-samples harder ones according to past failure tasks with the lowest validation accuracy. In addition, we add the meta-gradient regularization on each task that each task is optimized by using the weighted sum of meta-gradients of both current and previous tasks. The aim is to force the meta-learner not to forget old knowledge in afterward learning.

Our overall contribution is thus three-fold: i) we propose a novel **MTL** method that learns to transfer large-scale pre-trained DNN weights for solving few-shot learning tasks; ii) we propose a novel **HT meta-batch** learning strategy that forces meta-transfer to "grow faster and stronger through hardship"; and iii) we conduct extensive experiments on three few-shot learning benchmarks, namely *mini*ImageNet [VBL$^+$16], *tiered*ImageNet [RTR$^+$18] and Fewshot-CIFAR100 (FC100) [ORL18], and achieve the state-of-the-art performance on both supervised and semi-supervised few-shot learning.

7.2 RELATED WORK

We borrow the idea of transfer learning when leveraging the large-scale pre-training step in prior to meta-transfer. For task sampling, our HT meta-batch scheme is related to curriculum learning and hard negative sampling methods.

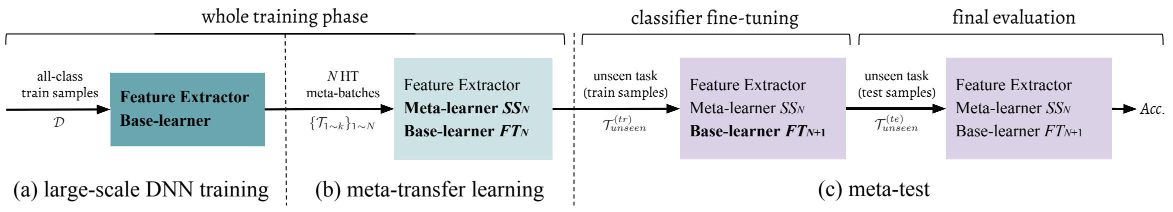


Figure 7.1: The pipeline of our proposed few-shot learning method, including: (a) DNN pre-training on large-scale data, i.e. using the entire training dataset; and (b) meta-transfer learning (MTL) that learns the parameters of *Scaling* and *Shifting* (*SS*), on the basis of pre-trained feature extractor (Section 7.3.2). The learning process is scheduled by the proposed HT meta-batch (Section 7.3.3) and regularized by meta-gradient regularization (Section 7.3.4). In (c), it is meta-test on unseen task whose processing consists of a base-learner (classifier) *Fine-Tuning* (*FT*) stage and a final evaluation stage, described in the last paragraph in Section 7.3.1. Input data are along with arrows. Modules with names in bold get updated at corresponding phases.

In this section, we discuss related works on transfer learning and curriculum learning. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Transfer learning. Transfer learning or knowledge transfer has the goal to transfer the information of trained models to solve unknown tasks, thereby reducing the effort to collect new training data. *What* and *how* to transfer are key issues to be addressed. Different methods are applied to different source-target domains and bridge different transfer knowledge [PTKY11, YYH07, WZHY18, ZSS+18, SSF17, LSL+20, PSdV+18]. For deep models, a powerful transfer method is adapting a pre-trained model for a new task, often called *fine-tuning* (*FT*). Models pre-trained on large-scale datasets have proven to generalize better than randomly initialized ones [EBC+10]. Another popular transfer method is taking pre-trained networks as backbone and adding high-level functions, e.g. for object detection [HRS+17] and image segmentation [HGDG17, CPK+18]. Besides, the knowledge to transfer can be from multi-modal category models, e.g. the word embedding models used for zero-shot learning [RES13, XSSA19] and trained attribute models used for social relationship recognition [SSF17].

In this chapter, our meta-transfer learning leverages the idea of transferring pre-trained weights and our model meta-learns how to effectively transfer. The large-scale trained DNN weights are *what* to transfer, and the operations of *Scaling* and *Shifting* indicate *how* to transfer. Some existing few-shot learning methods [KVSN18, MRCA18, QLSY18, SRM18, RRS+19] also deployed pre-trained DNNs. DNN weights in these methods are usually fixed for feature extraction or simply fine-tuned on each task. In contrast, our approach defines an explicit meta-learner to extract and apply usable knowledge of pre-learned DNNs to tackling the challenging few-shot learning tasks.

Curriculum learning & hard sample mining. Curriculum learning was proposed by Bengio *et al.* [BLCW09] and is popular for multi-task learning [SGNK17, WCA18, GBM+17]. They showed that instead of observing samples at random it is better to organize samples in a meaningful way so that fast convergence, effective learning and better generalization can be achieved. Kumar *et al.* [KPK10] introduced an iterative self-paced learning algorithm where each iteration simultaneously selects easy samples and learns a new parameter vector. Intuitively, the curriculum is determined by the pupil's abilities rather than being fixed by a teacher. Pentina *et al*. [PSL15] use adaptive SVM classifiers to evaluate task difficulty for later organization. Most recently, Jiang *et al*. [JZL+18] designed a MentorNet that provides a "curriculum", i.e., sample weighting scheme, for StudentNet to focus on the labels which are probably correct. The trained MentorNet can be directly applied for the training of StudentNet on a new dataset. Differently, our MTL method does task difficulty evaluation online at the phase of test in each task, without needing any auxiliary model.

Hard sample mining was proposed by Shrivastava *et al*. [SGG16] for object detection with DNNs. It treats image proposals overlapped with ground truth (i.e. causing more confusions) as hard negative samples. Training on more confusing data enables the detection model to achieve higher robustness and better performance [CF16, HKC+17, DT05]. Inspired by this, we sample harder tasks online and make our MTL learner "grow faster and stronger through more hardness". In our experiments, we show that this can be generalized to different architectures with different meta-training operations, i.e. *SS* and *FT*, referring to Figure 7.4.

7.3 METHODOLOGY

As shown in Figure 7.1, our method consists of three main training phases in order to achieve effective few-shot classifiers. First, we train a DNN on large-scale data, e.g., on *mini*ImageNet with 64 classes and 600 samples per class [VBL+16], and then fix convolutional layers as the Feature Extractor. Second, in the meta-transfer learning phase, our MTL learns the *Scaling* and *Shifting* (*SS*) parameters for the neurons of Feature Extractor, enabling the fast adaptation to few-shot episodes (Section 7.3.2). To boost the overall learning efficiency, we apply the HT meta-batch scheme (Section 7.3.3) and the meta-gradient regularization (Section 7.3.4) to the meta-train phase. The overall algorithm of our approach is given in Section 7.3.5. Finally, in Section 7.3.6, we introduce how to plug this algorithm into existing methods.

7.3.1 Denotations for meta-learning

In this section, we briefly introduce the unified episodic formulation in meta-learning, following related works [VBL+16, RL17, FAL17, ORL18, RRS+19]. Then, we introduce the task-level data denotations used at two phases, i.e., meta-train and meta-test.

Meta-learning has an episodic formulation that was proposed for tackling few-shot tasks first in [VBL+16]. It is different from traditional image classification, in three aspects: (1) the main phases are not train and test but meta-train and meta-test, each of which includes training and testing; (2) the samples in meta-train and meta-test are not data points but episodes, and each episode is a few-shot classification task; and (3) the objective is not classifying unseen data points but to fast adapt the meta-learned experience or knowledge to the learning of a new few-shot classification task.

The denotations of two phases, meta-train and meta-test, are as follows. A meta-train example is a classification task $\mathcal{T}$ sampled from a distribution $p(\mathcal{T})$. $\mathcal{T}$ is called an episode, including a training split $\mathcal{T}^{(tr)}$ to optimize the base-learner, i.e., the classifiers in our model, and a test split $\mathcal{T}^{(te)}$ to optimize the meta-learner, i.e., the scaling and shifting parameters in our model. In particular, meta-train aims to learn from a number of episodes $\{\mathcal{T}\}$ sampled from $p(\mathcal{T})$. An unseen episode $\mathcal{T}_{unseen}$ in meta-test will start from that experience of the meta-learner and adapt the base-learner. The final evaluation is done by testing a set of unseen data points in $\mathcal{T}_{unseen}^{(te)}$.

Meta-train phase. This phase aims to learn a meta-learner from multiple episodes. In each episode, meta-training has a two-stage optimization. Stage-1 is called base-learning, where the cross-entropy loss is used to optimize the parameters of the base-learner. Stage-2 contains a feed-forward test on episode test data points. The test loss (also called meta loss) is used to optimize the parameters of the meta-learner. Specifically, given an episode $\mathcal{T} \in p(\mathcal{T})$, the base-learner $\theta_{\mathcal{T}}$ is learned from episode training data $\mathcal{T}^{(tr)}$ and its corresponding loss $\mathcal{L}_{\mathcal{T}}(\theta_{\mathcal{T}}, \mathcal{T}^{(tr)})$. After optimizing this loss, the base-learner has parameters $\tilde{\theta}_{\mathcal{T}}$. Then, the meta-learner is updated using meta loss $\mathcal{L}_{\mathcal{T}}(\tilde{\theta}_{\mathcal{T}}, \mathcal{T}^{(te)})$. After meta-training on all episodes, the meta-learner is optimized by meta losses $\{\mathcal{L}_{\mathcal{T}}(\tilde{\theta}_{\mathcal{T}}, \mathcal{T}^{(te)})\}_{\mathcal{T}\in p(\mathcal{T})}$. Therefore, the number of meta-learner updates equals the number of episodes.

Meta-test phase. This phase aims to test the performance of the trained meta-learner for fast adaptation to unseen episodes. Given $\mathcal{T}_{unseen}$, the meta-learner $\tilde{\theta}_{\mathcal{T}}$ teaches the base-learner $\theta_{\mathcal{T}_{unseen}}$ to adapt to the objective of $\mathcal{T}_{unseen}$ by some means, e.g. through initialization [FAL17]. Then, the test result on $\mathcal{T}_{unseen}^{(te)}$ is used to evaluate the meta-learning approach. If there are multiple unseen episodes $\{\mathcal{T}_{unseen}\}$, the average result on $\{\mathcal{T}_{unseen}^{(te)}\}$ will be the final evaluation.

7.3.2 Meta-transfer learning (MTL)

During pre-training, we merge all data $\mathcal{D}$ and derive the many-shot many-class model using the cross-entropy loss. The model is composed of the Feature Extractor Θ and a many-class classifier. The Θ keeps frozen in the following meta-training and meta-test phases, as shown in Figure 7.1. The many-class classifier is discarded, because few-shot episodes contain different classification objectives, e.g., 5-class instead of 64-class classification for *mini*ImageNet [VBL+16].

As shown in Figure 7.1(b), our MTL optimizes only the meta operations *Scaling* and *Shifting* (*SS*) through HT meta-batch training (Section 7.3.3). Figure 7.2 visualizes the difference of updating through *SS* and *FT* (*Fine-Tuning*). *SS* operations, denoted as Φ_{S_1} and Φ_{S_2}, do not change the frozen neuron weights of Θ during learning, while *FT* updates the complete Θ. Note that this *FT* is distinct from the Base-learner *FT* (on θ).

In the following, we expand the details of *SS* operations corresponding to Figure 7.1(b). Given an episode $\mathcal{T}$, the loss of $\mathcal{T}^{(tr)}$ is used to optimize the current base-learner (classifier) θ' by gradient descent:

$$\theta' \leftarrow \theta - \beta \nabla_{\theta} \mathcal{L}_{\mathcal{T}^{(tr)}}\big([\Theta;\theta], \Phi_{S_{\{1,2\}}}\big), \tag{7.1}$$

where θ concerns a few classes, e.g., 5 classes, to classify each time in a novel few-shot setting. θ' corresponds to a temporal classifier working only in the current episode, initialized by the θ optimized by previous episodes (see Eq. (7.3)).

Φ_{S_1} is initialized by ones and Φ_{S_2} by zeros. Then, they are optimized by the meta loss of $\mathcal{T}^{(te)}$ as follows,

$$\Phi_{S_i} =: \Phi_{S_i} - \gamma \nabla_{\Phi_{S_i}} \mathcal{L}_{\mathcal{T}^{(te)}}\big([\Theta;\theta'], \Phi_{S_{\{1,2\}}}\big), i = 1, 2. \tag{7.2}$$

In this step, θ is updated with the same learning rate γ as in Eq. (7.2),

$$\theta =: \theta - \gamma \nabla_{\theta} \mathcal{L}_{\mathcal{T}^{(te)}}\big([\Theta;\theta'], \Phi_{S_{\{1,2\}}}\big). \tag{7.3}$$

Re-linking to Eq. (7.1), we note that the above θ' comes from the last epoch of base-learning on $\mathcal{T}^{(tr)}$.

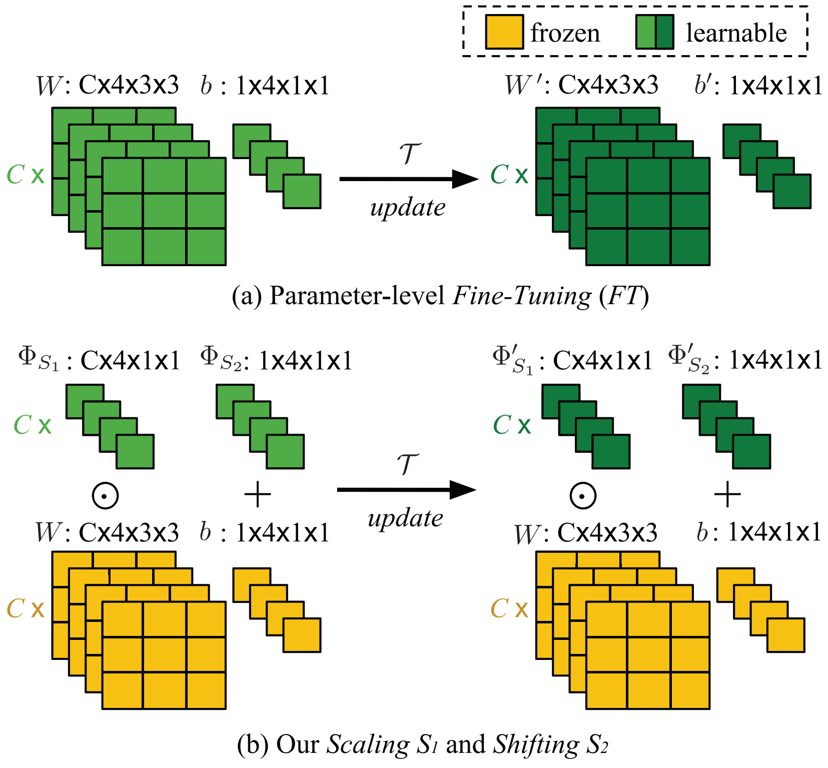


Figure 7.2: Two kinds of meta operations on pre-trained weights. (a) Parameter-level *Fine-Tuning* (*FT*) is a conventional meta-train operation used in related works such as MAML [FAL17], ProtoNets [SSZ17] and RelationNets [SYZ+18]. Its update works for all neuron parameters, W and b. (b) Our neuron-level *Scaling* and *Shifting* (*SS*) operations in MTL. They reduce the number of learning parameters and avoid overfitting problems. In addition, they keep large-scale trained parameters (in yellow) frozen, preventing "catastrophic forgetting" [LR17, MC89].

Next, we describe how we apply $\Phi_{S_{\{1,2\}}}$ to the frozen neurons as shown in Figure 7.2(b). Given the trained Θ, for its l-th layer containing K neurons, we have K pairs of parameters, respectively as weight and bias, denoted as $\{(W_{i,k}, b_{i,k})\}$. Note that the neuron location l, k will be omitted for readability. Based on MTL, we learn K pairs of scalars $\{\Phi_{S_{\{1,2\}}}\}$. Assuming X is the input, we apply $\{\Phi_{S_{\{1,2\}}}\}$ to (W, b) as,

$$SS(X; W, b; \Phi_{S_{\{1,2\}}}) = (W \odot \Phi_{S_1})X + (b + \Phi_{S_2}), \tag{7.4}$$

where $\odot$ denotes the element-wise multiplication.

Taking Figure 7.2(b) as an example of a single 3×3 filter, after *SS* operations, this filter is scaled by Φ_{S_1} then the feature maps after convolutions are shifted by Φ_{S_2} in addition to the original bias b. Detailed steps of *SS* are given in Algorithm 4 in Section 7.3.5.

Figure 7.2(a) shows a typical parameter-level *FT* operation, which is in the meta optimization phase of our related work MAML [FAL17]. It is obvious that *FT* updates the complete values of W and b, and has a large number of parameters, and our *SS* reduces this number to below $\frac{2}{9}$ in the example of the figure. In summary, *SS* can benefit the few-shot learning model in three aspects. 1) It starts from a strong initialization based on a large-scale trained DNN,

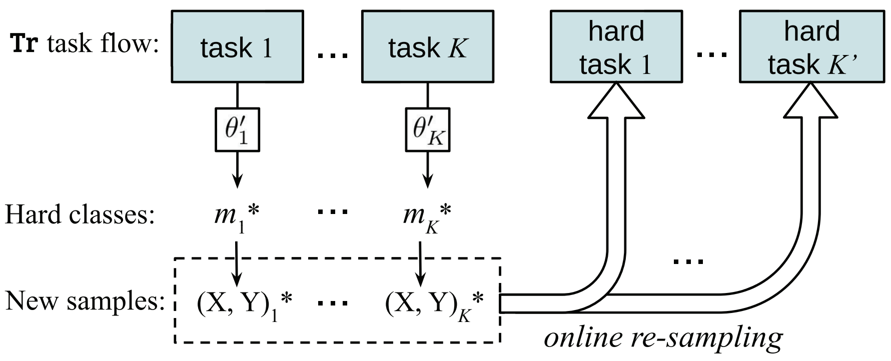


Figure 7.3: The computation flow of online hard task sampling. During an HT meta-batch phase, the meta-training first goes through K random tasks then continues on re-sampled K' hard tasks.

yielding fast convergence for MTL. 2) It does not change DNN weights, thereby avoiding the problem of "catastrophic forgetting" [LR17, MC89] when learning specific episodes in MTL. 3) It is light-weight, reducing the chance of overfitting of MTL in few-shot scenarios. In experiments, we compare *SS* with *FT* based on multiple baseline methods, and show the clear superiority of *SS* against the problem of "forgetting".

7.3.3 Hard task (HT) meta-batch

In this section, we introduce a method to schedule hard tasks in meta-training batches. The conventional meta-batch is composed of randomly sampled episodes, where the randomness implies random difficulties [FAL17]. In our meta-training pipeline, we intentionally pick up failure cases in each episode and re-compose their data to be harder episodes for adverse re-training. The task flow is shown in Figure 7.3. We aim to force our meta-learner to "grow up through hardness".

Pipeline. Given a (M-class, N-shot) episode $\mathcal{T}$, a meta-batch $\{\mathcal{T}_{1\sim k}\}$ contains two splits, $\mathcal{T}^{(tr)}$ and $\mathcal{T}^{(te)}$, for base-learning and test, respectively. The base-learner is optimized by the loss of $\mathcal{T}^{(tr)}$ (in multiple epochs). *SS* parameters are then optimized by the loss of $\mathcal{T}^{(te)}$ once. During the loss computation on $\mathcal{T}^{(te)}$, we can also get the recognition accuracy for M classes. Then, we choose the lowest accuracy Acc_{m^*} to determine the most difficult class m^* (also called failure class) in the current episode.

After obtaining all failure classes $\{m^*\}$ from $\{\mathcal{T}_{1\sim k}\}$ in the current meta-batch (k is the batch size), we re-sample episodes from the data indexed by $\{m^*\}$. Specifically, we assume $p(\mathcal{T}|\{m^*\})$ is the task distribution, we sample a "harder" episode $\mathcal{T}^{hard} \in p(\mathcal{T}|\{m^*\})$. Two important details are given below.

Choosing hard class m^*. We choose the failure class $m*$ from each episode by ranking the class-level accuracies instead of fixing a threshold. In a dynamic online setting as ours, it is more sensible to choose the hardest cases based on ranking rather than fixing a threshold ahead of time.

Two methods of hard tasking using $\{m^*\}$. Chosen $\{m^*\}$, we can re-sample episodes $\mathcal{T}^{hard}$ by (1) directly using the samples of m^*-th class in the current episode $\mathcal{T}$, or (2) indirectly using the index m^* to sample new samples of that class. In fact, setting (2) considers to include more data variance of m^*-th class and it works better than setting (1) in general.

Algorithm 4: MTL with HT meta-batch strategy

```
Input: Task distribution $p(\mathcal{T})$ and corresponding dataset $\mathcal{D}$, learning rates $\alpha$, $\beta$ and $\gamma$
Output: Feature extractor $\Theta$, base learner $\theta$, Scaling and Shifting parameters $\Phi_{S_{\{1,2\}}}$
1  Randomly initialize $\Theta$ and $\theta$;
2  for samples in $\mathcal{D}$ do
3      Evaluate $\mathcal{L}_{\mathcal{D}}([\Theta;\theta])$ ;
4      Optimize $\Theta$ and $\theta$;
5  end
6  Initialize $\Phi_{S_1}$ by ones, initialize $\Phi_{S_2}$ by zeros; Reset $\theta$ for few-shot episodes;
    Randomly initialize $\theta$; Initialize $\{m^*\}$ as an empty set.
7  for iterations in meta-training do
8      Randomly sample a batch of episodes $\{\mathcal{T}_{1\sim K}\} \in p(\mathcal{T})$;
9      for k from 1 to K do
10         for samples in $\mathcal{T}_k^{(tr)}$ do
11             Evaluate $\mathcal{L}_{\mathcal{T}_k^{(tr)}}$;
12             Optimize $\theta'$ by Eq. (7.1);
13         end
14         Optimize $\Phi_{S_{\{1,2\}}}$ and $\theta$ by Eq. (7.5) and Eq. (7.6);
15         for $m \in \{1 \sim M\}$ do
16             Classify samples of $m$-th class in $\mathcal{T}_k^{(te)}$;
17             Compute $Acc_m$;
18         end
19         Get the returned $m^*$-th class, then add it to set $\{m^*\}$;
20     end
21     Sample hard tasks $\{\mathcal{T}^{hard}\} \subseteq p(\mathcal{T}|\{m^*\})$;
22     for k from 1 to K' do
23         Sample episode $\mathcal{T}_k^{hard} \in \{\mathcal{T}^{hard}\}$ ;
24         for samples in $\mathcal{T}_k^{hard,(tr)}$ do
25             Evaluate $\mathcal{L}_{\mathcal{T}_k^{hard,(tr)}}$;
26             Optimize $\theta'$ by Eq. (7.1);
27         end
28         Optimize $\Phi_{S_{\{1,2\}}}$ and $\theta$ by Eq. (7.5) and Eq. (7.6);
29     end
30     Empty $\{m^*\}$.
31 end
```

7.3.4 Meta-gradient regularization

In order to further reduce the "catastrophic forgetting" problem, we deploy an easy and efficient meta-gradient regularization method for each training episode. Particularly, we apply this regularization to updating $\Phi_{S_{\{1,2\}}}$ and θ. Let q denote the index of the current episode. Let $\nabla\mathcal{L}_{\mathcal{T}_r^{(te)}}$ be the gradient of the r-th episode. Eq. (7.2) and Eq. (7.3) can be rewritten as,

$$\begin{aligned}\Phi_{S_i} =:&\Phi_{S_i} - \gamma\nabla_{\Phi_{S_i}}\mathcal{L}_{\mathcal{T}_q^{(te)}}\big([\Theta;\theta'],\Phi_{S_{\{1,2\}}}\big)\\&-\gamma\psi_1\sum_{r=q-p}^{q-1}\nabla_{\Phi_{S_i}}\mathcal{L}_{\mathcal{T}_r^{(te)}}\big([\Theta;\theta'],\Phi_{S_{\{1,2\}}}\big), i=1,2;\end{aligned}\tag{7.5}$$

$$\begin{aligned}\theta =:&\theta - \gamma\nabla_{\theta}\mathcal{L}_{\mathcal{T}_q^{(te)}}\big([\Theta;\theta'],\Phi_{S_{\{1,2\}}}\big)\\&-\gamma\psi_2\sum_{r=q-p}^{q-1}\nabla_{\theta}\mathcal{L}_{\mathcal{T}_r^{(te)}}\big([\Theta;\theta'],\Phi_{S_{\{1,2\}}}\big),\end{aligned}\tag{7.6}$$

where ψ_1 and ψ_2 are two temperature scalars to balance the weights of the meta-gradients from current and previous episodes.

7.3.5 The overall algorithm

We elaborate on the training process using our approach in Algorithm 1. There are two main training stages: large-scale DNN training (lines 1-5) and meta-transfer learning (lines 6-31). In particular, the proposed HT meta-batch sampling (with the subsequent training) is given on lines 21-30. Note that the indices of failure classes are returned on line 19.

7.3.6 Plug MTL into baseline methods

Conventional supervised few-shot learning methods include metric learning based (e.g., ProtoNets [SSZ17], MatchingNets [VBL+16], and RelationNets [SYZ+18]) and optimization based (e.g., MAML [FAL17]). For semi-supervised few-shot learning, there are Masked Soft k-Means [RTR+18], TPN [LLP+19], and LST [LSL+19]. The neural network architecture in these methods is often composed of two modules, i.e., convolutional-layer feature extractor Θ and fully-connected-layer classifier θ. Our MTL operations *SS* are conducted on convolutional neurons, so they are generic and easy to plug in Θ.

First, we pre-train Θ on a many-shot classification task using the whole set of $\mathcal{D}$. Then, we plug-in *Scaling* and *Shifting* weights Φ_{SS} on each neuron of Θ and update them with meta loss. Given an episode $\mathcal{T}$, we feed training images $x^{(tr)}$ and test images $x^{(te)}$ to the feature extractor $\Theta \odot \Phi_{SS}$, and obtain the embedding $e^{(tr)}$ and $e^{(te)}$, respectively. We apply different classifier architectures [SSZ17, VBL+16, SYZ+18, FAL17, RTR+18, LLP+19, LSL+19] to train classifiers with $e^{(tr)}$ and $y^{(tr)}$, and then test with $e^{(te)}$ resulting in the predictions $\hat{y}^{(te)}$. We then compute the test loss using $\hat{y}^{(te)}$ and $y^{(te)}$. Using this loss, we proceed meta-gradient back-propagation to update Φ_{SS} as well as the original meta-learner proposed in the baseline methods, e.g. the initialization network of base-learner θ in MAML [FAL17]. In experiments, we report all our plug-in results compared to those of using *FT* operations (see Table 7.4 and Table 7.5).

7.4 EXPERIMENTS

We evaluate the proposed approach in terms of few-shot recognition accuracy and model convergence speed. Below we describe the datasets we evaluate on and detailed settings, followed by the comparisons to state-of-the-art methods, validations on several baseline methods with *SS* plugin, and an ablation study regarding the key components of our approach, i.e., *SS* operations, HT meta-batch, and meta-gradient regularization. In the end, we demonstrate the statistical numbers and Gaussian fitting curves for the meta-learned *SS* parameters.

7.4.1 Datasets

We conduct few-shot learning experiments on three benchmarks, *mini*ImageNet [VBL+16], *tiered*ImageNet [RTR+18] and Fewshot-CIFAR100 (FC100) [ORL18]. *mini*ImageNet is the most widely used in related works [FAL17, RL17, GFL+18, FFS+18, MYMT18], and the later ones are more recently published with a larger scale and a more challenging setting, i.e., lower image resolution and stricter training-test splits.

***mini*ImageNet [VBL+16].** It was proposed especially for the few-shot learning evaluation [VBL+16]. Its complexity is high due to the use of ImageNet images, but it requires fewer resources and infrastructure than running on the full ImageNet dataset [RDS+15]. In total, there are 100 classes with 600 samples of 84×84 color images per class. These 100 classes are divided into 64, 16, and 20 classes respectively for sampling episodes for meta-train, meta-validation and meta-test, following related works [FAL17, RL17, GFL+18, FFS+18, MYMT18].

***tiered*ImageNet [RTR+18].** Compared to *mini*ImageNet, it is a larger subset of ImageNet with 608 classes (779, 165 images) grouped into 34 super-class nodes. These nodes are partitioned into 20, 6, and 8 disjoint sets respectively for meta- training, validation, and test. The corresponding sub-classes are used to build the classification tasks in each of which the 5 sub-classes are randomly sampled. As argued in [RTR+18], this super-class based training-test split results in a more challenging and realistic regime with meta- test and validation episodes that are less similar to meta-train episodes.

Fewshot-CIFAR100 (FC100) [ORL18]. This dataset is based on the popular object classification dataset CIFAR100 [Kri09]. Its training-test splits are also based on super-classes [ORL18]. In total, it contains 100 object classes (600 images per class) belonging to 20 super-classes. meta-train data are from 60 classes belonging to 12 super-classes. Meta-validation and meta-test sets contain 20 classes belonging to 4 super-classes, respectively. Compared to the ImageNet subsets above, FC100 offers a more challenging scenario with lower image resolution, i.e. each sample is a 32×32 color image. In addition, the super-class gap on FC100 is more significant than that on ImageNet datasets.

Semi-supervised splits. On *mini*ImageNet and *tiered*ImageNet, we follow the semi-supervised task splitting method used in previous works [LSL+19, RTR+18, LLP+19]. In addition to the supervised data (same as above), we use 30 (50) unlabeled images per class for every 1-shot (5-shot) episode. In a more difficult setting, we use unlabeled data from 3 distracting classes (same number of samples with non-distracting classes) that are excluded in the support set [LLP+19, RTR+18]

7.4.2 Implementation details

Episode sampling. We use the same episode sampling method as related works [FAL17], on all datasets. Specifically, (1) we consider the 5-class classification, (2) during meta-train, we sample 5-class, 1-shot (or 5-shot) episodes to contain 1 (or 5) samples for train episode and 15 (uniform) samples for episode test, and (3) during meta-validation and meta-test, we sample 5-class, 1-shot (or 5-shot) episodes to contain 1 (or 5) samples for train episode and 1 (uniform) sample for episode test. Note that in some related works, e.g., [ORL18], 32 samples are used for episode test on 5-shot episodes. Using such a larger number of test samples results in a lower standard variance of recognition accuracies.

In total, we sample at most $20k$ episodes ($10k$ meta-batches) for meta-train (same for the cases *w/* and *w/o* HT meta-batch), and sample 600 random episodes for both meta-validation and meta-test [FAL17]. Note that we choose the trained models which have the highest meta-validation accuracies, for meta-test.

Network architectures. We present the details of network architectures for Feature Extractor parameters Θ, MTL meta-learner with *Scaling* and *Shifting* (*SS*) parameters Φ_{S_1}, Φ_{S_2}, and MTL base-learner (classifier) parameters θ. For Θ, in our conference version [SLCS19], we used ResNet-12 and 4CONV which are also commonly used in previous works [FAL17, VBL+16, RL17, MYMT18, MRCA18, ORL18, LMRS19]. In this journal version, we implement two deeper architectures – ResNet-18, ResNet-25 and WRN-28-10 which have been adopted in newly published related works [LED+19, QLSY18, YHZS18, HMX+20], and we achieve the top performance using ResNet-25 and WRN-28-10. In specific, **4CONV** consists of 4 layers with 3×3 convolutions and 32 filters, followed by batch normalization (BN) [IS15], a ReLU nonlinearity, and 2×2 max-pooling. MTL only works with the following deep nets. **ResNet-12** contains 4 residual blocks and each block has 3 CONV layers with 3×3 kernels. At the end of each residual block, a 2×2 max-pooling layer is applied. The number of filters starts from 64 and is doubled every next block. Following 4 blocks, there is a mean-pooling layer to compress the output feature maps to a feature embedding. **ResNet-18** contains 4 residual blocks and each block has 4 CONV layers with 3×3 kernels. The number of filters starts from 64 and is doubled every next block. Before the residual blocks, there is one additional CONV layer with 64 filter and 3×3 kernels at the beginning of the network. The residual blocks are followed by an average pooling layer. The ResNet-18 backbone we use exactly follows [HZRS16] except that the last FC layer is removed. **ResNet-25** is exactly the same as the released code of [QLSY18, YHZS20]. Three residual blocks are used after an initial convolutional layer. Each block has 4 CONV layers with 3×3 kernels. The number of filters starts from 160 and is doubled every next block. After a global average pooling layer, it leads to a 640-dim embedding. **WRN-28-10** has its depth and width set to 28 and 10, respectively. After a global average pooling in the last layer of the backbone, it gets a 640-dimensional embedding. For this backbone, we resize the input image to $80 \times 80 \times 3$ for a fair comparison with related methods [SLCS19, HMX+20]. Other details are the same as those of ResNet-25 [YHZS20, RRS+19]. Note that we employ this architecture using the code of SIB [HMX+20] and implement only our *SS* operations to it.

For the architecture of Φ_{S_1} and Φ_{S_2}, actually, they are generated according to the architecture of Θ, as introduced in Section 7.3.2. For example, when using ResNet-25 in MTL, Φ_{S_1} and Φ_{S_2} also have 25 layers, respectively.

For the architecture of θ (the parameters of the base-learner), we empirically find that in our cases a single FC layer (as θ) is faster to train and more effective for classification

Base-learning	**Dim. of θ**	***mini*ImageNet**	
		1-shot	**5-shot**
θ (2 FC layers)	512, 5	59.1 $\pm$ 1.9	70.7 $\pm$ 0.9
θ (3 FC layers)	1024, 512, 5	56.2 $\pm$ 1.8	68.7 $\pm$ 0.9
Θ, θ	5	59.6 $\pm$ 1.8	71.6 $\pm$ 0.9
θ (**Ours**)	5	**60.6** $\pm$ 1.9	**74.3** $\pm$ 0.8

Table 7.1: The 5-way, 1-shot and 5-shot classification accuracy (%) on *mini*ImageNet, for choosing the best architecture of base-learner (i.e., the classifier θ). "meta-batch" and "ResNet-12 (pre)" are used.

than multiple layers, taking the most popular dataset *mini*ImageNet as an example. Results are given in Table 7.1, in which we can see the performance drop when changing this θ to multiple layers.

Pre-training stage. For the phase of DNN training on large-scale data, the model is trained by Adam optimizer [KB14]. The learning rate is initialized as 0.001, and decays to its half every $5k$ iterations until it is lower than 0.0001. We set the keep probability of the dropout as 0.9 and batch-size as 64. The pre-training stops after $10k$ iterations. Note that for hyperparameter selection, we randomly choose 550 samples for each class as the training set, and the rest as validation. After the grid search for hyperparameters, we fix them and mix up all samples (64 classes, 600 samples each class) to do the final pre-training. Image samples in these steps are augmented by horizontal flipping.

Meta-train stage. This is a task-level training in which the base-learning in one task considers a training step for optimizing base-learner, followed by a validation step for optimizing meta-learner. The base-learner θ is optimized by batch gradient descent with the learning rate of 0.01. It is updated with 20 and 60 epochs respectively for 1-shot and 5-shot episodes on the *mini*ImageNet and *tiered*ImageNet datasets, and 20 epochs for all episodes on the FC100 dataset. Specially when using ResNet-25, we use 100 epochs for all episodes on all datasets. The meta-learner, i.e. the parameters of the *SS* operations, is optimized by Adam optimizer [KB14]. Its learning rate is initialized as 0.001, and decays to the half every $1k$ iterations until 0.0001. The size of meta-batch is set to 2 (episodes) due to the memory limit. For meta-gradient regularization, each time we deploy 8 previous episodes to compute meta gradients, and set temperature scalars ψ_1 and ψ_2 both as 1.0.

HT meta-batch. Hard tasks are sampled every time after running 10 meta-batches, i.e., the failure classes used for sampling hard tasks are from 20 episodes as each meta-batch contains 2 episodes. The number of hard tasks is selected for different settings by validation: 10 and 4 hard tasks respectively for the 1-shot and 5-shot experiments, on the *mini*ImageNet and *tiered*ImageNet datasets; and respectively 20 and 10 hard tasks for the 1-shot, 5-shot experiments, on the FC100 dataset.

Ablative settings. In order to show the effectiveness of our *SS* operations, we carefully design several ablative settings: two baselines without meta-learning but more classic learning, named as *update**, four baselines of *Fine-Tuning* (*FT*) on different numbers of parameters in the outer-loop based on MAML [FAL17], named as *FT**, and two *SS* variants on smaller numbers of parameters, named as *SS**. Table 3.2 shows the results in these settings, for which we simply use the classical architecture (ResNet-12) containing 4 residual blocks named

$\Theta 1 \sim \Theta 4$ and an FC layer θ (classifier). The bullet names used in the Table are explained as follows.

update $[\Theta; \theta]$ **(or** θ**).** There is no meta-train phase. During test phase, each episode has its whole model $[\Theta; \theta]$ (or the classifier θ) updated on $\mathcal{T}^{(tr)}$, and then tested on $\mathcal{T}^{(te)}$.

FT θ **(**$[\Theta 4; \theta]$ **or** $[\Theta 3; \Theta 4; \theta]$ **or** $[\Theta; \theta]$**).** These are straight-forward ways to reduce the quantity of meta-learned parameters. For example, "$[\Theta 3; \Theta 4; \theta]$" does not update the the first two residual blocks which encode the low-level image features. Specially, "θ" means only the classifier parameters are updated during meta-train.

SS $[\Theta 4; \theta]$ **(or** $[\Theta 3; \Theta 4; \theta]$ **or** $[\Theta; \theta]$**).** During the meta-train, *SS* parameters are defined and used on $\Theta 4$. Low-level residual blocks, e.g. $\Theta 1$, deploy the pre-trained weights without meta-level update.

SS $[\Theta; \theta]$**, *regularized.*** Our method of meta-transfer learning on the whole backbone and with meta-gradient regularization.

7.4.3 Comparison to the state-of-the-art

Table 7.2 and Table 7.3 present the overall comparisons to related works, on the *mini*ImageNet, *tiered*ImageNet, and FC100 datasets. Note that these numbers are the meta-test results of the meta-trained models which have the highest meta-validation accuracies. On the *mini*ImageNet, models on 1-shot and 5-shot are meta-trained for $6k$ and $10k$ iterations, respectively. On the *tiered*ImageNet, iterations for 1-shot and 5-shot are at $8k$ and $10k$, respectively. On the FC100, iterations are all at $3k$.

***mini*ImageNet.** In Table 7.2, we can see that "SIB + *SS* $[\Theta; \theta]$" and "*SS* $[\Theta; \theta]$, HT meta-batch" achieve top performances for 1-shot and 5-shot tasks, respectively. Regarding the network architecture, we can see that models using deeper ones, e.g. ResNet-25 and WRN-28-10, outperform 4CONV-based models by quite large margins, e.g. 4CONV models have the best 1-shot result with 55.51% [LLP+19] which is 9.4% lower than 64.9% (our method on ResNet-25). This clearly validates our contribution of utilizing deeper neural networks to tackle the few-shot classification problems.

***tiered*ImageNet.** In Table 7.3, we give the results on the larger dataset — *tiered*ImageNet. Since this dataset is newly proposed [RTR+18], its results of using previous methods [SSZ17, SYZ+18, FAL17] were reported by [RTR+18, LLP+19]. From the table, we again confirm that "SIB + *SS* $[\Theta; \theta]$" outperforms others, e.g. it achieves around a margin of 2.6% over the original SIB [HMX+20] on 1-shot tasks. An interesting observation is that on this larger and more challenging dataset, our deeper version of MTL (ResNet-25) outperforms the shallower one (ResNet-12) by 7% on 1-shot, which is twice the margin on *mini*ImageNet (3.5%). This shows our idea of transferring knowledge from pre-trained DNNs is more promising for handling harder few-shot settings.

FC100. In Table 7.3, we also show the results on the FC100. We report the numbers of TADAM [ORL18] and MetaOptNet [LMRS19] given in original papers, and obtain the results of classical methods, i.e. MAML [FAL17], MAML++ [AES19], RelationNets [SYZ+18] and MatchineNets [VBL+16], by implementing their open-sourced code on the deeper pre-trained networks. From the table, we can see that our approach consistently outperforms MAML and its improved version MAML++ by large margins, e.g. over 7% for 1-shot tasks. Besides, it surpasses TADAM and MetaOptNet by 6% and 5%, respectively. Implementing *SS* operations on SIB brings around 1% gains over the original both for 1-shot and 5-shot.

Few-shot Learning Method		**Backbone**	***mini*ImageNet (test)**	
			1-shot	**5-shot**
Data augmentation	Adv. ResNet, [MD17]	WRN-40 (pre)	55.2	69.6
	Delta-encoder, [SKS+18]	VGG-16 (pre)	58.7	73.6
Metric learning	MatchingNets, [VBL+16]	4 CONV	43.44 ± 0.77	55.31 ± 0.73
	ProtoNets, [SSZ17]	4 CONV	49.42 ± 0.78	68.20 ± 0.66
	RelationNets, [SYZ+18]	4 CONV	50.44 ± 0.82	65.32 ± 0.70
	Graph neural network, [SE18]	4 CONV	50.33 ± 0.36	66.41 ± 0.63
	Ridge regression, [BHTV19]	4 CONV	51.9 ± 0.2	68.7± 0.2
	TransductiveProp, [LLP+19]	4 CONV	55.51	69.86
Memory network	Meta Networks, [MY17]	5 CONV	49.21 ± 0.96	–
	SNAIL, [MRCA18]	ResNet-12 (pre)$^\diamond$	55.71 ± 0.99	68.88 ± 0.92
	TADAM, [ORL18]	ResNet-12 (pre)†	58.5 ± 0.3	76.7 ± 0.3
	Cross-Modulation Nets, [PDH18]	4 CONV	50.94 ± 0.61	66.65 ± 0.67
	Isotropic Gaussian, [BRCŚ+17]	ResNet-34 (pre)	56.3 ± 0.4	73.9 ± 0.3
Gradient descent	MAML, [FAL17]	4 CONV	48.70 ± 1.75	63.11 ± 0.92
	MAML++, [AES19]	4 CONV	52.15 ± 0.26	68.32 ± 0.44
	Meta-LSTM, [RL17]	4 CONV	43.56 ± 0.84	60.60 ± 0.71
	Hierarchical Bayes, [GFL+18]	4 CONV	49.40 ± 1.83	–
	MT-net, [LC18]	4 CONV	51.70 ± 1.84	–
	Bilevel Programming, [FFS+18]	ResNet-12$^\diamond$	50.54 ± 0.85	64.53 ± 0.68
	MetaGAN, [ZCG+18]	ResNet-12	52.71 ± 0.64	68.63 ± 0.67
	adaResNet, [MYMT18]	ResNet-12‡	56.88 ± 0.62	71.94 ± 0.57
	MetaOptNet, [LMRS19]	ResNet-12	62.64 ± 0.35	78.63 ± 0.68
	LEO, [RRS+19]	WRN-28-10 (pre)	61.67 ± 0.08	77.59 ± 0.12
	LGM-Net, [LDM+19]	MetaNet+4CONV	69.13 ± 0.35	71.18 ± 0.68
	CTM, [LED+19]	ResNet-18 (pre)	64.12 ± 0.82	80.51 ± 0.13
	SIB, [HMX+20]	WRN-28-10 (pre)	***70.0*** ± 0.6	79.2 ± 0.4
Ours	*FT* $[\Theta;\theta]$, HT meta-batch	ResNet-12 (pre)	58.7 ± 1.8	73.2 ± 0.8
	SS $[\Theta;\theta]$, HT meta-batch	ResNet-12 (pre)	61.4 ± 1.8	75.9 ± 0.8
	SS $[\Theta;\theta]$, HT meta-batch	ResNet-18 (pre)	62.0 ± 1.9	76.0 ± 0.8
	SS $[\Theta;\theta]$, HT meta-batch	ResNet-25 (pre)	64.9 ± 1.8	81.2 ± 0.8
	SIB + *SS* $[\Theta;\theta]$	WRN-28-10 (pre)	71.0 ± 0.7	***81.0*** ± 0.4

$^\diamond$Additional 2 convolutional layers
‡Additional 1 convolutional layer
†Additional 72 fully connected layers

Table 7.2: The 5-way, 1-shot and 5-shot classification accuracy (%) on *mini*ImageNet datasets. "pre" means including our pre-training step with all training data points. The best and ***second best*** results are highlighted. Note that (1) the standard variance is affected by the number of episode test samples, and our sample splits are the same with MAML [FAL17]; and (2) our methods with *SS* $[\Theta;\theta]$ all use meta-gradient regularization.

Few-shot Learning Method	**Backbone**	***tiered*ImageNet (test)**		**FC100 (test)**	
		1-shot	**5-shot**	**1-shot**	**5-shot**
ProtoNets, [SSZ17] (by [RTR+18])	4 CONV	53.31 $\pm$ 0.89	72.69 $\pm$ 0.74	–	–
ProtoNets, [SSZ17] (by us)	ResNet-25	65.30 $\pm$ 1.70	83.00 $\pm$ 0.70	41.1 $\pm$ 1.8	58.6 $\pm$ 0.8
RelationNets, [SYZ+18] (by [LLP+19])	4 CONV	54.48 $\pm$ 0.93	71.32 $\pm$ 0.78	–	–
TransductiveProp, [LLP+19]	4 CONV	57.41 $\pm$ 0.94	71.55 $\pm$ 0.74	–	–
MAML, [FAL17] (by us)	4CONV	49.0 $\pm$ 1.8	66.5 $\pm$ 0.9	38.1 $\pm$ 1.7	50.4 $\pm$ 1.0
MAML++, [AES19] (by us)	4CONV	51.5 $\pm$ 0.5	70.6 $\pm$ 0.5	38.7 $\pm$ 0.4	52.9 $\pm$ 0.4
TADAM, [ORL18]	ResNet-12 (pre)†	62.13 $\pm$ 0.31	81.92 $\pm$ 0.30	40.1 $\pm$ 0.4	56.1 $\pm$ 0.4
MetaOptNet [LMRS19]	ResNet-12	65.99 $\pm$ 0.72	81.56 $\pm$ 0.53	41.1 $\pm$ 0.6	55.5 $\pm$ 0.6
CTM, [LED+19]	ResNet-18 (pre)	68.41 $\pm$ 0.39	***84.28*** $\pm$ 1.73	–	–
LEO, [RRS+19]	WRN-28-10 (pre)	66.33 $\pm$ 0.05	81.44 $\pm$ 0.09	–	–
SIB, [HMX+20] (by us)	WRN-28-10 (pre)	***72.9*** $\pm$ 0.6	82.8 $\pm$ 0.4	45.2 $\pm$ 0.6	55.9 $\pm$ 0.4
FT $[\Theta;\theta]$, HT meta-batch	ResNet-12 (pre)	64.7 $\pm$ 1.7	78.5 $\pm$ 0.8	42.0 $\pm$ 1.8	55.2 $\pm$ 0.8
SS $[\Theta;\theta]$, HT meta-batch	ResNet-12 (pre)	65.3 $\pm$ 1.8	81.2 $\pm$ 0.8	45.3 $\pm$ 1.8	57.5 $\pm$ 0.8
SS $[\Theta;\theta]$, HT meta-batch	ResNet-18 (pre)	68.1 $\pm$ 1.8	82.3 $\pm$ 0.8	45.5 $\pm$ 1.8	***57.9*** $\pm$ 0.8
SS $[\Theta;\theta]$, HT meta-batch	ResNet-25 (pre)	72.3 $\pm$ 1.8	85.6 $\pm$ 0.8	46.1 $\pm$ 1.8	61.4 $\pm$ 0.8
SIB + *SS* $[\Theta;\theta]$	WRN-28-10 (pre)	75.5 $\pm$ 0.7	84.3 $\pm$ 0.4	***45.9*** $\pm$ 0.7	56.7 $\pm$ 0.4

†Additional 72 fully connected layers.

Table 7.3: The 5-way, 1-shot and 5-shot classification accuracy (%) on *tiered*ImageNet and FC100 datasets. "pre" means including our pre-training step with all training datapoints. "by [*]" means the results were reported in [*]. "by us" means our implementation using open-sourced code. The best and ***second best*** results are highlighted. Note that (1) the standard variance is affected by the number of episode test samples, and our sample splits are the same with MAML [FAL17]; and (2) our methods with *SS* $[\Theta;\theta]$ all use meta-gradient regularization.

Method	**Operation**	***mini*ImageNet**		***mini*ImageNet (*tiered*Pre)**		***tiered*ImageNet**	
		1-shot	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
ProtoNets [SSZ17]	*SS*	56.7 ± 1.9	72.0 ± 0.9	62.0 ± 1.9	77.9 ± 1.0	62.2 ± 2.1	78.1 ± 0.9
	FT	55.2 ± 1.9	70.8 ± 0.9	57.2 ± 1.9	75.9 ± 0.9	54.9 ± 2.0	73.0 ± 1.0
MatchingNets [VBL+16]	*SS*	58.1 ± 1.8	66.9 ± 0.9	63.6 ± 1.7	73.2 ± 0.9	64.5 ± 1.9	73.9 ± 0.9
	FT	57.4 ± 1.7	67.5 ± 0.8	61.1 ± 1.8	72.6 ± 0.8	62.4 ± 1.8	73.5 ± 0.8
RelationNets [SYZ+18]	*SS*	57.2 ± 1.8	71.1 ± 0.9	61.5 ± 1.8	74.9 ± 0.9	***65.6*** ± 1.9	77.5 ± 0.9
	FT	56.0 ± 1.8	69.0 ± 0.8	58.9 ± 1.8	72.0 ± 0.8	62.2 ± 1.8	76.0 ± 0.9
MTL (FC)	*SS*	60.6 ± 1.9	***74.3*** ± 0.8	***65.7*** ± 1.8	***78.4*** ± 0.8	***65.6*** ± 1.7	***78.7*** ± 0.9
MAML [FAL17] (FC)	*FT*	58.3 ± 1.9	71.6 ± 0.9	61.6 ± 1.9	73.5 ± 0.8	62.0 ± 1.8	70.6 ± 0.9
MTL (Cosine)	*SS*	58.2 ± 1.8	74.6 ± 0.8	66.1 ± 1.8	79.7 ± 0.9	67.1 ± 1.8	80.0 ± 0.8
MAML [FAL17] (Cosine)	*FT*	***59.8*** ± 1.8	72.8 ± 0.9	59.9 ± 1.9	76.5 ± 0.7	65.1 ± 1.9	78.2 ± 0.8

Table 7.4: The 5-way, 1-shot and 5-shot classification accuracy (%) on *mini*ImageNet and *tiered*ImageNet datasets. "meta-batch" and "ResNet-12 (pre)" are used. "(*tiered*Pre)" means the pre-training stage is finished on the *tiered*ImageNet. We implement the public code of related methods [FAL17, SSZ17, VBL+16, SYZ+18, CLK+19] in our framework by which we are able to conduct different meta operations, i.e. *FT* $[\Theta; \theta]$ and *SS* $[\Theta; \theta]$. The best and ***second best*** results are highlighted in each block. Note that (1) cosine classifiers have been used in MatchingNets [SSZ17] and Baseline++ [CLK+19] for few-shot classification; and (2) MAML in this table is not exactly the same with original MAML [FAL17], as it works on deep neural networks and does not update convolutional layers during base-training.

Method	*mini*ImageNet		*tiered*ImageNet		*mini*ImageNet w/$\mathcal{D}$		*tiered*ImageNet w/$\mathcal{D}$	
	1-shot	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Masked Soft k-Means [RTR+18]	50.7 ± 0.3	64.4 ± 0.2	52.4 ± 0.4	69.9 ± 0.2	49.0 ± 0.3	63.0 ± 0.1	51.4 ± 0.4	69.1 ± 0.3
Masked Soft k-Means *w/* MTL	58.6 ± 1.8	72.2 ± 0.8	65.1 ± 0.8	80.1 ± 0.8	57.2 ± 1.8	71.0 ± 0.8	63.5 ± 1.8	80.2 ± 0.8
TPN [LLP+19]	52.8 ± 0.3	66.4 ± 0.2	55.7 ± 0.3	71.0 ± 0.2	50.4 ± 0.8	64.9 ± 0.7	53.5 ± 0.9	69.9 ± 0.8
TPN *w/* MTL	59.6 ± 1.8	72.3 ± 0.8	68.3 ± 1.9	80.4 ± 0.8	59.1 ± 1.8	71.0 ± 0.8	67.7 ± 1.9	80.3 ± 0.8
LST [LSL+19] *w/o* MTL	64.7 ± 1.9	74.8 ± 0.8	73.3 ± 1.6	82.9 ± 0.8	59.8 ± 1.9	74.8 ± 0.8	70.2 ± 1.6	81.9 ± 0.8
LST *w/* MTL	70.1 ± 1.9	78.7 ± 0.8	77.7 ± 1.6	85.2 ± 0.8	64.1 ± 1.9	77.4 ± 0.8	73.5 ± 1.6	83.4 ± 0.8

Table 7.5: Semi-supervised 5-way, 1-shot and 5-shot classification accuracy (%) on *mini* and *tiered*. "meta-batch" and "ResNet-12 (pre)" are used. "w/$\mathcal{D}$" means additionally including the unlabeled data from 3 distracting classes (5 unlabeled samples per class) that are **excluded** in the "5-way" classes of the task [LLP+19, RTR+18, LSL+19].

7.4.4 Plug-in evaluation

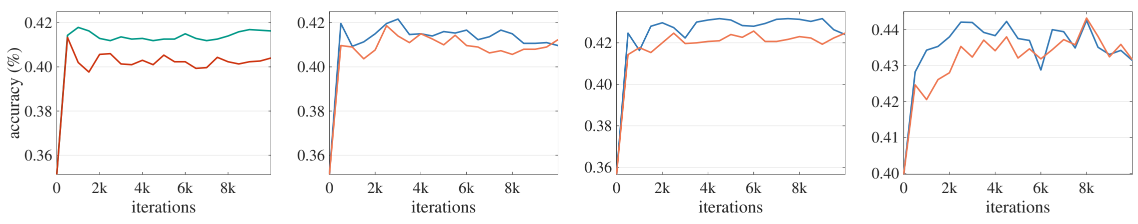
The *Scaling* and *Shifting* (*SS*) operations in our proposed MTL approach work on pre-trained convolutional neurons thus are easy to be applied to other CNN-based few-shot learning models. Detailed plug-in steps are given in Section 7.3.6.

Table 7.4 shows the results of implementing *SS* operations on supervised models, i.e. ProtoNets [SSZ17], MatchingNets [VBL+16], RelationNets [SYZ+18], MAML [FAL17] (with a single FC layer as the base-learner [SLCS19]), and MAML [FAL17] (with a cosine distance classifier as the base-learner [VBL+16, CLK+19]). In their original methods, *FT* is the meta-level operation and 4CONV is the uniform architecture. For an easy and fair comparison, we also implement the results of using *FT* on deeper networks (e.g. ResNet-12). Note that more results of using ResNet-18 are provided in the Appendix. Regarding the table, we have three columns. "*mini*ImageNet (*tiered* Pre)" denotes that the model is pre-trained on *tiered*ImageNet and its weights are then meta-transferred to the learning of few-shot models on *mini*ImageNet episodes. We can see that in all settings, (1) the best performance is achieved by our proposed MTL, e.g., MTL (FC) outperforms MAML (FC) by 3.6% and 8.1% on *tiered*ImageNet 1-shot and 5-shot, respectively; and (2) classical methods using *SS* get consistent improvements over the original version of using *FT*, e.g., RelationNets [SYZ+18] gains 3.4% and 1.5% on *tiered*ImageNet 1-shot and 5-shot, respectively.

In addition, we verify the generalization ability of our MTL to semi-supervised few-shot learning (SSFSL) methods [LLP+19, RTR+18, LSL+19]. Our results on the *mini*ImageNet and *tiered*ImageNet datasets are presented in Table 7.5. Note that "w/$\mathcal{D}$" indicates the more challenging setting of including 3 distracting classes in the unlabeled set (see Section 7.4.1). From Table 7.5, we can see that three models "*w/* MTL" obtain consistent improvements (over their originals) by quite large margins, e.g. the highest as 14.2% on *tiered*ImageNet 1-shot w/$\mathcal{D}$.

Settings	*mini*ImageNet		*mini*ImageNet (tieredPre)		FC100		FC100 (tieredPre)	
	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
update $[\Theta;\theta]$	45.3 ± 1.9	64.6 ± 0.9	54.4 ± 1.8	73.7 ± 0.8	38.4 ± 1.8	52.6 ± 0.9	37.6 ± 1.9	52.7 ± 0.9
update θ	50.0 ± 1.8	66.7 ± 0.9	51.3 ± 1.8	70.3 ± 0.8	39.3 ± 1.9	51.8 ± 0.9	38.3 ± 1.8	52.9 ± 1.0
FT θ	55.9 ± 1.9	71.4 ± 0.9	61.6 ± 1.8	73.5 ± 0.9	41.6 ± 1.9	54.9 ± 1.0	40.4 ± 1.9	54.7 ± 0.9
FT $[\Theta4;\theta]$	57.2 ± 1.8	71.6 ± 0.8	62.3 ± 1.8	73.9 ± 0.9	40.9 ± 1.8	54.3 ± 1.0	41.2 ± 1.8	53.6 ± 1.0
FT $[\Theta3,\Theta4;\theta]$	58.1 ± 1.8	70.9 ± 0.8	63.0 ± 1.8	74.8 ± 0.9	41.5 ± 1.8	53.7 ± 0.9	40.7 ± 1.9	53.8 ± 0.9
FT $[\Theta;\theta]$	58.3 ± 1.8	71.6 ± 0.8	63.2 ± 1.9	75.7 ± 0.8	41.6 ± 1.9	54.4 ± 1.0	41.1 ± 1.9	54.5 ± 0.9
SS $[\Theta4;\theta]$	59.2 ± 1.8	73.1 ± 0.9	64.0 ± 1.8	76.9 ± 0.8	42.4 ± 1.9	55.1 ± 1.0	42.7 ± 1.9	55.9 ± 1.0
SS $[\Theta3,\Theta4;\theta]$	59.4 ± 1.8	73.4 ± 0.8	64.5 ± 1.8	77.2 ± 0.8	42.5 ± 1.9	54.5 ± 1.0	43.4 ± 1.8	56.4 ± 1.0
SS $[\Theta;\theta]$	***60.6*** ± 1.8	***74.3*** ± 0.8	***65.7*** ± 1.8	***78.4*** ± 0.9	43.6 ± 1.9	55.4 ± 1.0	43.5 ± 1.9	***57.1*** ± 1.0
SS $[\Theta;\theta]$, *regularized*	61.0 ± 1.8	74.5 ± 0.8	66.2 ± 1.8	79.1 ± 0.8	***43.5*** ± 1.7	***55.3*** ± 0.8	43.5 ± 1.9	57.2 ± 0.8

Table 7.6: The 5-way, 1-shot and 5-shot classification accuracy (%) using ablative models, on two datasets. "meta-batch" and "ResNet-12 (pre)" are used. "(tieredPre)" means the pre-training stage is finished on the *tiered*ImageNet. The best and ***second best*** results are highlighted.



(a) *FT*, ResNet-12 (b) *SS*, ResNet-12 (c) *SS*, ResNet-18 (d) *SS*, ResNet-25

Figure 7.4: The 5-way, 1-shot meta-validation accuracy plots on the FC100, using *FT* (pre-trained ResNet-12 and MAML [FAL17]) and our MTL on different pre-trained networks. Red curve uses the original meta-batch [FAL17] and others use our proposed HT meta-batch.

7.4.5 Ablation study

Table 7.6 shows the results of ablation studies on the *mini*ImageNet and FC100. Figure 7.4 demonstrates the performance gap between *w/* and *w/o* HT meta-batch in terms of recognition accuracy and converging speed on the FC100. Table 7.7 summarizes the accuracies of *w/* and *w/o* HT meta-batch on ImageNet-based datasets.

MTL *vs.* No meta-learning. Table 7.6 shows the results of *No meta-learning* methods on the top block. Compared to these, our approach achieves significantly better performance, e.g., the largest margins on *mini*ImageNet are 11.0% for 1-shot and 7.8% for 5-shot. This validates the effectiveness of meta-learning method for tackling few-shot learning problems. Between two *No meta-learning* methods, we can see that updating both feature extractor Θ and classifier θ is inferior to updating θ only (Θ is pre-trained), e.g., around 5% reduction on *mini*ImageNet 1-shot. One reason is that in few-shot settings, there are too many parameters to optimize with few-shot data. This is our motivation to learn only θ during base-learning (see Table 7.1).

Setting	*mini*ImageNet		*tiered*ImageNet	
	1-**shot**	5-**shot**	1-**shot**	5-**shot**
FT $[\Theta;\theta]$	58.3 ± 1.9	71.6 ± 0.9	61.6 ± 1.9	73.5 ± 0.8
FT $[\Theta;\theta]$, HT	58.7 ± 1.8	73.2 ± 0.8	64.7 ± 1.7	78.5 ± 0.8
SS $[\Theta;\theta]$	60.6 ± 1.9	74.3 ± 0.8	65.6 ± 1.7	78.7 ± 0.9
SS $[\Theta;\theta]$, HT	61.4 ± 1.8	75.9 ± 0.8	65.3 ± 1.8	81.2 ± 0.8

Table 7.7: Ablation results for the 5-way, 1-shot and 5-shot classification accuracy (%) on *mini*ImageNet and *tiered*ImageNet datasets. "ResNet-12 (pre)" is used.

SS $[\Theta;\theta]$ **works better than light-weight *FT* variants.** Table 7.6 shows that our approach with *SS* $[\Theta;\theta]$ achieves the best performances for all few-shot settings. *SS* actually meta-learns a smaller set of transferring parameters on $[\Theta;\theta]$ than *FT*. People may argue that *FT* is weaker because it learns a larger set of initialization parameters, whose quantity is equal to the size of $[\Theta;\theta]$, causing the model to overfit to few-shot data. In the middle block of Table 7.6, we show the ablation study of freezing low-level pre-trained layers and meta-learn only the high-level layers (e.g. Θ4 of ResNet-12) by *FT* operations. It is obvious that they all yield inferior performances than using our *SS*. An additional observation is that our methods *SS** perform consistently better than *FT**.

Meta-gradient regularization is effective. On the last two rows of Table 7.6, we validate the effectiveness of meta-gradient regularization (see Section 7.3.4). From the results, we can see deploying such cross-task memory regularization (*SS* $[\Theta;\theta]$, *regularized*) achieves better performance than using *SS* $[\Theta;\theta]$ whose meta gradients each time come from an individual episode. This is because our regularization forces meta-learner to be less forgetting about previous episodes, and additionally stabilize the meta-gradients of each few-shot episode.

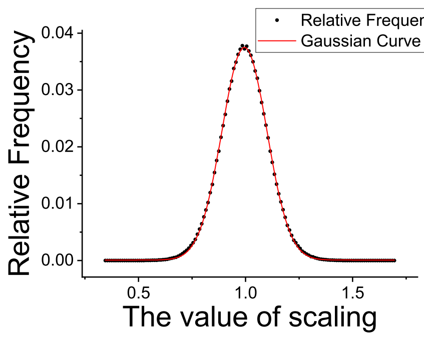
Accuracy gain by HT meta-batch. HT meta-batch is basically a curriculum learning scheme, and can be generalized to the models with different network architectures. In Table 7.7, we show the ablation results for HT meta-batch on ImageNet-based datasets. Comparing *SS* $[\Theta;\theta]$, HT (HT meta-batch) with *SS* $[\Theta;\theta]$, HT meta-batch improves the results by an average accuracy of 1.5%. In Figure 7.4, we show the validation curves including the cases of using *FT* as well as *SS* during meta-training on the FC100 dataset. Red curves are the results of using conventional meta-batch [FAL17]. It is clear that HT meta-batch boosts both *FT* and *SS* based meta-learners.

Speed of convergence of MTL with HT meta-batch. From Figure 7.4 (a)-(d), we can see that impressively, the models using our proposed HT meta-batch require only $1 \sim 4k$ episodes to converge to a good performance. Note that (1) each iteration contains 2 training episodes, and (2) MAML [FAL17] without deep pre-trained networks used over $200k$ episodes to achieve the best performance on miniImageNet. We attest to this for three reasons. First, our methods start from the pre-trained deep neural networks. Second, our *SS* needs to learn only $< \frac{2}{9}$ parameters of the number of *FT* parameters. Third, our HT meta-batch is a hard negative mining step and brings accelerations by learning challenging tasks [SGG16].

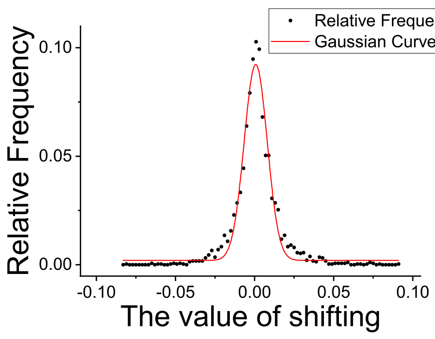
7.4.6 Statistical data of *SS*

We evaluate to what extent neuron weights and biases (e.g., on ResNet-12) have drifted after *SS* operations. We present the statistics on the learned *SS* weights in Table 7.8. Each number

(a) *mini*ImageNet



(b) *mini*ImageNet



		***mini*ImageNet**	***tiered*ImageNet**	**FC100**
Scaling	y_0	9.58×10^{-5}	1.95×10^{-4}	8.01×10^{-5}
	x_c	9.94×10^{-1}	9.98×10^{-1}	9.93×10^{-1}
	w	2.11×10^{-1}	1.49×10^{-1}	2.27×10^{-1}
	A	9.87×10^{-3}	9.79×10^{-3}	9.90×10^{-3}
Shifting	y_0	1.99×10^{-3}	1.32×10^{-3}	1.30×10^{-3}
	x_c	7.52×10^{-4}	7.61×10^{-4}	6.28×10^{-4}
	w	1.46×10^{-2}	1.53×10^{-2}	1.70×10^{-2}
	A	1.65×10^{-3}	1.73×10^{-3}	1.71×10^{-3}

Table 7.8: Statistical values of *SS* parameters, i.e. to see how much network parameters drifted after the meta-training using *SS*. The experiments are conducted with the settings of ResNet-12, meta-batch, 5-way and 1-shot. *Scaling* and *Shifting* parameters are counted with bin size 0.01 and 0.002, respectively. Relative frequency of each *SS* value is computed. All dots match a fit to the Gaussian distribution ($y = y_0 + \frac{A}{w\sqrt{\pi/2}} e^{-2(\frac{x-x_c}{w})^2}$). x_c and w are the values of mean and standard deviation, respectively. y_0 and A are two parameters of the distribution to enable the exact fit.

in the table shows how much the weight (or bias) drifts from the original weight (or bias) pre-trained using large-scale data. Each dot curve in (a) and (b) presents the distribution of those numbers, matching well with the Gaussian distribution (in red).

We find that *Scaling* parameters are more scattered than *Shifting* on three datasets. The average shift of mean values x_c of *Scaling* is 4.51×10^{-3} higher than that of *Shifting* 7.13×10^{-4} (note that initialization for *Scaling* parameter is 1 and for *Shifting* is 0). The standard deviation w shows also higher for *Scaling*. We think these are due to the fact that convolution neuron weights (rather than neuron biases) encode the most of image representation knowledge. We also see the differences among three datasets: the parameter drifting is more obvious on smaller datasets such as the FC100. In other words, the gap between pre-trained model and meta-learner is more significant on such dataset. We think this is because the feature representations learned on small-size data are not as generalizable as those learned on larger-scale data.

7.5 CONCLUSION

In this chapter, we show that our novel MTL model trained with HT meta-batch learning curriculum achieves the top performance for tackling few-shot learning problems. The key operations of MTL on pre-trained DNN neurons proved to be highly efficient for adapting the learning experience to the unseen task. The superiority was particularly achieved in the extreme 1-shot cases on three challenging benchmarks – *mini*ImageNet, *tiered*ImageNet, and FC100. The generalization ability of our method is validated by implementing MTL on the classical supervised few-shot models as well as the state-of-the-art semi-supervised few-shot models. The consistent improvements by MTL prove that large-scale pre-trained deep networks can offer a good "knowledge base" to conduct efficient few-shot learning on. In terms of learning scheme, HT meta-batch showed consistently good performance for the ablative models. On the more challenging FC100 benchmark, it showed to be particularly helpful for boosting convergence speed. This design is independent of any specific model or architecture and can be generalized well whenever the hardness of the task is easy to evaluate in online iterations.

8 LEARNING TO ENSEMBLE DEEP MODELS

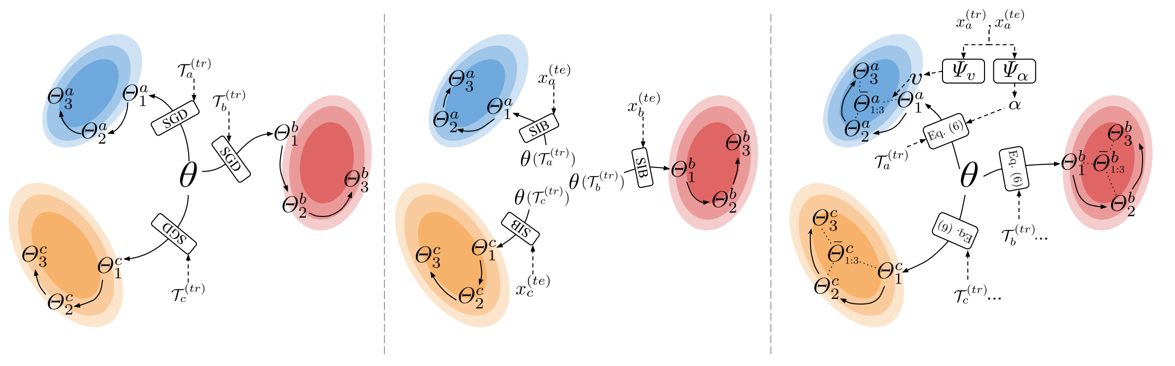
Contents

The lack of training data in few-shot learning leads to poor models that perform high-variance or low-confidence predictions. In this chapter, we propose to meta-learn the ensemble of epoch-wise empirical Bayes models (E^{3}BM) to achieve robust predictions. "Epoch-wise" means that each training epoch has a Bayes model whose parameters are specifically learned and deployed. "Empirical" means that the hyperparameters, e.g., used for learning and ensembling the epoch-wise models, are generated by hyperprior learners conditional on task-specific data. We introduce four kinds of hyperprior learners by considering inductive *vs.* transductive, and epoch-dependent *vs.* epoch-independent, in the paradigm of meta-learning. We conduct extensive experiments for five-class few-shot tasks on three challenging benchmarks: *mini*ImageNet, *tiered*ImageNet, and FC100, and achieve top performance using the epoch-dependent transductive hyperprior learner, which captures the richest information. Our ablation study shows that both "epoch-wise ensemble" and "empirical" encourage high efficiency and robustness in the model performance.

This chapter is based on [LSS20]. As the first author of [LSS20], Yaoyao Liu conducted all experiments and was the main writer. This work received more than 80 citations and is used as the baseline framework for many top-tier conference papers [XL22, SXH$^+$21, LSA21].

8.1 INTRODUCTION

The ability to learn new concepts from a handful of examples is well-handled by humans, while in contrast, it remains challenging for machine models whose typical training requires



(a) MAML [13] (b) SIB [25] (c) E^{3}BM (ours)

Figure 8.1: Conceptual illustrations of the model adaptation on the **blue**, **red** and **yellow** tasks. (a) MAML [FAL17] is the classical inductive method that meta-learns a network initialization θ that is used to learn a single base-learner on each task, e.g., Θ_3^a in the blue task. (b) SIB [HMX$^+$20] is a transductive method that formulates a variational posterior as a function of both labeled training data $\mathcal{T}^{(tr)}$ and unlabeled test data $x^{(te)}$. It also uses a single base-learner and optimizes the learner by running several synthetic gradient steps on $x^{(te)}$. (c) Our E^{3}BM is a generic method that learns to combine the epoch-wise base-learners (e.g., Θ_1, Θ_2, and Θ_3), and to generate task-specific learning rates α and combination weights v that encourage robust adaptation. $\bar{\Theta}_{1:3}$ denotes the ensemble result of three base-learners; Ψ_α and Ψ_v denote the hyperprior learners learned to generate α and v, respectively. Note that figure (c) is based on E^{3}BM+MAML, i.e., plug-in our E^{3}BM to MAML baseline. Other plug-in versions are introduced in Sec. 8.3.5.

a significant amount of data for good performance [KSH12]. However, in many real-world applications, we have to face the situations of lacking a significant amount of training data, as e.g., in the medical domain. It is thus desirable to improve machine learning models to handle few-shot settings where each new concept has very scarce examples [LFP06, FAL17, SLCS19, JL20].

Meta-learning methods aim to tackle the few-shot learning problem by transferring experience from similar few-shot tasks [Car95]. There are different meta strategies, among which the gradient descent based methods are particularly promising for today's neural networks [FAL17, FXL18, GFL$^+$18, FFS$^+$18, LC18, ZCG$^+$18, SLCS19, AES19, ZLL$^+$19b, HMX$^+$20, ZLL$^+$19a, ZCLS20, WHD$^+$20]. These methods follow a unified meta-learning procedure that contains two loops. The inner loop learns a base-learner for each individual task, and the outer loop uses the validation loss of the base-learner to optimize a meta-learner. In previous works [FAL17, FXL18, AES19, SLCS19], the task of the meta-learner is to initialize the base-learner for the fast and efficient adaptation to the few training samples in the new task.

In this chapter, we aim to address two shortcomings of the previous works. First, the learning process of a base-learner for few-shot tasks is quite unstable [AES19], and often results in high-variance or low-confidence predictions. An intuitive solution is to train an ensemble of models and use the combined prediction, which should be more robust [Bre96, OV12, JBvdL18]. However, it is not obvious how to obtain and combine multiple base-learners given the fact that a very limited number of training examples are available. Rather than learning multiple independent base-learners [YKD+18], we propose a novel method of utilizing the sequence of epoch-wise base-learners (while training a single base-learner) as the ensemble. Second, it is well-known that the values of hyperparameters, e.g., for initializing and updating models, are critical for best performance, and are particularly important for few-shot learning. In order to explore the optimal hyperparameters, we propose to employ the empirical Bayes method in the paradigm of meta-learning. In specific, we meta-learn hyperprior learners with meta-training tasks, and use them to generate task-specific hyperparameters, e.g., for updating and ensembling multiple base-learners. We call the resulting novel approach **E^{3}BM**, which learns the **E**nsemble of **E**poch-wise **E**mpirical **B**ayes **M**odels for each few-shot task. Our "epoch-wise models" are *different models* since each one of them is resulted from a specific training epoch and is trained with a specific set of hyperparameter values. During test, E^{3}BM combines the ensemble of models' predictions with soft ensembling weights to produce more robust results. In this chapter, we argue that during model adaptation to the few-shot tasks, the most active adapting behaviors actually happen in the early epochs, and then converge to and even overfit to the training data in later epochs. Related works use the single base-learner obtained from the last epoch, so their meta-learners learn only partial adaptation experience [FAL17, SLCS19, FXL18, HMX+20]. In contrast, our E^{3}BM leverages an ensemble modeling strategy that adapts base-learners at different epochs and each of them has task-specific hyperparameters for updating and ensembling. It thus obtains the optimized combinational adaptation experience. Figure 8.1 presents the conceptual illustration of E^{3}BM, compared to those of the classical method MAML [FAL17] and the state-of-the-art SIB [HMX+20].

Our main contributions are three-fold. (1) A novel few-shot learning approach E^{3}BM that learns to learn and combines an ensemble of epoch-wise Bayes models for more robust few-shot learning. (2) Novel hyperprior learners in E^{3}BM to generate the task-specific hyperparameters for learning and combining epoch-wise Bayes models. In particular, we introduce four kinds of hyperprior learner by considering inductive [FAL17, SLCS19] and transductive learning methods [HMX+20], and each with either epoch-dependent (e.g., LSTM) or epoch-independent (e.g., epoch-wise FC layer) architectures. (3) Extensive experiments on three challenging few-shot benchmarks, *mini*ImageNet [VBL+16], *tiered*ImageNet [RTR+18] and Fewshot-CIFAR100 (FC100) [ORL18]. We plug in our E^{3}BM to the state-of-the-art few-shot learning methods [FAL17, SLCS19, HMX+20] and obtain consistent performance boosts. We conduct extensive model comparison and observe that our E^{3}BM employing an epoch-dependent transductive hyperprior learner achieves the top performance on all benchmarks.

8.2 RELATED WORKS

In this section, we discuss related works on hyperparameter optimization and ensemble modeling. We will not repeat the incremental learning works that have been discussed in Chapter 2.

Hyperparameter optimization. Building a model for a new task is a process of exploration-exploitation. Exploring suitable architectures and hyperparameters are important before training. Traditional methods are model-free, e.g., based on grid search [BB12, LJD+17, JDO+17]. They require multiple full training trials and are thus costly. Model-based hyperparameter optimization methods are adaptive but sophisticated, e.g., using random forests [HHL11], Gaussian processes [SLA12] and input warped Gaussian processes [SSZA14] or scalable Bayesian optimization [SRS+15]. In our approach, we meta-learn a hyperprior learner to output optimal hyperparameters by gradient descent, without additional manual labor. Related methods using gradient descent mostly work for single model learning in an inductive way [Ben00, Dom12, MDA15, LRBG16, FFS+18, MMCSD19, LZCL17, LSL+20]. While, our hyperprior learner generates a sequence of hyperparameters for multiple models, in either the inductive or the transductive learning manner.

Ensemble modeling. It is a strategy [HLP+17, ZSC+19] to use multiple algorithms to improve machine learning performance, and which is proved to be effective to reduce the problems related to overfitting [KW03, SK96]. Mitchell et al. [Mit97] provided a theoretical explanation for it. Boosting is one classical way to build an ensemble, e.g., AdaBoost [FS97] and Gradient Tree Boosting [Fri02]. Stacking combines multiple models by learning a combiner and it applies to both tasks in supervised learning [Bre96, OV12, JBvdL18] and unsupervised learning [SW99]. Bootstrap aggregating (i.e., Bagging) builds an ensemble of models through parallel training [Bre96], e.g., random forests [Ho95]. The ensemble can also be built on a temporal sequence of models [LA17]. Some recent works have applied ensemble modeling to few-shot learning. Yoon et al. proposed Bayesian MAML (BMAML) that trains multiple instances of base-model to reduce mete-level overfitting [YKD+18]. The most recent work [DSM19] encourages multiple networks to cooperate while keeping predictive diversity. Its networks are trained with carefully-designed penalty functions, different from our automated method using empirical Bayes. Besides, its method needs to train much more network parameters than ours. Detailed comparisons are given in the experiment section.

8.3 AN ENSEMBLE OF EPOCH-WISE EMPIRICAL BAYES MODELS

As shown in Fig. 8.2, E^{3}BM trains a sequence of epoch-wise base-learners $\{\Theta_m\}$ with training data $\mathcal{T}^{(tr)}$ and learns to combine their predictions $\{z_m^{(te)}\}$ on test data $x^{(te)}$ for the best performance. This ensembling strategy achieves more robustness during prediction. The hyperparameters of each base-learner, i.e., learning rates α and combination weights v, are generated by the hyperprior learners conditional on task-specific data, e.g., $x^{(tr)}$ and $x^{(te)}$. This approach encourages the high diversity and informativeness of the ensembling models.

8.3.1 Denotations

In this section, we introduce the unified episodic formulation of few-shot learning, following [VBL+16, RL17, FAL17]. This formulation was proposed for few-shot classification first in [VBL+16]. Its problem definition is different from traditional classification in three aspects: (1) the main phases are not training and test but meta-training and meta-test, each of which includes training and test; (2) the samples in meta-training and meta-testing are not data points but episodes, i.e. few-shot classification tasks; and (3) the objective is not classifying unseen data points but to fast adapt the meta-learned knowledge to the learning of new tasks.

Given a dataset $\mathcal{D}$ for meta-training, we first sample few-shot episodes (tasks) $\{\mathcal{T}\}$ from a task distribution $p(\mathcal{T})$ such that each episode $\mathcal{T}$ contains a few samples of a few classes, e.g., 5 classes and 1 shot per class. Each episode $\mathcal{T}$ includes a training split $\mathcal{T}^{(tr)}$ to optimize a specific base-learner, and a test split $\mathcal{T}^{(te)}$ to compute a generalization loss to optimize a global meta-learner. For meta-test, given an unseen dataset $\mathcal{D}_{un}$ (i.e., samples are from

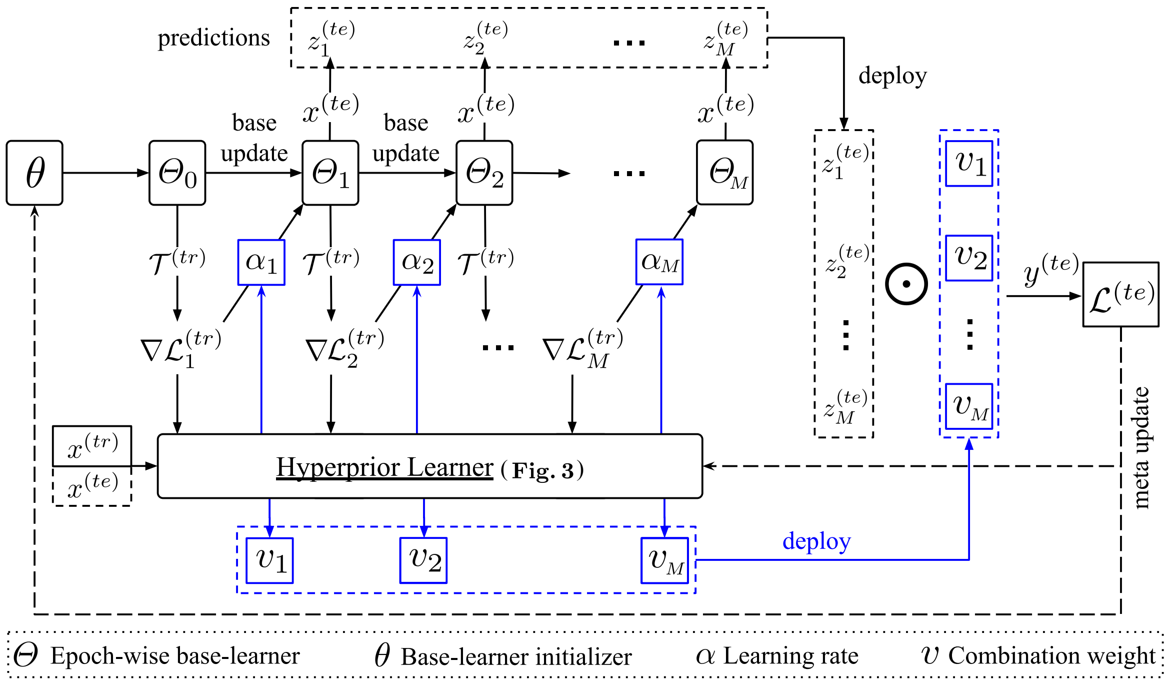


Figure 8.2: The computing flow of the proposed E^{3}BM approach in one meta-training episode. For the meta-test task, the computation will be ended with predictions. Hyper-learner predicts task-specific hyperparameters, i.e., learning rates and multi-model combination weights. When its input contains $x^{(te)}$, it is transductive, otherwise inductive. Its detailed architecture is given in Fig. 8.3.

unseen classes), we sample a test task $\mathcal{T}_{un}$ to have the same-size training/test splits. We first initiate a new model with meta-learned network parameters (output from our hyperprior learner), then train this model on the training split $\mathcal{T}_{un}^{(tr)}$. We finally evaluate the performance on the test split $\mathcal{T}_{un}^{(te)}$. If we have multiple tasks, we report average accuracy as the final result.

8.3.2 Empirical Bayes method

Our approach can be formulated as an empirical Bayes method that learns two levels of models for a few-shot task. The first level has hyperprior learners that generate hyperparameters for updating and combining the second-level models. More specifically, these second-level models are trained with the loss derived from the combination of their predictions on training data. After that, their loss of test data is used to optimize the hyperprior learners. This process is also called meta update. See the dashed arrows in Fig. 8.2.

In specific, we sample K episodes $\{\mathcal{T}_k\}_{k=1}^{K}$ from the meta-training data $\mathcal{D}$. Let Θ denote base-learner and ψ represent its hyperparameters. An episode $\mathcal{T}_k$ aims to train Θ to recognize different concepts, so we consider using concept-related (task specific) data for customizing the Θ through a hyperprior $p(\psi_k)$. To achieve this, we first formulate the empirical Bayes method with marginal likelihood according to hierarchical structure among data as follows,

$$p(\mathcal{T}) = \prod_{k=1}^{K} p(\mathcal{T}_k) = \prod_{k=1}^{K} \int_{\psi_k} p(\mathcal{T}_k|\psi_k)p(\psi_k)d\psi_k. \tag{8.1}$$

Then, we use variational inference [HBWP13] to estimate $\{p(\psi_k)\}_{k=1}^{K}$. We parametrize distribution $q_{\varphi_k}(\psi_k)$ with φ_k for each $p(\psi_k)$, and update φ_k to increase the similarity betweeen $q_{\varphi_k}(\psi_k)$ and $p(\psi_k)$. As in standard probabilistic modeling, we derive an evidence lower bound on the log version of Eq. (8.1) to update φ_k,

$$\log p(\mathcal{T}) \geqslant \sum_{k=1}^{K} \Big[\mathbb{E}_{\psi_k \sim q_{\varphi_k}}\big[\log p(\mathcal{T}_k|\psi_k)\big] - D_{\mathrm{KL}}(q_{\varphi_k}(\psi_k)||p(\psi_k))\Big]. \tag{8.2}$$

Therefore, the problem of using $q_{\varphi_k}(\psi_k)$ to approach to the best estimation of $p(\psi_k)$ becomes equivalent to the objective of maximizing the evidence lower bound [BKM17, HBWP13, HMX$^+$20] in Eq. (8.2), with respect to $\{\varphi_k\}_{k=1}^{K}$, as follows,

$$\min_{\{\varphi_k\}_{k=1}^{K}} \frac{1}{K} \sum_{k=1}^{K} \Big[\mathbb{E}_{\psi_k \sim q_{\varphi_k}}\big[-\log p(\mathcal{T}_k|\psi_k)\big] + D_{\mathrm{KL}}(q_{\varphi_k}(\psi_k)||p(\psi_k))\Big]. \tag{8.3}$$

To improve the robustness of few-shot models, existing methods sample a significant amount number of episodes during meta-training [FAL17, SLCS19]. Each episode employing its own hyperprior $p(\psi_k)$ causes a huge computation burden, making it difficult to solve the aforementioned optimization problem. To tackle this, we leverage a technique called "amortized variational inference" [KW14, RMW14, HMX$^+$20]. We parameterize the KL term in $\{\varphi_k\}_{k=1}^{K}$ (see Eq. (8.3)) with a unified deep neural network $\Psi(\cdot)$ taking $x_k^{(tr)}$ (inductive learning) or $\{x_k^{(tr)}, x_k^{(te)}\}$ (transductive learning) as inputs, where $x_k^{(tr)}$ and $x_k^{(te)}$ respectively denote the training and test samples in the k-th episode. In this chapter, we call $\Psi(\cdot)$ hyperprior learner. As shown in Fig. 8.3, we additionally feed the hyperprior learner with the training gradients $\nabla\mathcal{L}_{\Theta}(\mathcal{T}_k^{(tr)})$ to $\Psi(\cdot)$ to encourage it to "consider" the current state of the training epoch. We mentioned in

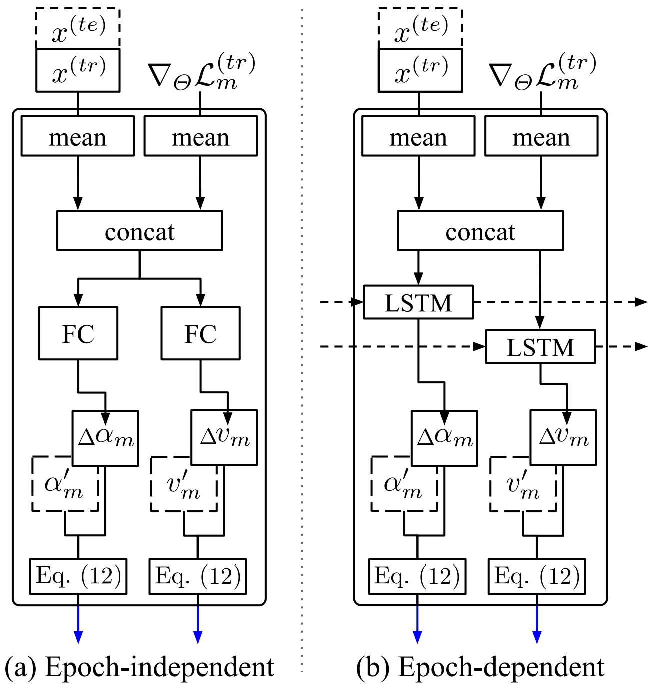


Figure 8.3: Two options of hyperprior learner at the m-th base update epoch. In terms of the mapping function, we deploy either FC layers to build epoch-independent hyperprior learners, or LSTM to build an epoch-dependent learner. Values in dashed box were learned from previous tasks.

Sec. 8.1 that base-learners at differ-
ent epochs are adapted differently, so we expect the corresponding hyperprior learner to "observe" and "utilize" this information to produce effective hyperparameters. By replacing q_{φ_k} with $q_{\Psi(\cdot)}$, Problem (8.3) can be rewritten as:

$$\min_{\Psi} \frac{1}{K} \sum_{k=1}^{K} \Big[\mathbb{E}_{\psi_k \sim q_{\Psi(\cdot)}} \big[-\log p(\mathcal{T}_k|\psi_k) \big] + D_{\mathrm{KL}}(q_{\Psi(\cdot)}(\psi_k)||p(\psi_k)) \Big]. \tag{8.4}$$

Then, we solve Problem (8.4) by optimizing $\Psi(\cdot)$ with the meta gradient descent method used in classical meta-learning paradigms [FAL17, SLCS19, HMX$^+$20]. We elaborate the details of learning $\{\Theta_m\}$ and meta-learning $\Psi(\cdot)$ in the following sections.

8.3.3 Learning the ensemble of base-learners

Previous works have shown that training multiple instances of the base-learner is helpful to achieve robust few-shot learning [YKD$^+$18, DSM19]. However, they suffer from the computational burden of optimizing multiple copies of neural networks in parallel, and are not easy to generalize to deeper neural architectures. If include the computation of second-order derivatives in meta gradient descent [FAL17], this burden becomes more unaffordable. In contrast, our approach is free from this problem, because it is built on top of optimization-based meta-learning models, e.g., MAML [FAL17], MTL [SLCS19], and SIB [HMX$^+$20], which naturally produce a sequence of models along the training epochs in each episode.

Given an episode $\mathcal{T} = \{\mathcal{T}^{(tr)}, \mathcal{T}^{(te)}\} = \{\{x^{(tr)}, y^{(tr)}\}, \{x^{(te)}, y^{(te)}\}\}$, let Θ_m denote the parameters of the base-learner working at epoch m (w.r.t. m-th base-learner or BL-m), with $m \in \{1, ..., M\}$. Basically, we initiate BL-1 with parameters θ (network weights and bias) and hyperparameters (e.g., learning rate α), where θ is meta-optimized as in MAML [FAL17], and α is generated by the proposed hyperprior learner Ψ_α. We then adapt BL-1 with normal gradient descent on the training set $\mathcal{T}^{(tr)}$, and use the adapted weights and bias to initialize BL-2. The general process is thus as follows,

$$\Theta_0 \leftarrow \theta, \tag{8.5}$$

$$\Theta_m \leftarrow \Theta_{m-1} - \alpha_m \nabla_\Theta \mathcal{L}_m^{(tr)} = \Theta_{m-1} - \Psi_\alpha(\tau, \nabla_\Theta \mathcal{L}_m^{(tr)}) \nabla_\Theta \mathcal{L}_m^{(tr)}, \tag{8.6}$$

where α_m is the learning rate outputted from Ψ_α, and $\nabla_\Theta \mathcal{L}_m^{(tr)}$ are the derivatives of the training loss, i.e, gradients. τ represents either $x^{(tr)}$ in the inductive setting, or $\{x^{(tr)}, x^{(te)}\}$ in the transductive setting. Note that Θ_0 is introduced to make the notation consistent, and a subscript m is omitted from Ψ_α for conciseness. Let $F(x; \Theta_m)$ denote the prediction scores of input x, so the base-training loss $\mathcal{T}^{(tr)} = \{x^{(tr)}, y^{(tr)}\}$ can be unfolded as,

$$\mathcal{L}_m^{(tr)} = L_{ce}\big(F(x^{(tr)}; \Theta_{m-1}), y^{(tr)}\big), \tag{8.7}$$

where L_{ce} is the softmax cross entropy loss. During the episode test, each base-learner BL-m infers the prediction scores z_m for test samples $x^{(te)}$,

$$z_m = F(x^{(te)}; \Theta_m). \tag{8.8}$$

Assume the hyperprior learner Ψ_v generates the combination weight v_m for BL-m. The final prediction score is initialized as $\hat{y}_1^{(te)} = v_1 z_1$. For the m-th base epoch, the prediction z_m will be calculated and added to $\hat{y}^{(te)}$ as follows,

$$\hat{y}_m^{(te)} \leftarrow v_m z_m + \hat{y}_{m-1}^{(te)} = \Psi_v(\tau, \nabla_\Theta \mathcal{L}_m^{(tr)}) F(x^{(te)}; \Theta_m) + \hat{y}_{m-1}^{(te)}. \tag{8.9}$$

In this way, we can update prediction scores without storing base-learners or feature maps in the memory.

8.3.4 Meta-learning the hyperprior learners

As presented in Fig. 8.3, we introduce two architectures, i.e., LSTM or individual FC layers, for the hyperprior learner. FC layers at different epochs are independent. Using LSTM to "connect" all epochs is expected to "grasp" more task-specific information from the overall training states of the task. In the following, we elaborate the meta-learning details for both designs.

Assume before the k-th episode, we have meta-learned the base learning rates $\{\alpha'_m\}_{m=1}^{M}$ and combination weights $\{v'_m\}_{m=1}^{M}$. Next in the k-th episode, specifically at the m-th epoch as shown in Fig. 8.3, we compute the mean values of τ and $\nabla_{\Theta_m} \mathcal{L}_m^{(tr)}$, respectively, over all samples[1]. We then input the concatenated value to FC or LSTM mapping function as follows,

$$\Delta\alpha_m, \Delta v_m = \text{FC}_m(\text{concat}[\bar{\tau}; \overline{\nabla_{\Theta_m} \mathcal{L}_m^{(tr)}}]), \textbf{ or} \tag{8.10}$$

$$[\Delta\alpha_m, \Delta v_m], h_m = \text{LSTM}(\text{concat}[\bar{\tau}; \overline{\nabla_{\Theta_m} \mathcal{L}_m^{(tr)}}], h_{m-1}), \tag{8.11}$$

where h_m and h_{m-1} are the hidden states at epoch m and epoch $m-1$, respectively. We then use the output values to update hyperparameters as,

$$\alpha_m = \lambda_1 \alpha'_m + (1-\lambda_1)\Delta\alpha, \; v_m = \lambda_2 v'_m + (1-\lambda_2)\Delta v, \tag{8.12}$$

where λ_1 and λ_2 are fixed fractions in $(0, 1)$. Using learning rate α_m, we update BL-$(m-1)$ to be BL-m with Eq. (8.6). After M epochs, we obtain the combination of predictions $\hat{y}_M^{(te)}$ (see Eq. (8.9)) on test samples. In training tasks, we compute the test loss as,

$$\mathcal{L}^{(te)} = L_{ce}(\hat{y}_M^{(te)}, y^{(te)}). \tag{8.13}$$

We use this loss to calculate meta gradients to update Ψ as follows,

$$\Psi_\alpha \leftarrow \Psi_\alpha - \beta_1 \nabla_{\Psi_\alpha} \mathcal{L}^{(te)}, \;\; \Psi_v \leftarrow \Psi_v - \beta_2 \nabla_{\Psi_v} \mathcal{L}^{(te)}, \tag{8.14}$$

where β_1 and β_2 are meta-learning rates that determine the respective stepsizes for updating Ψ_α and Ψ_v. These updates are to back-propagate the test gradients till the input layer, through unrolling all base training gradients of $\Theta_1 \sim \Theta_M$. The process thus involves a gradient through a gradient [FAL17, FXL18, SLCS19]. Computationally, it requires an additional backward pass through $\mathcal{L}^{(tr)}$ to compute Hessian-vector products, which is supported by standard numerical computation libraries such as TensorFlow [AAB+16] and PyTorch [SDC+19].

[1] In the inductive setting, training images are used to compute $\bar{\tau}$; while in the transductive setting, test images are additionally used.

8.3.5 Plugging-in to baseline methods

The optimization of Ψ relies on meta gradient descent method, which was first applied to few-shot learning in MAML [FAL17]. Recently, MTL [SLCS19] showed more efficiency by implementing that method on deeper pre-trained CNNs (e.g., ResNet-12 [SLCS19], and ResNet-25 [SLC$^+$22]). SIB [HMX$^+$20] was built on even deeper and wider networks (WRN-28-10), and it achieved top performance by synthesizing gradients in transductive learning. These three methods are all optimization-based, and use the single base-learner of the last base-training epoch. In the following, we describe how to learn and combine multiple base-learners in MTL, SIB and MAML, respectively, using our E^{3}BM approach.

According to [SLCS19, HMX$^+$20], we pre-train the feature extractor f on a many-shot classification task using the whole set of $\mathcal{D}$. The meta-learner in MTL is called scaling and shifting weights Φ_{SS}, and in SIB is called synthetic information bottleneck network $\phi(\lambda, \xi)$. Besides, there is a common meta-learner called base-learner initializer θ, i.e., the same θ in Fig. 8.2, in both methods. In MAML, the only base-learner is θ and there is no pre-training for its feature extractor f.

Given an episode $\mathcal{T}$, we feed training images $x^{(tr)}$ and test images $x^{(te)}$ to the feature extractor $f \odot \Phi_{SS}$ in MTL (f in SIB and MAML), and obtain the embedding $e^{(tr)}$ and $e^{(te)}$, respectively. Then in MTL, we use $e^{(tr)}$ with labels to train base-learner Θ for M times to get $\{\Theta_m\}_{m=1}^{M}$ with Eq. (8.6). In SIB, we use its multilayer perceptron (MLP) net to synthesize gradients conditional on $e^{(te)}$ to indirectly update $\{\Theta_m\}_{m=1}^{M}$. During these updates, our hyperprior learner Ψ_α derives the learning rates for all epochs. In episode test, we feed $e^{(te)}$ to $\{\Theta_m\}_{m=1}^{M}$ and get the combined prediction $\{z_m\}_{m=1}^{M}$ with Eq. (8.9). Finally, we compute the test loss to meta-update $[\Psi_\alpha; \Psi_v; \Phi_{SS}; \theta]$ in MTL, $[\Psi_\alpha; \Psi_v; \phi(\lambda, \xi); \theta]$ in SIB, and $[f; \theta]$ in MAML. We call the resulting methods MTL+E^{3}BM, SIB+E^{3}BM, and MAML+E^{3}BM, respectively, and demonstrate their improved efficiency over baseline models [SLCS19, HMX$^+$20, FAL17] in experiments.

8.4 EXPERIMENTS

We evaluate our approach in terms of its overall performance and the effects of its two components, i.e. ensembling epoch-wise models and meta-learning hyperprior learners. In the following sections, we introduce the datasets and implementation details, compare our best results to the state-of-the-art, and conduct an ablation study.

8.4.1 Datasets and implementation details

Datasets. We conduct few-shot image classification experiments on three benchmarks: *mini*ImageNet [VBL$^+$16], *tiered*ImageNet [RTR$^+$18] and FC100 [ORL18]. *mini*ImageNet is the most widely used in related works [FAL17, HMX$^+$20, HCB$^+$19, SLCS19, SYZ$^+$18, HMX$^+$20]. *tiered*ImageNet and FC100 are either with a larger scale or a more challenging setting with lower image resolution, and have stricter training-test splits.

***mini*ImageNet** was proposed in [VBL$^+$16] based on ImageNet [RDS$^+$15]. There are 100 classes with 600 samples per class. Classes are divided into 64, 16, and 20 classes respectively for sampling tasks for meta-training, meta-validation and meta-test. ***tiered*ImageNet** was proposed in [RTR$^+$18]. It contains a larger subset of ImageNet [RDS$^+$15] with 608 classes

(779, 165 images) grouped into 34 super-class nodes. These nodes are partitioned into 20, 6, and 8 disjoint sets respectively for meta-training, meta-validation and meta-test. Its super-class based training-test split results in a more challenging and realistic regime with test tasks that are less similar to training tasks. **FC100** is based on the CIFAR100 [Kri09]. The few-shot task splits were proposed in [ORL18]. It contains 100 object classes and each class has 600 samples of 32×32 color images per class. On these datasets, we consider the (5-class, 1-way) and (5-class, 5-way) classification tasks. We use the same task sampling strategy as in related works [FAL17, AES19, HMX+20].

Backbone architectures. In MAML+E^{3}BM, we use a 4-layer convolution network (4CONV) [FAL17, AES19]. In MTL+E^{3}BM, we use a 25-layer residual network (ResNet-25) [QLSY18, YHZS20, SLC+22]. Followed by convolution layers, we apply an average pooling layer and a fully-connected layer. In SIB+E^{3}BM, we use a 28-layer wide residual network (WRN-28-10) as SIB [HMX+20].

The configuration of base-learners. In MTL [SLCS19] and SIB [HMX+20], the base-learner is a single fully-connected layer. In MAML [FAL17], the base-learner is the 4-layer convolution network. In MTL and MAML, the base-learner is randomly initialized and updated during meta-learning. In SIB, the base-learner is initialized with the averaged image features of each class. The number of base-learners M in MTL+E^{3}BM and SIB+E^{3}BM are respectively 100 and 3, i.e., the original numbers of training epochs in [SLCS19, HMX+20].

The configuration of hyperprior learners. In Fig. 8.3, we show two options for hyperprior learners (i.e., Ψ_α and Ψ_v). Fig. 8.3(a) is the epoch-independent option, where each epoch has two FC layers to produce α and v respectively. Fig. 8.3(b) is the epoch-dependent option which uses an LSTM to generate α and v at all epochs. In terms of the learning hyperprior learners, we have two settings: inductive learning denoted as "Ind.", and transductive learning as "Tra.". "Ind." is the supervised learning in classical few-shot learning methods [FAL17, SLCS19, LMRS19, VBL+16, SSZ17]. "Tra." is semi-supervised learning, based on the assumption that all test images of the episode are available. It has been applied to many recent works [LLP+19, HCB+19, HMX+20].

Ablation settings. We conduct a careful ablative study for two components, i.e., "ensembling multiple base-learners" and "meta-learning hyperprior learners". We show their effects indirectly by comparing our results to those of using arbitrary constant or learned values of v and α. **In terms of** v, we have 5 ablation options: (v1) "E^{3}BM" is our method generating v from Ψ_v; (v2) "learnable" is to set v to be update by meta gradient descent same as θ in [FAL17]; (v3) "optimal" means using the values learned by option (a2) and freezing them during the actual learning; (v4) "equal" is an simple baseline using equal weights; (v5) "last-epoch" uses only the last-epoch base-learner, i.e., v is set to $[0, 0, ..., 1]$. In the experiments of (v1)-(v5), we simply set α as in the following (a4) [FAL17, SLCS19, HMX+20]. **In terms of** α, we have 4 ablation options: (a1) "E^{3}BM" is our method generating α from Ψ_α; (a2) "learnable" is to set α to be update by meta gradient descent same as θ in [FAL17]; (a3) "optimal" means using the values learned by option (a2) and freezing them during the actual learning; (a4) "fixed" is a simple baseline that uses manually chosen α following [FAL17, SLCS19, HMX+20]. In the experiments of (a1)-(a4), we simply set v as in (v5), same with the baseline method [SLCS19].

Methods	Backbone	*mini*ImageNet		*tiered*ImageNet		FC100	
		1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
MatchNets [VBL+16]	4CONV	43.44	55.31	–	–	–	–
ProtoNets [SSZ17]	4CONV	49.42	68.20	53.31	72.69	–	–
MAML$^\diamond$ [FAL17]	4CONV	48.70	63.11	49.0	66.5	38.1	50.4
MAML++$^\diamond$ [AES19]	4CONV	52.15	68.32	51.5	70.6	38.7	52.9
TADAM [ORL18]	ResNet-12	58.5	76.7	–	–	40.1	56.1
MetaOptNet [LMRS19]	ResNet-12	62.64	78.63	65.99	81.56	41.1	55.5
CAN [HCB+19]	ResNet-12	63.85	79.44	***69.89***	84.23	–	–
CTM [LED+19]	ResNet-18	***64.12***	80.51	68.41	***84.28***	–	–
MTL [SLCS19]	ResNet-12	61.2	75.5	–	–	**45.1**	57.6
MTL$^\diamond$ [SLCS19]	ResNet-25	63.4	80.1	69.1	84.2	43.7	***60.1***
LEO [RRS+19]	WRN-28-10	61.76	77.59	66.33	81.44	–	–
Robust20-dist‡ [DSM19]	WRN-28-10	63.28	**81.17**	–	–	–	–
MAML+E^{3}BM	4CONV	53.2(↑4.5)	65.1(↑2.0)	52.1(↑3.1)	70.2(↑3.7)	39.9(↑1.8)	52.6(↑2.2)
(+time, +param)	–	(8.9, 2.2)	(9.7, 2.2)	(10.6, 2.2)	(9.3, 2.2)	(7.8, 2.2)	(12.1, 2.2)
MTL+E^{3}BM	ResNet-25	**64.3**(↑0.9)	***81.0***(↑0.9)	**70.0**(↑0.9)	**85.0**(↑0.8)	***45.0***(↑1.3)	**60.5**(↑0.4)
(+time, +param)	–	(5.9, 0.7)	(10.2, 0.7)	(6.7, 0.7)	(9.5, 0.7)	(5.7, 0.7)	(7.9, 0.7)

(a) Inductive Methods

Methods	Backbone	*mini*ImageNet 1-shot	*mini*ImageNet 5-shot	*tiered*ImageNet 1-shot	*tiered*ImageNet 5-shot	FC100 1-shot	FC100 5-shot
EGNN [KKKY19]	ResNet-12	64.02	77.20	65.45	82.52	–	–
CAN+T [HCB+19]	ResNet-12	67.19	***80.64***	73.21	**84.93**	–	–
SIB$^{\diamond\ddagger}$ [HMX+20]	WRN-28-10	***70.0***	79.2	***72.9***	82.8	***45.2***	***55.9***
SIB+E^{3}BM‡	WRN-28-10	**71.4**(↑1.4)	**81.2**(↑2.0)	**75.6**(↑2.7)	***84.3***(↑1.5)	**46.0**(↑0.8)	**57.1**(↑1.2)
(+time, +param)	–	(2.1, 0.04)	(5.7, 0.04)	(5.2, 0.04)	(4.9, 0.04)	(6.1, 0.04)	(7.3, 0.04)

(b) Transductive Methods

$^\diamond$Our implementation on *tiered*ImageNet and FC100. ‡Input image size: $80 \times 80 \times 3$.

Table 8.1: The 5-class few-shot classification accuracies (%) on *mini*ImageNet, *tiered*ImageNet, and FC100. "(+time, +param)" denote the additional computational time (%) and parameter size (%), respectively, when plugging-in E^{3}BM to baselines (MAML, MTL and SIB). "–" means no reported results in original papers. The **best** and ***second best*** results are highlighted.

8.4.2 Results and analyses

In Table 8.1, we compare our best results to the state-of-the-arts. In Table 8.2, we present the results of using different kinds of hyperprior learner, i.e., regarding two architectures (FC and LSTM) and two learning strategies (inductive and transductive). In Fig. 8.4(a)(b), we show the validation results of our ablative methods, and demonstrate the change during meta-training iterations. In Fig. 8.4(c)(d), we plot the generated values of v and α during meta-training.

Comparing to the state-of-the-arts. Table 8.1 shows that the proposed E^{3}BM achieves the best few-shot classification performance in both 1-shot and 5-shot settings, on three benchmarks. Please note that [DSM19] reports the results of using different backbones and input image sizes. We choose its results under the same setting as ours, i.e., using WRN-28-10 networks and $80 \times 80 \times 3$ images, for fair comparison. In our approach, plugging-in E^{3}BM to the state-of-the-art model SIB achieves 1.6% of improvement on average, based on the identical network architecture. This improvement is significantly larger as 2.9% when taking MAML as the baseline. All these show to be more impressive if considering the tiny overheads from

No.	Setting			*mini*ImageNet		*tiered*ImageNet		FC100	
	Method	Hyperprior	Learning	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
1	MTL [SLCS19]	–	Ind.	63.4	80.1	69.1	84.2	43.7	60.1
2	MTL+E^{3}BM	FC	Ind.	64.3	80.9	69.8	84.6	44.8	60.5
3	MTL+E^{3}BM	FC	Tra.	64.7	80.7	69.7	84.9	44.7	**60.6**
4	MTL+E^{3}BM	LSTM	Ind.	64.3	81.0	70.0	***85.0***	45.0	60.4
5	MTL+E^{3}BM	LSTM	Tra.	64.5	***81.1***	70.2	**85.3**	45.1	**60.6**
6	SIB [HMX$^+$20]	–	Tra.	70.0	79.2	72.9	82.8	45.2	55.9
7	SIB+E^{3}BM	FC	Tra.	***71.3***	81.0	***75.2***	83.8	***45.8***	56.3
8	SIB+E^{3}BM	LSTM	Tra.	**71.4**	**81.2**	**75.6**	84.3	**46.0**	57.1

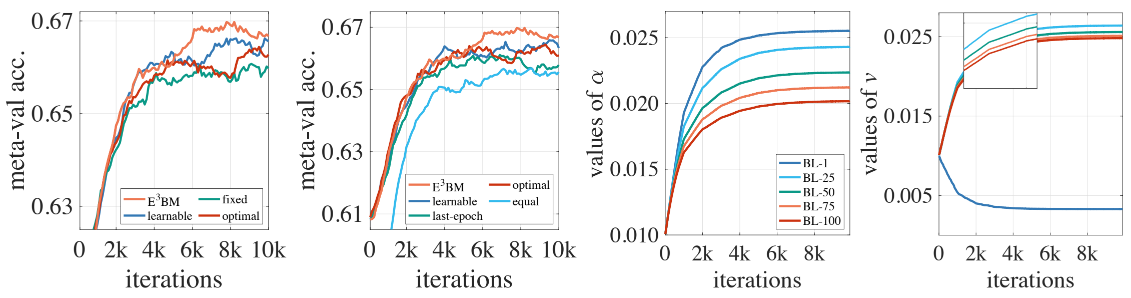
Table 8.2: The 5-class few-shot classification accuracies (%) of using different hyperprior learners, on the *mini*ImageNet, *tiered*ImageNet, and FC100. "Ind." and "Tra." denote the inductive and transductive settings, respectively. The **best** and ***second best*** results are highlighted.

pluging-in. For example, using E^{3}BM adds only 0.04% learning parameters to the original SIB model, and it gains only 5.2% average overhead regarding the computational time. It is worth mentioning that the amount of learnable parameters in SIB+E^{3}BM is around 80% less than that of model in [DSM19] which ensembles 5 deep networks in parallel (and later learns a distillation network).

Hyperprior learners. In Table 8.2, we can see that using transductive learning clearly outperforms inductive learning, e.g., No. 5 *vs.* No. 4. This is because the "transduction" leverages additional data, i.e., the episode-test images (no labels), during the base-training. In terms of the network architecture, we observe that LSTM-based learners are slightly better than FC-based (e.g., No. 3 *vs.* No. 2). LSTM is a sequential model and is indeed able to "observe" more patterns from the adaptation behaviors of models at adjacent epochs.

Ablation study. Fig. 8.4(a) shows the comparisons among α related ablation models. Our E^{3}BM (**orange**) again performs the best, over the models of using any arbitrary α (**red** or **light blue**), as well as over the model with α optimized by the meta gradient descent (**blue**) [FAL17]. Fig. 8.4(b) shows that our approach E^{3}BM works consistently better than the ablation models related to v. We should emphasize that E^{3}BM is clearly more efficient than the model trained with meta-learned v (**blue**) through meta gradient descent [FAL17]. This is because E^{3}BM hyperprior learners generate empirical weights conditional on task-specific data. The LSTM-based learners can leverage even more task-specific information, i.e., the hidden states from previous epochs, to improve efficiency.

The values of α and v learned by E^{3}BM. Fig. 8.4(c)(d) shows the values of α and v during the meta-training iterations in our approach. Fig. 8.4(c) show the base-learners working at later training epochs (e.g., BL-100) tend to get smaller values of α. This is actually similar to the common manual schedule, i.e. monotonically decreasing learning rates, of conventional large-scale network training [HZZ$^+$19]. The difference is that in our approach, this is "scheduled" in a total automated way by hyperprior learners. Another observation is that the highest learning rate is applied to BL-1. This actually encourages BL-1 to make an influence as significant as possible. It is very helpful to reduce meta gradient diminishing when unrolling and back-propagating gradients through many base-learning epochs (e.g., 100 epochs in MTL). Fig. 8.4(d) shows that BL-1 working at the initial epoch has the lowest values of v. In other words, BL-1 is almost disabled in the prediction of episode test. Intriguingly,



(a) Val. acc. for α (b) Val. acc. for v (c) Values of α (d) Values of v

Figure 8.4: (a)(b): The meta-validation accuracies of ablation models. The legends are explained in (a1)-(a4) and (v1)-(v5) in Sec. 8.4.1 **Ablation settings**. All curves are smoothed with a rate of 0.9 for better visualization. (c)(d): The values of α and v generated by Ψ_α and Ψ_v, respectively. The setting is using MTL+E^{3}BM, ResNet-25, on *mini*ImageNet, 1-shot.

BL-25 instead of BL-100 gains the highest v values. Our explanation is that during the base-learning, base-learners at latter epochs get more overfitted to the few training samples. Their functionality is thus suppressed. Note that our empirical results revealed that including the overfitted base-learners slightly improves the generalization capability of the approach.

8.5 CONCLUSIONS

We propose a novel E^{3}BM approach that tackles the few-shot problem with an ensemble of epoch-wise base-learners that are trained and combined with task-specific hyperparameters. In specific, E^{3}BM meta-learns the hyperprior learners to generate such hyperparameters conditional on the images as well as the training states for each episode. Its resulting model allows making use of multiple base-learners for more robust predictions. It does not change the basic training paradigm of episodic few-shot learning, and is thus *generic* and easy to plug-and-play with existing methods. By applying E^{3}BM to multiple baseline methods, e.g., MAML, MTL, and SIB, we achieved top performance on three challenging few-shot image classification benchmarks, with little computation or parametrization overhead.

III

Incremental Object Detection: Learning in Real-world Applications

In the third part, we apply incremental learning and few-shot learning algorithms in more challenging real-world applications.

Specifically, in Chapter 9, we propose a ContinuaL DEtection TRansformer (CL-DETR), a new method for transformer-based incremental object detection that enables effective usage of two popular incremental learning techniques for image classification, knowledge distillation and exemplar replay, on recent top-performance transformer-based detectors.

9 CONTINUAL DETECTION TRANSFORMER

Contents

Incremental object detection (IOD) aims to train an object detector in phases, each with annotations for new object categories. As in other incremental settings, IOD is subject to catastrophic forgetting, which is often addressed by techniques such as knowledge distillation (KD) and exemplar replay (ER). However, KD and ER do not work well if applied directly to state-of-the-art transformer-based object detectors such as Deformable DETR [ZSL+21] and UP-DETR [DCLC21]. In this chapter, we solve these issues by proposing a ContinuaL DEtection TRansformer (CL-DETR), a new method for transformer-based IOD which enables effective usage of KD and ER in this context. First, we introduce a Detector Knowledge Distillation (DKD) loss, focusing on the most informative and reliable predictions from old versions of the model, ignoring redundant background predictions, and ensuring compatibility with the available ground-truth labels. We also improve ER by proposing a calibration strategy to preserve the label distribution of the training set, therefore better matching training and testing statistics. We conduct extensive experiments on COCO 2017 and demonstrate that CL-DETR achieves state-of-the-art results in the IOD setting.

This chapter is based on [LSVR23]. As the first author of [LSVR23], Yaoyao Liu conducted all experiments and was the main writer.

9.1 INTRODUCTION

Humans inherently learn in an incremental manner, acquiring new concepts over time without forgetting previous ones. In contrast, machine learning suffers from *catastrophic forgetting* [MC89, MH93, KPR+17], where learning from non-i.i.d. data can override knowledge acquired previously. Unsurprisingly, forgetting also affects object detection models [FWY22, AKT19, PZL20, SSA17, VYP+22, YDS+22, JRK+21]. In this context, the problem

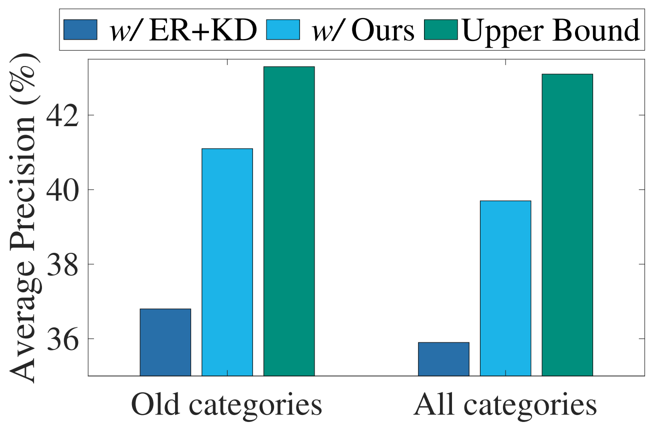


Figure 9.1: The final Average Precision (AP, %) of two-phase incremental object detection on COCO 2017. We observe 70 and 10 categories in the first and second phases, respectively. The baseline is Deformable DETR [ZSL+21]. "Upper bound" shows the results of joint training with all previous data accessible in each phase.

was formalized by Shmelkov et al. [SSA17], who defined an incremental object detection (IOD) protocol, where the training samples for different object categories are observed in different phases, restricting the ability of the trainer to access past data.

Popular methods to address forgetting in tasks other than detection include Knowledge Distillation (KD) and Exemplar Replay (ER). KD [LH18, DCO+20, HPL+19, HTM+21, ZXG+20] introduces regularization terms in the learning objective in an attempt to preserve previous knowledge when training the model on new data. The key idea is to encourage the new model's logits or feature maps to be close to those of the old model. ER methods [RKSL17, LSL+20, LSS21b, WZY+22, CMG+18, LWM+20] work instead by memorising some of the past training data (the *exemplars*), replaying them in the following phases to "remember" the old object categories.

Recent state-of-the-art results in object detection have been achieved by a family of transformer-based architectures that include DETR [CMS+20], Deformable DETR [ZSL+21] and UP-DETR [DCLC21]. In this chapter, we show that KD and ER do not work well if applied directly to these models. For instance, in Fig. 9.1 we show that applying KD and ER to Deformable DETR leads to much worse results compared to training with all data accessible in each phase (i.e., the standard non-incremental setting).

We identify two main issues that cause this drop in performance. First, transformer-based detectors work by testing a large number of object hypotheses in parallel. Because the number of hypotheses is much larger than the typical number of objects in an image, most of them are negative, resulting in an unbalanced KD loss. Furthermore, because both old and new object categories can co-exist in any given training image, the KD loss and regular training objective can provide contradictory evidence. Second, ER methods for image classification try to sample the same number of exemplars for each category. In IOD, this is not a good strategy because the true object category distribution is typically highly skewed. Balanced sampling causes a mismatch between the training and testing data statistics.

In this chapter, we solve these issues by proposing *ContinuaL DEtection TRansformer* (CL-DETR), a new method for transformer-based IOD which enables effective usage of KD and ER in this context. CL-DETR introduces the concept of *Detector Knowledge Distillation* (DKD), selecting the most confident object predictions from the old model, merging them with the ground-truth labels for the new categories while resolving conflicts, and applying standard joint bipartite matching between the merged labels and the current model's predictions for training. This approach subsumes the KD loss, applying it only for foreground predictions correctly matched to the appropriate model's hypotheses. CL-DETR also improves ER by introducing a new calibration strategy to preserve the distribution of object categories observed in the training data. This is obtained by carefully engineering the set of exemplars remembered to match the desired distribution. Furthermore, each phase consists of a main training step followed by a smaller one focusing on better calibrating the model.

We also propose a more realistic variant of the IOD benchmark protocol. In previous works [SSA17, FWY22], in each phase, the incremental detector is allowed to observe all images that contain a certain type of objects. Because images often contain a mix of object classes, both old and new, this means that the same images can be observed in different training phases. This is incompatible with the standard definition of incremental learning [RKSL17, LSL+20, HPL+19] where, with the exception of the examples deliberately stored in the exemplar memory, the images observed in different phases do not repeat. We redefine the IOD protocol to avoid this issue.

We demonstrate CL-DETR by applying it to different transformer-based detectors including Deformable DETR [ZSL+21] and UP-DETR [DCLC21]. As shown in Fig. 9.1, our results on COCO 2017 show that CL-DETR leads to significant improvements compared to the baseline, boosting AP by 4.2 percentage points compared to a direct application of KD and ER to the underlying detector model. We further study and justify our modeling choices via ablations.

To summarise, we make **four contributions**: (1) The DKD loss that improves KD for knowledge distillation by resolving conflicts between distilled knowledge and new evidence and by ignoring redundant background detections; (2) A calibration strategy for ER to match the stored exemplars to the training set distribution; (3) A revised IOD benchmark protocol that avoids observing the same images in different training phases; (4) Extensive experiments on COCO 2017, including state-of-the-art results, an in-depth ablation study, and further visualizations.

9.2 RELATED WORK

In this section, we discuss related works on transformer-based object detection, online learning, and hyperparameter optimization. We will not repeat the incremental learning and incremental object detection works that have been discussed in Chapter 2.

Transformer-based object detection. DEtection TRansformer (DETR) [CMS+20] proposes an elegant architecture for object detection based on a visual transformer [VSP+17]. Compared to pre-transformer approaches, DETR eliminates the need for non-maximum suppression in post-processing because self-attention can learn to remove duplicated detection by itself. This is achieved by using the Hungarian loss, matching each object hypothesis to exactly one target or background using bipartite matching [SCYK21]. Deformable DETR [ZSL+21] improves the performance of DETR, particularly for small objects, via sparse attention on multi-level feature maps. UP-DETR [DCLC21] leverages unsupervised learning to pre-train the parameters of the encoder and decoder in DETR to further boost the performance.

Our method does not fundamentally change these detectors and is in fact applicable to all similar ones. Instead, it proposes broadly-applicable changes that make transformer-based detector work well in combination with KD and ER for the IOD problem.

9.3 METHODOLOGY

After defining the incremental detection problem (Section 9.3.1) and providing the necessary background (Section 9.3.2), we introduce ContinuaL DEtection TRansformer (CL-DETR), a new method for incremental object detection that extends DETR-like detectors with knowledge distillation (KD; Section 9.3.3) and exemplar replay (ER; Section 9.3.4).

9.3.1 Incremental object detection

In *incremental object detection* (IOD) the goal is to train a detector in phases, where in each phase the model is only given annotations for a subset of the object categories. Formally, let $\mathcal{D} = \{(x, y)\}$ be a dataset of images x with corresponding object annotations y, such as COCO 2017 [LMB+14], and let $\mathcal{C} = \{1, \ldots, C\}$ be the set of object categories. We adapt such a dataset for benchmarking IOD as follows. First, we partition $\mathcal{D}$ and $\mathcal{C}$ into M subsets $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_M$ and $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_M$, one for each training phase. For each phase i, we modify the samples $(x, y) \in \mathcal{D}_i$ so that y only contains annotations for objects of class $\mathcal{C}_i$ and drop the others.[1]

In phase i of training, the model is only allowed to observe images $\mathcal{D}_i$ with annotations for objects of types $\mathcal{C}_i \subset \mathcal{C}$. Notably, images can and do contain objects of any possible type $\mathcal{C}$, but only types $\mathcal{C}_i$ are annotated in this phase. After phase i is complete, training switches to the next phase $i + 1$, so the model observes different images $\mathcal{D}_{i+1}$ and annotations for objects of different types $\mathcal{C}_{i+1}$.

For exemplar replay, we relax this training protocol and allow the model to memorise a small number of *exemplars* $\mathcal{E}_i \subset \mathcal{D}_i$ from the previous phases. In this case, the model is trained on the union $\mathcal{D}_i \cup \mathcal{E}_{1:i-1}$ where $\mathcal{E}_{1:i-1} = \mathcal{E}_1 \cup \cdots \cup \mathcal{E}_{i-1}$ forms the exemplar memory.

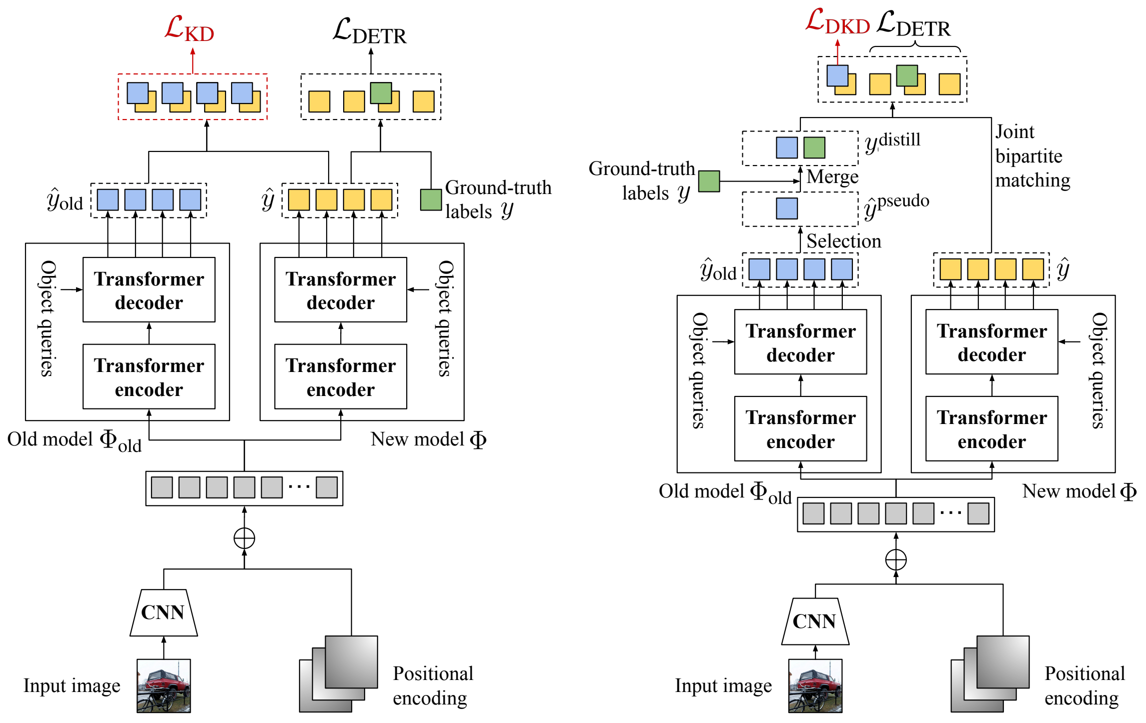
Note that this is a stricter and improved protocol compared to prior works in IOD [SSA17, FWY22]. In these works, the model is still presented a subset of annotations restricted to classes $\mathcal{C}_i$ in each phase; however, $\mathcal{D}_i \subset \mathcal{D}$ is defined as the subset of all images that contains objects of type $\mathcal{C}_i$. Because images contain a mix of object categories that can span different subsets $\mathcal{C}_i$, this means that different subsets $\mathcal{D}_i$ can overlap, so that the same images can be observed multiple times in different phases. This violates the standard definition of incremental learning [RKSL17, LSL+20, HPL+19] which assumes that different samples are observed in different phases. Our setting retains this property.

9.3.2 Transformer-based detectors

State-of-the-art methods like DETR [ZSL+21, CMS+20, DCLC21, LWZ+21, SCYK21, ZLL+22] build on powerful visual transformers to solve the object detection problem. In order to motivate and explain our method, we first review briefly how they work.

With reference to Fig. 9.2, the model Φ takes as input an image $x \in \mathbb{R}^{3 \times H \times W}$ and outputs the object predictions $\hat{y} = \Phi(x)$ using a number of attention and self-attention layers. The output $\hat{y} = (\hat{y}_j)_{j \in \mathcal{N}}$ is a sequence $\mathcal{N} = \{1, \ldots, N\}$ of object predictions $\hat{y}_j = (\hat{p}_j, \hat{b}_j)$,

[1] In this way, some images end up containing no annotated objects.



(a) Classical knowledge distillation **(b) Detector knowledge distillation (ours)**

Figure 9.2: **(a) Classical knowledge distillation.** There are two issues when directly applying KD [HVD15, LH18] to the transformer-based detectors [ZSL+21, CMS+20, DCLC21]. (i) Transformer-based detectors work by testing a large number of object hypotheses in parallel. Because the number of hypotheses is much larger than the typical number of objects in an image, most of them are negative, resulting in an unbalanced KD loss. (ii) Because both old and new object categories can co-exist in any given training image, the KD loss and regular training objective can provide contradictory evidence. **(b) Detector knowledge distillation (ours).** We select the most confident foreground predictions from the old model and use them as pseudo labels. We purposefully ignore background predictions because they are imbalanced and they can contradict the labels of the new classes available in the current phase. Then, we *merge* the pseudo labels for the old categories with the ground-truth labels for the new categories and use bipartite matching to train the model on the joint labels. This inherits the good properties of the original formulation such as ensuring one-to-one matching between labels and hypotheses and avoiding duplicate detections.

consisting of a class probability vector $\hat{p}_j : \mathcal{C} \cup \{\phi\} \rightarrow [0,1]$ and a vector $b_j \in [0,1]^4$ specifying the center and size of the object bounding box relative to the image size. Note that the support of $\hat{p}_j$ includes element ϕ that denotes the background class, or 'no object' (hence, $\hat{p}_j$ has $C+1$ dimensions).

The object predictions correspond to a fixed set of *object queries* internal to the model. Each query is thus mapped to an object instance or background. The order of the queries is conceptually immaterial, but queries are fixed and non-interchangeable after training. For instance, $\hat{y}_1$ is always the prediction that corresponds to the first query in the model. This is relevant for the application of KD.

For supervised training, the model is given ground truth object annotations $y = ((p_j, b_j))_{j \in \mathcal{N}}$ where p_j is the indicator vector of the category of the object and $b_j \in [0,1]^4$ is its bounding box. Images usually contain fewer objects than the number N of hypotheses, so y is padded with background detections for which $p_i(\phi) = 1$ and b_i is arbitrary. The model is trained end-to-end to optimise the loss,

$$\mathcal{L}_{\text{DETR}}(\hat{y}, y) = \sum_{i \in \mathcal{N}} \langle -\log \hat{p}_{\hat{\sigma}_i}, p_i \rangle + \mathbf{1}_{c(p_i) \neq \phi} \mathcal{L}_{\text{box}}(\hat{b}_{\hat{\sigma}_i}, b_i), \tag{9.1}$$

where $c(p_i) = \operatorname{argmax}_{c \in \mathcal{C} \cup \{\phi\}} p_i(c)$ is the class encoded by p_i, $\mathcal{L}_{\text{box}}(\hat{b}_{\hat{\sigma}_i}, b_i) = \gamma_1 \mathcal{L}_{\text{IoU}}(\hat{b}_{\hat{\sigma}_i}, b_i) + \gamma_2 \|\hat{b}_{\hat{\sigma}_i} - b_i\|_1$ is the bounding box prediction loss and $\hat{\sigma}$ is the best association of ground truth labels to object hypotheses, obtained by solving the matching problem,

$$\hat{\sigma} = \underset{\sigma \in \mathcal{S}_N}{\operatorname{argmax}} \sum_{i \in \mathcal{N}} \mathbf{1}_{c(p_i) \neq \phi} \left\{ -\langle \hat{p}_{\sigma_i}, p_i \rangle + \mathcal{L}_{\text{box}}(\hat{b}_{\sigma_i}, b_i), \right\} \tag{9.2}$$

using the Hungarian algorithm [Kuh55, SAN16]. Please see [CMS+20] for details.

9.3.3 Detector knowledge distillation

In a multi-phase learning scenario, at the beginning of a new phase, the model is initialized as $\Phi \leftarrow \Phi^{\text{old}}$ where Φ^{old} is the model trained in the phase before. As the new data for the current phase is received, training the model Φ as normal by minimising Eq. (9.1) leads to forgetting.

KD [HVD15, LH18] reduces forgetting by maintaining a copy of the old model and making sure that the outputs of the new and old models stay close. Applied to our transformer-based detectors, given a new training image-label pair (x, y), one computes the old model's output $\hat{y}^{\text{old}} = \Phi^{\text{old}}(x)$ and, minimizes the sum of the $\mathcal{L}_{\text{DETR}}(\hat{y}, y)$ loss with the *knowledge distillation loss*

$$\mathcal{L}_{\text{KD}}(\hat{y}, \hat{y}^{\text{old}}) = \sum_{j \in \mathcal{N}} \left[\sum_{c \in \mathcal{C}} -\hat{p}_j(c) \log \hat{p}_j^{\text{old}}(c) \right] + \mathcal{L}_{\text{box}}(\hat{b}_j, \hat{b}_j^{\text{old}}).$$

This loss compares the output tokens of the new and old models, which makes sense since they depend on the same object queries, at least initially, and are thus in correspondence. However, we find that this loss is dominated by background information because most of the tokens predict background. Furthermore, transformer-based detectors aim to find one-to-one matchings between predictions and ground-truth labels without duplicates, which is not accounted for by the classical KD loss.

The key issue is that summing losses $\mathcal{L}_{\text{DETR}} + \mathcal{L}_{\text{KD}}$ as in standard KD fails to properly account for the *structure* of the labels, which is crucial for detection problems, particularly in an incremental learning setting. Specifically, the old model knows about all categories seen so far during training *except* the new categories that are annotated in the current phase. However, the new training images contain multiple objects, including the old types, which are thus *not* annotated in the current phase. This means that $\mathcal{L}_{\text{DETR}}$ and $\mathcal{L}_{\text{KD}}$ provide potentially contradictory supervision.

We thus suggest that, in a detection context, new and old knowledge should be fused in a *structured manner*. As illustrated in Fig. 9.2, we do so by selecting the most confident foreground predictions from the old model and using them as pseudo labels. We purposefully ignore background predictions because they are imbalanced and they can contradict the labels of the new classes available in the current phase. Then, we *merge* the pseudo labels for the old categories with the ground-truth labels for the new categories and use bipartite matching to train the model on the joint labels. This inherits the good properties of the original formulation such as ensuring one-to-one matching between labels and hypotheses and avoiding duplicate detections.

Formally, given the predictions $\hat{y}^{\text{old}}$ from the old model, we first identify the subset $\mathcal{F} \subset \mathcal{N}$ of the ones that are predicted as foreground:

$$\mathcal{F} = \{j \in \mathcal{N} : \forall c \in \mathcal{C} : \hat{p}_j^{\text{old}}(c) > \hat{p}_j^{\text{old}}(\phi)\}.$$

Of these, we pick the subset $\mathcal{P} \subset \mathcal{F}$, $|\mathcal{P}| = K$ formed by the K most confident predictions, i.e.,

$$\forall i \in \mathcal{P},\ j \in \mathcal{F} - \mathcal{P} :\ \max_{c \in \mathcal{C}} \hat{p}_i^{\text{old}}(c) > \max_{c \in \mathcal{C}} \hat{p}_j^{\text{old}}(c).$$

Finally, we further restrict the predictions to the subset $\mathcal{Q} \subset \mathcal{P}$ that does not overlap too much with the ground-truth labels for the new categories:

$$\mathcal{Q} = \{j \in \mathcal{P} :\ \forall i \in \mathcal{N} :\ c(p_i) \neq \phi \Rightarrow \text{IoU}(\hat{b}_j^{\text{old}}, b_i) \leq \lambda\}.$$

In the experiments, we set $\lambda = 0.7$. With remain with a filtered set of pseudo-labels:

$$\hat{y}^{\text{pseudo}} = (\hat{y}_j^{\text{old}})_{j \in \mathcal{Q}}. \tag{9.3}$$

Next, we distill knowledge from the current labels y and the pseudo-labels obtained from the old model into a single, coherent set of labels

$$y^{\text{distill}} = (y_i)_{i:c(p_i) \neq \phi} \oplus \hat{y}^{\text{pseudo}} \oplus y^{\text{bg}}, \tag{9.4}$$

where we concatenate the object labels for the new categories, the pseudo-labels, and enough background labels y^{bg} to pad y^{distill} to contain N elements.

In this manner, the distillation occurs at the level of the labels. The model is still trained by using Eq. (9.1) as before, resulting in the *detector knowledge distillation* (DKD) loss:

$$\mathcal{L}_{\text{DKD}}(\hat{y}, y^{\text{distill}}) = \mathcal{L}_{\text{DETR}}(\hat{y}, y^{\text{distill}}). \tag{9.5}$$

Besides the usage of the distilled labels, the main difference between Eqs. (9.1) and (9.5) is that, while the class distribution p_i for the new label is deterministic, it is *not* for the pseudo-labels. Plugged in Eq. (9.1), this results in the standard distillation effect for categorial distributions trained using the cross entropy loss.

Algorithm 5: CL-DETR (the i-th phase)

```
   Input: new category data $\mathcal{D}_i$; old category exemplars $\mathcal{E}_{1:i-1}$; old model $\Phi^{\text{old}}$.
   Output: new model $\Phi$; exemplars $\mathcal{E}_{1:i}$.
1  Get $\mathcal{D}_i$ and load $\mathcal{E}_{1:i-1}$ from memory;
2  Let $\Phi \leftarrow \Phi^{\text{old}}$;
3  for epochs do
4      for mini-batches $(x, y) \in \mathcal{D}_i \cup \mathcal{E}_{1:i-1}$ do
5          Let $\hat{y}^{\text{old}} \leftarrow \Phi^{\text{old}}(x)$;
6          Get $\hat{y}^{\text{pseudo}}$ from $\hat{y}^{\text{old}}$ and $y$ using Eq. (9.3);
7          Get $y^{\text{distill}}$ from $\hat{y}^{\text{pseudo}}$ and $y$ using Eq. (9.4);
8          Let $\hat{y} \leftarrow \Phi(x)$;
9          Get $\hat{\sigma}$ by matching $y^{\text{distill}}$ to $\hat{y}$ using Eq. (9.2);
10         Compute $\mathcal{L}_{\text{DKD}}(\hat{y}, y^{\text{distill}})$ using Eq. (9.5);
11         Update $\Phi$ via a gradient step.
12     end
13 end
14 Build the exemplar set $\mathcal{E}_{1:i}$ using Algorithm 6;
15 for epochs do
16     for mini-batches $(x, y) \in \mathcal{E}_{1:i}$ do
17         Let $\hat{y} \leftarrow \Phi(x)$;
18         Compute $\mathcal{L}_{\text{DETR}}(\hat{y}, y)$ using Eq. (9.1);
19         Update $\Phi$ via a gradient step;
20     end
21 end
22 Save $\mathcal{E}_{1:i}$ to the memory.
```

Algorithm 6: Exemplar selection (the i-th phase)

```
   Input: new category data $\mathcal{D}_i$; old category exemplars $\mathcal{E}_{1:i-1}$; target number of
          exemplars $R_i$.
   Output: exemplars $\mathcal{E}_{1:i}$.
1  Let $\mathcal{E}_i \leftarrow \{\}$;
2  repeat
3      Select $e \in \mathcal{D}_i$ according to Eq. (9.6);
4      Let $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{x\}$;
5  until $R_i$ times;
6  Let $\mathcal{E}_{1:i} \leftarrow \mathcal{E}_i \cup \mathcal{E}_{1:i-1}$.
```

Setting	Method	Detection baseline	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
70+10	ERD [FWY22]	UP-DETR	$36.2_{\pm0.3}$	$54.8_{\pm0.4}$	$39.3_{\pm0.4}$	$20.8_{\pm0.3}$	$39.3_{\pm0.5}$	$47.9_{\pm0.3}$
	CL-DETR (ours)	UP-DETR	$37.6_{\pm0.2}$	$56.5_{\pm0.4}$	$39.4_{\pm0.3}$	$20.5_{\pm0.3}$	$39.1_{\pm0.4}$	$49.9_{\pm0.3}$
	LwF [LH18]	Deformable DETR	$24.5_{\pm0.3}$	$36.6_{\pm0.2}$	$26.7_{\pm0.4}$	$12.4_{\pm0.2}$	$28.2_{\pm0.4}$	$35.2_{\pm0.4}$
	iCaRL [RKSL17]	Deformable DETR	$35.9_{\pm0.4}$	$52.5_{\pm0.3}$	$39.2_{\pm0.3}$	$19.1_{\pm0.3}$	$39.4_{\pm0.5}$	$48.6_{\pm0.3}$
	ERD [FWY22]	Deformable DETR	$36.9_{\pm0.4}$	$55.7_{\pm0.4}$	$40.1_{\pm0.4}$	$21.4_{\pm0.3}$	$39.6_{\pm0.3}$	$48.7_{\pm0.3}$
	CL-DETR (ours)	Deformable DETR	$\mathbf{40.1}_{\pm0.3}$	$\mathbf{57.8}_{\pm0.4}$	$\mathbf{43.7}_{\pm0.3}$	$\mathbf{23.2}_{\pm0.3}$	$\mathbf{43.2}_{\pm0.2}$	$\mathbf{52.1}_{\pm0.3}$
40+40	ERD [FWY22]	UP-DETR	$35.4_{\pm0.4}$	$55.1_{\pm0.3}$	$38.3_{\pm0.3}$	$17.9_{\pm0.4}$	$39.0_{\pm0.3}$	$49.8_{\pm0.3}$
	CL-DETR (ours)	UP-DETR	$37.0_{\pm0.2}$	$\mathbf{56.2}_{\pm0.2}$	$39.1_{\pm0.4}$	$\mathbf{20.9}_{\pm0.2}$	$38.9_{\pm0.3}$	$49.2_{\pm0.3}$
	LwF [LH18]	Deformable DETR	$23.9_{\pm0.2}$	$41.5_{\pm0.3}$	$25.0_{\pm0.3}$	$12.0_{\pm0.4}$	$26.4_{\pm0.3}$	$33.0_{\pm0.5}$
	iCaRL [RKSL17]	Deformable DETR	$33.4_{\pm0.4}$	$52.0_{\pm0.3}$	$36.0_{\pm0.2}$	$18.0_{\pm0.3}$	$36.4_{\pm0.3}$	$45.5_{\pm0.4}$
	ERD [FWY22]	Deformable DETR	$36.0_{\pm0.2}$	$55.2_{\pm0.2}$	$38.7_{\pm0.3}$	$19.5_{\pm0.2}$	$38.7_{\pm0.3}$	$49.0_{\pm0.4}$
	CL-DETR (ours)	Deformable DETR	$\mathbf{37.5}_{\pm0.3}$	$55.1_{\pm0.4}$	$\mathbf{40.3}_{\pm0.2}$	$\mathbf{20.9}_{\pm0.2}$	$\mathbf{40.8}_{\pm0.4}$	$\mathbf{50.7}_{\pm0.2}$

Table 9.1: IOD results (%) on COCO 2017. In the $A + B$ setup, in the first phase, we observe a fraction $\frac{A}{A+B}$ of the training samples with A categories annotated. Then, in the second phase, we observe the remaining $\frac{B}{A+B}$ of the training samples, where B new categories are annotated. We test settings $A + B = 40 + 40$ and $70 + 10$. Exemplar replay is applied for all methods except for LwF [LH18]. We run experiments for three different categories and data orders and report the average AP with 95% confidence interval.

9.3.4 Distribution-preserving calibration

ER methods, which store a small number of exemplars and replay them in future phases, are shown to be effective in preserving the old category knowledge in IOD [JKKB21, LYR+20], but can suffer from the severe imbalance between old and new category annotations. Incremental learning methods for classification [HPL+19, LSS21a, WCW+19] usually use re-balancing strategies to address the imbalance problem. They create a category-balanced subset of the data and finetune some model components (e.g., the classifier) on it. However, such strategies do not apply directly to the IOD setting. First, the class distribution in detection is far from balanced, and a better strategy is to match the natural data distribution instead of the uniform one. Second, because there are multiple objects in each image, it is non-trivial to create a subset of exemplar images with a set number of objects for each category. We address these issues next.

Selecting exemplars to match the training distribution. Called during phase i, Algorithm 6 produces a new exemplar subset $\mathcal{E}_i$ whose distribution matches as well as possible the distribution of categories in the subset $\mathcal{D}_i$ of the data. This is achieved by adding to $\mathcal{E}_i$ a set number R_i of one exemplar $e^* \in \mathcal{D}_i$, one at a time, chosen by minimizing the Kullback-Leibler divergence [KL51] between the category marginals of $\mathcal{E}_i$ and $\mathcal{D}_i$:

$$e^* \leftarrow \sum_{c \in \mathcal{C}_i} p_{\mathcal{D}_i}(c) \log p_{\mathcal{E}_i \cup \{e\}}(c), \tag{9.6}$$

where $p_{\mathcal{D}}(c)$ denotes the probability of category c in dataset $\mathcal{D}$. Then, the overall exemplar set $\mathcal{E}_{1:i} = \mathcal{E}_i \cup \mathcal{E}_{1:i-1}$ is obtained as the union of the new subset just found and the previous exemplar et $\mathcal{E}_{1:i-1}$. Because classes in different subsets $\mathcal{D}_i$ are disjoint, this also means that, by the end of training, the distribution of classes in $\mathcal{E}_{1:M}$ approximates the one of the overall training set $\mathcal{D}$.

Learning using balanced data. In order to use the available data as well as possible while balancing the detector Φ, in each phase we update it in two steps. In the first step, the model is trained using the DKD loss on all the available data $\mathcal{D}_i \cup \mathcal{E}_{1:i-1}$ given by the union of the current data subset $\mathcal{D}_i$ and the exemplar memory $\mathcal{E}_{1:i-1}$ carried over the previous training phases. In the second step, the model is fine-tuned using the new exemplar set $\mathcal{E}_{1:i}$, ignoring $\mathcal{D}_i$ and using only the DETR loss, using less data but achieving better calibration. The overall algorithm is given in Algorithm 5.

9.4 EXPERIMENTS

We evaluate CL-DETR on COCO 2017 using two transformer-based detectors, Deformable DETR and UP-DETR [ZSL+21, DCLC21] as the baselines, and achieve consistent improvements compared to the baselines and a direct application of KD and ER. Below we describe the dataset and implementation details (Section 9.4.1) followed by results and analyses (Section 9.4.2).

9.4.1 Dataset and implementation details

Dataset and evaluation metrics. We conduct IOD experiments on COCO 2017 [LMB+14], which is widely used in related works [FWY22, PZM+21, ZSL+21, DCLC21]. Following [FWY22], the standard COCO metrics are used for evaluation, i.e., AP, AP_{50}, AP_{75}, AP_S, AP_M, and AP_L, In the ablation study, we introduce a new metric, *forgetting percentage points* (FPP), measuring the difference between the AP of the first phase model and the last phase model on the categories observed in the first phase.

Experiment setup. We conduct IOD experiments in the following setting. ***Two-phase setting:*** In the $A + B$ setup, in the first phase, we observe a fraction $\frac{A}{A+B}$ of the training samples with A categories annotated. Then, in the second phase, we observe the remaining $\frac{B}{A+B}$ of the training samples, where B new categories are annotated. We test settings $A + B = 40 + 40$ and $70 + 10$. ***Multiple-phase setting:*** In the $40 + X \times Y$ setup, in the first phase, we observe half of the training samples with 40 categories annotated. In each following phase, we observe $\frac{1}{2Y}$ of the training samples we have never seen before with annotations for X new categories. We run experiments for $40 + 20 \times 2$ and $40 + 10 \times 4$. We repeat each experiment three times, randomizing the order of categories and data in the different phases, and report the average APs. The total memory budget for the exemplars is set as 10% of the total dataset size.

Implementation details. For all experiments, we utilize an ImageNet pre-trained ResNet-50 backbone, following [ZSL+21, DCLC21]. For the experiments on Deformable DETR [ZSL+21], we use the standard configurations without their iterative bounding box refinement mechanism and the two-stage Deformable DETR. We train the model for 50 (Deformable DETR) and 150 epochs (UP-DETR), following the original implementations [ZSL+21, DCLC21]. In order to apply our distribution-preserving calibration (Section 9.3.4), we train the coarse Deformable DETR (UP-DETR) model for 40 (120) epochs and perform calibration for 10 (30) epochs to preserve the total number of epochs.

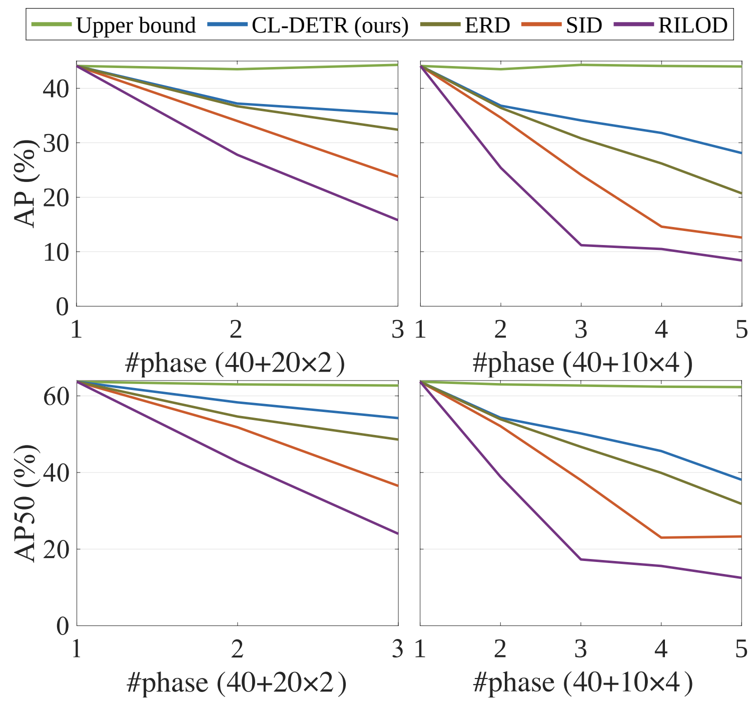


Figure 9.3: IOD results (AP/AP$_{50}$, %) on COCO 2017 in the $40 + 20 \times 2$ and $40 + 10 \times 4$ settings. Our method is based on Deformable DETR. Comparing methods: Upper Bound (the results of joint training with all previous data accessible in each phase), ERD [FWY22], SID [PZM^{+}21], and RILOD [LTG^{+}19]. The results of the related works are from [FWY22]. We use the same data split as [FWY22] for a fair comparison.

Row	Knowledge distillation (KD)	Joint bipartite matching	Pseudo label selection	Exemplar replay (ER)	Distribution preserving calibration	All categories ↑			
						AP	AP_S	AP_M	AP_L
1						4.2	1.6	4.7	5.8
2	✓					24.5	12.4	28.2	35.2
3	✓	✓				30.3	19.5	33.0	39.0
4	✓	✓	✓			33.9	16.3	37.1	49.2
5	✓	✓	✓	✓		37.9	20.8	40.9	50.4
6	✓	✓	✓		✓	**40.1**	**23.2**	**43.2**	**52.1**

Row	Knowledge distillation (KD)	Joint bipartite matching	Pseudo label selection	Exemplar replay (ER)	Distribution preserving calibration	Old categories ↑			
						AP	AP_S	AP_M	AP_L
1						0.7	0.2	0.8	0.8
2	✓					24.0	12.3	27.7	34.4
3	✓	✓				33.4	21.8	36.4	43.2
4	✓	✓	✓			33.9	16.6	36.8	50.0
5	✓	✓	✓	✓		39.0	21.6	41.7	52.3
6	✓	✓	✓		✓	**41.8**	**24.5**	**44.7**	**54.6**

Row	Knowledge distillation (KD)	Joint bipartite matching	Pseudo label selection	Exemplar replay (ER)	Distribution preserving calibration	FPP ↓			
						AP	AP_S	AP_M	AP_L
1						42.6	25.6	45.1	56.7
2	✓					19.3	13.5	18.2	23.1
3	✓	✓				9.9	4.0	9.5	14.3
4	✓	✓	✓			9.4	9.2	9.1	7.5
5	✓	✓	✓	✓		4.3	4.2	4.2	5.2
6	✓	✓	✓		✓	**1.5**	**1.3**	**1.2**	**2.9**

Table 9.2: Ablation results (%) for KD and ER, using Deformable DETR [ZSL+21] on COCO 2017 in the $70 + 10$ setting. "All categories" (higher is better) denote the results of the last phase model on 80 categories. "Old categories" (higher is better) denote the results of the last phase model on 70 categories observed in the first phase. "Forgetting percentage points (FPP)" (lower is better) show the difference between the AP of the first-phase model and the last-phase model on 70 categories observed in the first phase. The baseline (row 1) is finetuning the model without IOD techniques. Our method (CL-DETR) is shown in row 6.

Row	Setting	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1	K=5	39.7	57.4	43.1	22.7	42.6	**52.7**
2	K=10	**40.1**	**57.8**	**43.7**	23.2	**43.2**	52.1
3	K=20	39.9	57.8	43.2	**23.5**	42.9	51.7
4	$p \geq 0.1$	39.3	57.1	42.9	22.6	42.3	52.5
5	$p \geq 0.3$	39.6	57.5	43.0	23.2	42.4	52.2
6	$p \geq 0.5$	39.2	56.8	42.4	22.3	41.9	51.8

Table 9.3: Ablation result (%) for different pseudo label selection strategies on COCO 2017 using the $70 + 10$ setting. Rows 1–3 show the results for using different K when selecting top-K most-confident non-background predictions. Rows 4–6 show the results for using different thresholds p of the prediction scores to select the non-background predictions.

9.4.2 Results and analyses

Two-phase setting. Table 9.1 shows that, in the two-phase settings $70 + 10$ and $40 + 40$, applying CL-DETR to Deformable DETR [ZSL+21] and UP-DETR [DCLC21] consistently performs better than the state-of-the-art [FWY22] and other IOD methods [LH18, RKSL17]. In particular, Deformable DETR [ZSL+21] *w/* ours achieves the highest AP, e.g., 40.1% and 37.5% in the $70 + 10$ and $40 + 40$ settings, respectively. The performance gap is larger when we observe more categories in the 1-st phase. e.g., the AP differences between our method and [FWY22] are 3.2 and 1.5 percentage points when we observe 70 and 40 categories in the first phase, respectively. Likely due to CL-DETR benefiting more from a well-pre-trained model.

Multiple-phase setting. Figure 3.4 evaluates CL-DETR in the multiple-phase setting with large gains compared to other IOD methods in both the $40 + 20 \times 2$ and $40 + 10 \times 4$ experimental variants. The relative advantage of CL-DETR increases with the number of phases. For instance, our method improves the AP of [FWY22] by 2.9 percentage points in the $40 + 20 \times 2$ setting and by 7.4 percentage points in the $40 + 10 \times 4$ setting. This suggests that the advantage of CL-DETR shows more in challenging settings, where the forgetting problem is stronger due to the larger number of training phases.

Ablation study for DKD. In Table 9.2 (Rows 1–4) we ablate our DKD approach. By comparing row 2 to row 1, we observe that classical KD significantly improves the IOD performance compared to the baseline (i.e., finetuning the model without IOD techniques), but still results in large overall forgetting: 19.3 FPP. Comparing row 3 to row 2, we can see that joint bipartite matching works well and boosts the AP of all categories by 5.8 percentage points compared to conventional KD. The reason is that joint bipartite matching helps ensure a one-to-one matching between objects and hypotheses and discourages duplicate detections. Comparing row 4 to row 3, our pseudo label selection further improves the AP and reduces forgetting, helping the model to ignore the redundant background information and reducing conflicts between old and new labels.

Ablation study for ER. In Table 9.2 (Rows 5–6), we ablate our ER method. Comparing row 6 to row 5, we can see that the calibration strategy of Section 9.3.4 boosts both the

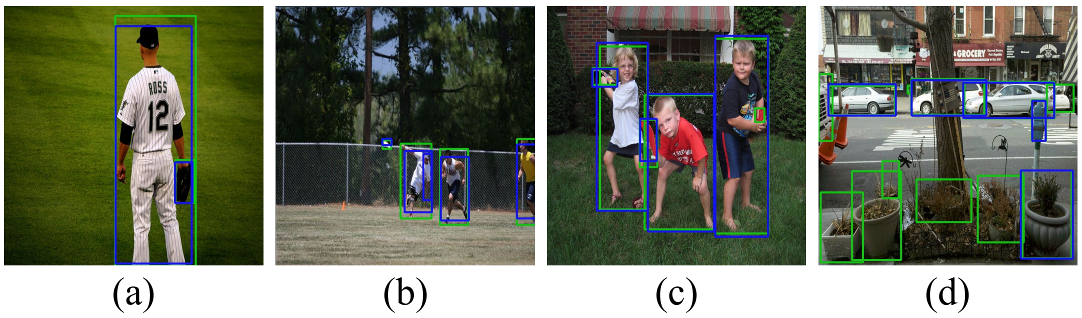


Figure 9.4: Visualizations of the old category pseudo (blue) and ground-truth (green) bounding boxes on COCO 2017 using the $70 + 10$ setting. **(a, b)**: Our method generates accurate pseudo bounding boxes that exactly match the ground-truth ones. **(c, d)**: When there are too many annotations in the images, generated pseudo bounding boxes cannot cover all ground-truth ones. However, the pseudo bounding boxes are still focused on the foreground objects.

all-category and old-category performance, by 1.8 and 2.1 percentage points respectively, compared to using conventional ER [RKSL17, LYR+20]. It also helps to overcome the catastrophic forgetting problem in IOD, reducing the AP forgetting by 2.1 percentage points. This is because the conventional ER balances the sample distributions, changing the category distribution of the training set, leading to inferior performance. Our method builds an exemplar set following the original category distribution of the training set, thus improves performance.

Ablation study for pseudo label selection strategies. In Table 9.3, we show the results for two pseudo label selection strategies: (1) selecting top-K most-confident non-background predictions (Rows 1–3); and (2) selecting the predictions using a threshold for the prediction scores (Rows 4–6). We observe the first strategy works better, with peak AP when K=10. The maximum performance difference is only 0.4 percentage points when using different values for K. This indicates our method is robust to its hyperparameter settings.

Visualizations. Figure 9.4 visualizes the old category pseudo (blue) and ground-truth (green) bounding boxes in some training samples in COCO 2017. In Fig. 9.4 (a,b), CL-DETR generates accurate pseudo bounding boxes that exactly match the ground-truth ones. This shows the effectiveness of our pseudo label selection strategy. In Fig. 9.4 (c,d), CL-DETR fails to generate pseudo bounding boxes for all objects in the images when there are too many. This is explained by our strategy of selecting the top-K most-confident non-background bounding boxes as the pseudo-labels followed by removing the ones that overlap with the new category ground-truth labels excessively. In this manner, the number of pseudo bounding boxes is always smaller than K. The trade-off, justified by our improvements in the experiments, is to prefer correct although possibly incomplete annotations than contradictory or noisy ones.

9.5 CONCLUSIONS

This chapter introduced CL-DETR, a novel IOD method that can effectively use KD and ER in transformer-based detectors. CL-DETR improves the standard KD loss by introducing DKD which selects the most informative predictions from the old model, rejecting redundant background predictions, and ensuring that the distilled information is consistent with the new ground-truth evidence. CL-DETR also improves ER by selecting exemplars to match the distribution of the training set. CL-DETR is fairly generic and can be easily applied to different transformer-based detectors, including Deformable DETR [ZSL+21] and UP-DETR [DCLC21], achieving large improvements. We have also defined a more realistic IOD benchmark protocol that avoids using duplicated images in different training phases. In the future, we plan to extend our method to more challenging settings such as online learning.

CONCLUSION AND FUTURE WORK 10

Contents

Significant progress has been made across various AI systems in recent years. Despite the success, AI algorithms still can not work well when training with imperfect data, e.g., continual data stream and limited labeled data. In contrast, human beings naturally possess the ability to learn from the above challenging imperfect data scenarios.

This thesis aims to explore the following imperfect data scenarios:

- Incremental learning, i.e., the training data distribution changes while learning. In many real scenarios, data are streaming, might disappear after a given period of time, or even can't be stored at all due to storage constraints or privacy issues. As a consequence, the old knowledge is over-written, and this phenomenon is called **catastrophic forgetting** [Alj19].
- Few-shot learning, i.e., the annotations of the training data are sparse. Collecting and manually labeling a large-scale amount of data can be very expensive and time-consuming. There are also many scenarios where scientists do not have access to the specific large-scale data of interest due to privacy and security reasons. As a consequence, the deep models **overfit the training data** and are very likely to make wrong decisions when they encounter rare circumstances [Yav20].

We tackle the above challenges based on the key idea, **learning to learn/optimize**, i.e., using **advanced learning and optimization techniques** to design data-driven methods to **dynamically adapt** the key elements in AI algorithms, e.g., selection of data, memory allocation, network architecture, essential hyperparameters, and control of knowledge transfer. We believe that the adaptive and dynamic design of system elements will significantly improve the capability of deep learning systems under limited data or continual streams, compared to the systems with fixed and non-optimized elements.

Our solutions are briefly summarized below.

- **Incremental learning.** In Part I, we study how to overcome the catastrophic forgetting problem by learning to optimize exemplar data, combine neural networks, and allocate memory. More specifically,
  - In Chapter 3, we propose a novel training framework by leveraging bilevel optimization to optimize a set of synthesized exemplar data to recall the old knowledge.

- In Chapter 4, we introduce a dynamic memory management strategy that learns to allocate memory budgets for different object classes in different incremental learning phases. The key technique is to utilize reinforcement learning to learn a policy to allocate memory for each class.
- In Chapter 5, we develop a generic network architecture that learns to combine high-stability and high-plasticity neural network blocks.
- In Chapter 6, we further balance the stability-plasticity trade-off for different data-receiving settings of incremental learning by introducing an online learning method that can adaptively optimize the tradeoff without knowing the setting as a priori.

• **Few-shot learning.** In Part II, we study how to improve the generalization ability of the model and tackle the overfitting problem by learning to transfer knowledge and ensemble deep models. Specifically,

  - In Chapter 7, we design a meta-transfer learning framework that allows us to leverage the transferrable pattern learned from existing large-scale tasks using meta-learning.
  - In Chapter 8, we introduce a method that learns to ensemble deep models to reduce the model uncertainty in few-shot learning.

The above AI algorithms are designed and evaluated based on the image classification task. In order to migrate the gap between these AI algorithms and more realistic applications, we further study the following topic in the thesis:

• **Incremental object detection.** In Part III, we apply incremental learning and few-shot learning algorithms in more challenging real-world applications. Specifically,

  - In Chapter 9, we propose a ContinuaL DEtection TRansformer (CL-DETR), a new method for transformer-based incremental object detection that enables effective usage of knowledge distillation and exemplar replay in a more realistic computer vision task, object detection.

In the following, we discuss our detailed contributions and future perspectives.

10.1 DISCUSSIONS OF CONTRIBUTIONS

10.1.1 Incremental learning

Natural learning systems, such as humans, inherently work incrementally as the number of concepts increases over time. They naturally learn new concepts while not forgetting previous ones. In contrast, when continuously updated using novel incoming data, current machine learning systems suffer from catastrophic forgetting, as the updates can override knowledge acquired from previous data. Catastrophic forgetting, thus, becomes a major problem for incremental learning systems. Our solution falls into four aspects – data, memory allocation, network architecture, and key hyperparameters.
Learning to optimize exemplar data. An intuitive approach to incremental learning is to keep a small number of old samples (i.e., exemplars) for replay within memory constraints. However, we found that the class boundaries learned from a few random or center exemplars

are weak in later training phases. We tackle this problem in Chapter 3, where we parameterize and learn the exemplars by solving a bi-level program.

Learning to allocate memory. In incremental learning, most of the old class samples are abandoned to free space for the next phase. The preserved data are exemplars used for replaying. However, existing methods use a static and ad hoc strategy for memory allocation, which is often sub-optimal. In Chapter 4, we introduce a memory management strategy based on reinforcement learning to achieve dynamic memory allocation for each class.

Learning to aggregate neural networks adaptively. In incremental learning, high-plasticity models easily forget old classes, but high-stability models are weak in learning new classes. We alleviate this issue by proposing a novel network architecture called Adaptive Aggregation Networks (AANets) Chapter 5, where we aggregate the feature maps from different network branches by meta-learned weights.

Learning to optimize the hyperparameter online. None of the existing incremental learning models can achieve the optimal trade-off in different data-receiving settings—where typically the training-from-half (TFH) setting needs more stability, but the training-from-scratch (TFS) needs more plasticity. To this end, in Chapter 6, we design an online learning method that can adaptively optimize the tradeoff without knowing the setting as prior. Specifically, we first introduce the key hyperparameters that influence the tradeoff, e.g., knowledge distillation (KD) loss weights, learning rates, and classifier types. Then, we formulate the hyperparameter optimization process as an online Markov Decision Process (MDP) problem and propose a specific algorithm to solve it.

10.1.2 Few-shot learning

We expect the machine learning model can learn new concepts from a handful of training examples, e.g., from 1 or 5 training images per class. Humans tend to be highly effective in such a context, often grasping the essential connection between new concepts and their own knowledge and experience, but this remains challenging for machine learning models. Besides, in many real-world applications, we lack large-scale training data, e.g., in medical image domains. It is thus desirable to improve machine models to handle few-shot settings.

Learning to transfer knowledge. One solution to few-shot learning is to leverage the transferrable pattern learned from existing large-scale tasks. However, fine-tuning pre-trained models on few-shot tasks often suffer from overfitting problem. In Chapter 7, we tackle this issue by updating neuron-level scaling and shifting weight using meta-learning. Our method helps deep neural networks converge faster while reducing the probability of overfitting when training on a few labeled data only.

Learning to ensemble deep models. In few-shot learning, the model uncertainty is high and often results in low performance. To tackle this issue, in Chapter 8, we proposed a solution of training an ensemble of models and using the combined prediction, which should be more robust. Furthermore, we use the sequence of base-learners while training a single base-learner as the ensemble and learn how to weigh them for best performance automatically.

10.1.3 Incremental object learning

Incremental object detection aims to train an object detector with the training samples for different object categories observed in different phases [SSA17]. Similar to incremental learning for image classification, the ability of the trainer to access past data is also restricted in incremental object detection. Incremental object detection is more challenging compared to incremental image classification because the catastrophic forgetting problem occurs in both localization and classification.

Continual detection transformer. Chapter 9 introduced CL-DETR, a novel IOD method that can effectively use KD and ER in transformer-based detectors. CL-DETR improves the standard KD loss by introducing DKD which selects the most informative predictions from the old model, rejecting redundant background predictions, and ensuring that the distilled information is consistent with the new ground-truth evidence. CL-DETR also improves ER by selecting exemplars to match the distribution of the training set. CL-DETR is fairly generic and can be easily applied to different transformer-based detectors, including Deformable DETR [ZSL+21] and UP-DETR [DCLC21], achieving large improvements.

10.2 FUTURE DIRECTIONS

My past research has demonstrated great promise in applying *learning to learn* techniques to standard classification models when facing limited or incrementally coming data. Looking forward, We envision a future where *learning to learn* methods are scaled to more realistic data settings and applied to more challenging vision tasks. To achieve this vision, below we outline a few future directions we plan to pursue.

10.2.1 Incremental learning

Vision language pre-training for incremental learning. A good initial model is very important to incremental learning performance [ZZC+22]. Thus, how to obtain such an initial model with accessible data is an open question for incremental learning. CLIP [RKH+21] has shown to be a very powerful pre-training strategy with only unlabeled vision and language training data, which can be easily obtained in many real-world applications. Thus, we plan to design an algorithm that can effectively pre-train the initial model with a CLIP-based framework and transfer it to boost performance on the continual data stream. To achieve this goal, we plan to leverage the ensemble of the Bayes models. My work [LSS20] can serve as a good starting point.

Incremental 3D object detection. If we take a closer look at the typical learning setup, most of the advances in 3D object detection have been realized using static images in an offline learning setup. On the contrary, humans receive a continuous temporal stream of visual data and train and test the model on the same visual data, which is an online continual setting. Hence, we plan to design a practical object detection framework that can handle a continual data stream without forgetting the learned knowledge. To achieve this goal, we plan to leverage many unlabeled objects in the training samples and develop re-balancing strategies for detection and segmentation. My work [LSS21a, LHL+21] can serve as a good starting point.

Cross-domain incremental learning. Despite the significant improvements that have been made in the incremental learning field, few existing works explore how to train an incremental model with the training samples coming from different domains continuously. However, we usually need to face the domain transfer problem in many real-world applications. Hence, we plan to design an incremental learning algorithm that adapts to domain shifts. To achieve this goal, we plan to leverage the *learning to learn* technique to adjust the key components, e.g., hyperparameters, to adapt to different domains. My work [LLSS23a] can serve as a good starting point.

10.2.2 Few-shot learning

Generic few-shot learning framework. Majority of existing few-shot learning methods tackle specific visual tasks by using task-specific prior information. However, little attention has been devoted to fundamental problems of exploring what makes it hard to transfer few-shot learning techniques into real-world applications. Hence, we would like to exploit generic few-shot learning frameworks that are applicable to different visual learning tasks, such as 3D-scene geometry and medical imaging. Possible tools include meta-learning, transfer learning, and neural architecture search. My work [LSS21a, LSS20, SLC^{+}22] can serve as a good starting point. Furthermore, We are also enthusiastic about collaborating with domain experts and designing algorithms based on specific domain knowledge.

Generalized few-shot learning. Existing few-shot learning algorithms mainly aim to achieve high performance on the novel classes only. However, in many real-world applications, we are interested in generalized few-shot learning, where the model has to predict both base and novel classes. As we need to adapt the few-shot models to the novel classes without losing the knowledge of the old classes, we plan to use the regularization techniques in incremental learning to achieve our goal. My work [LSS21a, LLSS23a] can serve as a good starting point.

Few-shot 3D object detection. In 3D-related tasks such as 3D object detection, we usually need to face few-shot training data because it is difficult to collect high-quality 3D training samples. Hence, we plan to design a practical object detection framework that can be trained with limited labeled samples. To achieve this goal, we plan to adapt the episodic training framework of meta-learning to 3D object detection. Thus, we can transfer knowledge from another data source to boost the performance of our target 3D task. My work [SLC^{+}22] can serve as a good starting point.

10.2.3 A broader view on the topic

Our long-term goal is to develop AI algorithms that work with imperfect training data. Incremental learning and few-shot learning are simply two directions toward this goal. In the future, we plan to explore other related topics, including semi-supervised and self-supervised learning.

Semi-supervised and self-supervised learning. The main challenges in incremental learning and few-shot learning come from lacking enough labeled data. Semi-supervised and self-supervised learning are both two practical solutions for learning with limited labeled data. While semi-supervised learning leverages unlabeled data in addition to labeled data, self-supervised learning learns from a completely unlabeled dataset by solving other proxy tasks that make use of the structure of the input data. Hence, I plan to improve class-incremental learning and few-shot learning frameworks by incorporating semi-supervised and self-supervised learning techniques. To achieve this goal, I plan to utilize and develop advanced tools from online learning and reinforcement learning to control the usage of unlabeled data in incremental learning and few-shot learning frameworks. My work [LSS21b, LHL$^+$21] can serve as a good starting point.

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[AAB$^+$16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 1603.04467, 2016. Cited on pages 31 and 114.

[ABB$^+$18] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*, 2018. Cited on pages 43 and 53.

[ABE$^+$18] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, pages 144–161, 2018. Cited on page 15.

[ACBFS02] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002. Cited on pages 5, 69, 71, 72, 75, and 76.

[ACT17] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *CVPR*, pages 3366–3375, 2017. Cited on page 3.

[AES19] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. In *ICLR*, 2019. Cited on pages 6, 19, 20, 86, 98, 99, 100, 108, 116, and 117.

[AG12] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *COLT*, volume 23, pages 39.1–39.26, 2012. Cited on pages 72 and 75.

[AKT19] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *CVPR*, pages 11254–11263, 2019. Cited on pages 3 and 123.

[Alj19] Rahaf Aljundi. *Continual learning in neural networks*. PhD thesis, KU Leuven, Belgium, 2019. Cited on pages 1, 3, and 139.

[And08] Terry Anderson. *The theory and practice of online learning*. Athabasca University Press, 2008. Cited on page 71.

[AO10] Peter Auer and Ronald Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010. Cited on page 72.

[ATB$^+$20] Davide Abati, Jakub Tomczak, Tijmen Blankevoort, Simone Calderara, Rita Cucchiara, and Babak Ehteshami Bejnordi. Conditional channel gated networks for task-aware continual learning. In *CVPR*, pages 3931–3940, 2020. Cited on pages 16 and 17.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. Cited on page 109.

[BBCG92] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992. Cited on pages 6 and 86.

[Ben00] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000. Cited on page 110.

[BHTV19] Luca Bertinetto, João F. Henriques, Philip H. S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR*, 2019. Cited on pages 18 and 99.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. Cited on page 46.

[BKM17] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. Cited on page 112.

[BLCW09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009. Cited on pages 87 and 88.

[BP19] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *CVPR*, pages 583–592, 2019. Cited on pages 16 and 70.

[BPRS18] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *JMLR*, 18:1–43, 2018. Cited on page 72.

[BRCŚ$^+$17] Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartlomiej Światkowski, Bernhard Schölkopf, and Richard E Turner. Discriminative k-shot learning using probabilistic models. *arXiv*, 1706.00326, 2017. Cited on pages 19 and 99.

[Bre96] Leo Breiman. Stacked regressions. *Machine Learning*, 24(1):49–64, 1996. Cited on pages 108 and 110.

[BV18] Sergey Bartunov and Dmitry P. Vetrov. Few-shot generative modelling with generative matching networks. In *AISTATS*, pages 670–678, 2018. Cited on page 86.

[Car95] Rich Caruana. Learning many related tasks at the same time with backpropagation. In *NIPS*, pages 657–664, 1995. Cited on page 108.

[CDAT18] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, pages 532–547, 2018. Cited on page 15.

[CF16] Olivier Canévet and François Fleuret. Large scale hard sample mining with monte carlo tree search. In *CVPR*, pages 5128–5137, 2016. Cited on page 89.

[CGS16] Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016. Cited on page 44.

[Cha20] Arslan Chaudhry. *Continual Learning for Efficient Machine Learning*. PhD thesis, University of Oxford, UK, 2020. Cited on page 3.

[CL18] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018. Cited on page 3.

[CLK$^+$19] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *ICLR*, 2019. Cited on pages 18, 101, 102, and 153.

[CMG+18] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, pages 241–257, 2018. Cited on pages 3, 15, 17, 20, 26, 30, 34, 37, 56, 57, 79, 124, 145, and 146.

[CMS+20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, 2020. Cited on pages 124, 125, 126, 127, 128, and 149.

[CPK+18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018. Cited on page 88.

[CRRE19] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019. Cited on pages 3 and 14.

[CUH16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016. Cited on pages 5 and 86.

[CYC19] Li Chen, Chunyan Yu, and Lvcai Chen. A new knowledge distillation for incremental object detection. In *IJCNN*, pages 1–7, 2019. Cited on page 21.

[DCLC21] Zhigang Dai, Bolun Cai, Yugeng Lin, and Junying Chen. UP-DETR: Unsupervised pre-training for object detection with transformers. In *CVPR*, pages 1601–1610, 2021. Cited on pages 8, 21, 123, 124, 125, 126, 127, 132, 135, 137, 142, and 149.

[DCO+20] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, 2020. Cited on pages 3, 4, 5, 14, 15, 17, 20, 42, 44, 47, 50, 51, 56, 57, 58, 60, 62, 64, 65, 66, 73, 74, 78, 79, 124, 146, and 152.

[Dom12] Justin Domke. Generic methods for optimization-based modeling. In *AISTATS*, pages 318–326, 2012. Cited on page 110.

[DSM19] Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Diversity with cooperation: Ensemble methods for few-shot classification. In *ICCV*, pages 10275–10284, 2019. Cited on pages 18, 110, 113, 117, and 118.

[DSP+19] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *CVPR*, pages 5138–5146, 2019. Cited on page 15.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005. Cited on page 89.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron C. Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010. Cited on page 88.

[EDKM05] Eyal Even-Dar, Sham M Kakade, and Yishay Mansour. Experts in a markov decision process. In *NIPS*, pages 401–408, 2005. Cited on pages 72, 75, and 76.

[EDKM09] Eyal Even-Dar, Sham M Kakade, and Yishay Mansour. Online markov decision processes. *Mathematics of Operations Research*, 34(3):726–736, 2009. Cited on pages 71, 72, and 76.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017. Cited on pages 6, 7, 17, 18, 19, 20, 27, 30, 56, 86, 87, 89, 90, 91, 92, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 108, 109, 110, 112, 113, 114, 115, 116, 117, 118, 148, 152, and 153.

[FFS+18] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, pages 1563–1572, 2018. Cited on pages 6, 7, 19, 72, 80, 81, 86, 87, 95, 99, 108, and 110.

[Fri02] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. Cited on page 110.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. Cited on page 110.

[FWY22] Tao Feng, Mang Wang, and Hangjie Yuan. Overcoming catastrophic forgetting in incremental object detection via elastic response distillation. In *CVPR*, pages 9427–9436, 2022. Cited on pages 7, 20, 21, 123, 125, 126, 131, 132, 133, 135, and 149.

[FXL18] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *NeurIPS*, pages 9537–9548, 2018. Cited on pages 6, 19, 86, 108, 109, and 114.

[GBM+17] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *ICML*, pages 1311–1320, 2017. Cited on page 88.

[GFL+18] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas L. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *ICLR*, 2018. Cited on pages 6, 19, 86, 95, 99, and 108.

[Gir15] Ross Girshick. Fast r-cnn. In *ICCV*, pages 1440–1448, 2015. Cited on page 20.

[GLK20] Qiang Gao, Zhipeng Luo, and Diego Klabjan. Efficient architecture search for continual learning. *arXiv*, 2006.04027, 2020. Cited on page 44.

[Glo00] Pierre Yves Glorennec. Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*, pages 14–15. Citeseer, 2000. Cited on page 45.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014. Cited on pages 16, 27, 57, and 60.

[GPAM+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. Cited on page 15.

[Ham13] John Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013. Cited on page 48.

[HBWP13] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013. Cited on page 112.

[HCB+19] Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. In *NeurIPS*, pages 4005–4016, 2019. Cited on pages 18, 115, 116, and 117.

[HFJT19] Yu Hao, Yanwei Fu, Yu-Gang Jiang, and Qi Tian. An end-to-end architecture for class-incremental object detection with knowledge distillation. In *ICME*, pages 1–6, 2019. Cited on page 21.

[HFLR19] Shenyang Huang, Vincent François-Lavet, and Guillaume Rabusseau. Neural architecture search for class-incremental learning. *arXiv*, 1909.06686, 2019. Cited on pages 4 and 44.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, 2017. Cited on page 88.

[HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, pages 507–523, 2011. Cited on page 109.

[HKC$^+$17] Ben Harwood, Vijay Kumar, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *ICCV*, pages 2840–2848, 2017. Cited on page 89.

[HLP$^+$17] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In *ICLR*, 2017. Cited on page 110.

[HMX$^+$20] Shell Xu Hu, Pablo G Moreno, Xi Shen1 Yang Xiao, Neil D Lawrence, Guillaume Obozinski, Andreas Damianou, and France Champs-sur Marne. Empirical bayes meta-learning with synthetic gradients. In *ICLR*, 2020. Cited on pages 6, 7, 19, 20, 86, 87, 96, 98, 99, 100, 108, 109, 112, 113, 115, 116, 117, 118, and 148.

[Ho95] Tin Kam Ho. Random decision forests. In *ICDAR*, volume 1, pages 278–282, 1995. Cited on page 110.

[HPL$^+$19] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *CVPR*, pages 831–839, 2019. Cited on pages 3, 5, 14, 15, 17, 20, 26, 27, 28, 30, 34, 35, 36, 37, 42, 44, 47, 50, 51, 56, 57, 58, 60, 61, 62, 63, 64, 65, 66, 70, 71, 73, 74, 75, 78, 79, 80, 81, 82, 124, 125, 126, 131, 145, 146, 147, 151, and 152.

[HRS$^+$17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, pages 3296–3297, 2017. Cited on page 88.

[HTM$^+$21] Xinting Hu, Kaihua Tang, Chunyan Miao, Xian-Sheng Hua, and Hanwang Zhang. Distilling causal effect of data in class-incremental learning. In *CVPR*, 2021. Cited on pages 3, 14, 42, 44, 47, 51, 58, 73, 79, and 124.

[HTW$^+$19] Steven C. Y. Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning. In *NeurIPS*, pages 13647–13657, 2019. Cited on page 16.

[HVD15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv*, 1503.02531, 2015. Cited on pages 7, 14, 28, 74, 127, 128, and 149.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. Cited on pages 34, 50, 56, 58, 62, 79, 86, and 96.

[HZZ$^+$19] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, pages 558–567, 2019. Cited on page 118.

[IBM20] IBM. Artificial intelligence (AI). https://www.ibm.com/cloud/learn/what-is-artificial-intelligence, 2020. Cited on page 1.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. Cited on page 96.

[IZLS20] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *ECCV*, pages 699–715, 2020. Cited on page 16.

[JBvdL18] Cheng Ju, Aurélien Bibaut, and Mark van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics*, 45(15):2800–2818, 2018. Cited on pages 108 and 110.

[JDO+17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv*, 1711.09846, 2017. Cited on page 109.

[JKK+22] K. J. Joseph, Salman Khan, Fahad Shahbaz Khan, Rao Muhammad Anwer, and Vineeth N Balasubramanian. Energy-based latent aligner for incremental learning. In *CVPR*, pages 7452–7461, 2022. Cited on pages 14, 15, and 55.

[JKKB21] K. J. Joseph, Salman H. Khan, Fahad Shahbaz Khan, and Vineeth N. Balasubramanian. Towards open world object detection. In *CVPR*, pages 5830–5840, 2021. Cited on pages 21 and 131.

[JL20] Hong-Gyu Jung and Seong-Whan Lee. Few-shot learning with geometric constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. Cited on page 108.

[JLLP20] Kaiyi Ji, Jason D Lee, Yingbin Liang, and H Vincent Poor. Convergence of meta-learning with task-specific adaptation over partial parameters. *NeurIPS*, pages 11490–11500, 2020. Cited on page 6.

[JRK+21] K. J. Joseph, Jathushan Rajasegaran, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. Incremental object detection via meta-learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. Cited on page 123.

[JZL+18] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*, pages 2309–2318, 2018. Cited on page 89.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 1412.6980, 2014. Cited on page 97.

[KBI+17] Anna Khoreva, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele. Lucid data dreaming for object tracking. *arXiv*, 1703.09554, 2017. Cited on page 86.

[KGL17] Nitin Kamra, Umang Gupta, and Yan Liu. Deep generative dual memory network for continual learning. *arXiv*, 1710.10368, 2017. Cited on page 16.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. Cited on pages 34, 50, 62, and 79.

[KK18] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *ICLR*, 2018. Cited on page 16.

[KKKY19] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D Yoo. Edge-labeling graph neural network for few-shot learning. In *CVPR*, pages 11–20, 2019. Cited on page 117.

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. Cited on page 131.

[KMA$^+$18] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *AAAI*, pages 3390–3398, 2018. Cited on pages 26 and 56.

[KPK10] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, pages 1189–1197, 2010. Cited on page 88.

[KPR$^+$17] 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. *PNAS*, pages 3521–3526, 2017. Cited on pages 15 and 123.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2009. Cited on pages 95 and 116.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. Cited on page 108.

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. Cited on page 128.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *ICLR Workshops*, 2019. Cited on page 48.

[KVSN18] Rohit Keshari, Mayank Vatsa, Richa Singh, and Afzel Noore. Learning structure and strength of CNN filters for small sample size training. In *CVPR*, pages 9349–9358, 2018. Cited on page 88.

[KW03] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003. Cited on page 110.

[KW14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014. Cited on page 112.

[LA17] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *ICLR*, 2017. Cited on page 110.

[LAM$^+$22] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory G. Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. Cited on page 3.

[LC18] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, pages 2933–2942, 2018. Cited on pages 19, 86, 99, and 108.

[LCL19] Yingying Li, Xin Chen, and Na Li. Online optimal control with linear dynamics and predictions: Algorithms and regret analysis. In *NeurIPS*, pages 14858–14870, 2019. Cited on pages 26 and 43.

[LDM$^+$19] Huai-Yu Li, Weiming Dong, Xing Mei, Chongyang Ma, Feiyue Huang, and Bao-Gang Hu. Lgm-net: Learning to generate matching networks for few-shot learning. In *ICML*, pages 3825–3834, 2019. Cited on pages 18, 19, and 99.

[LDT+21] Wenbin Li, Chuanqi Dong, Pinzhuo Tian, Tiexin Qin, Xuesong Yang, Ziyi Wang, Huo Jing, Yinghuan Shi, Lei Wang, Yang Gao, and Jiebo Luo. Libfewshot: A comprehensive library for few-shot learning. *arXiv*, 2109.04898, 2021. Cited on page 86.

[LED+19] Hongyang Li, David Eigen, Samuel Dodge, Matthew Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. In *CVPR*, pages 1–10, 2019. Cited on pages 19, 96, 99, 100, and 117.

[Les20] Timothée Lesort. *Apprentissage continu : S'attaquer à l'oubli foudroyant des réseaux de neurones profonds grâce aux méthodes à rejeu de données. (Continual Learning : Tackling Catastrophic Forgetting in Deep Neural Networks with Replay Processes)*. PhD thesis, Institut Polytechnique de Paris, France, 2020. Cited on pages 1 and 3.

[LFP06] Fei-Fei Li, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006. Cited on pages 86 and 108.

[LH18] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018. Cited on pages 3, 14, 17, 20, 26, 28, 32, 34, 35, 36, 37, 51, 56, 70, 74, 78, 124, 127, 128, 131, 135, 145, 147, 149, 151, 152, and 154.

[LHL+21] Xinzhe Li, Jianqiang Huang, Yaoyao Liu, Qin Zhou, Shibao Zheng, Bernt Schiele, and Qianru Sun. Learning to teach and learn for semi-supervised few-shot image classification. *Computer Vision and Image Understanding*, 212:103270, 2021. Cited on pages 11, 142, and 144.

[LJD+17] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:185:1–185:52, 2017. Cited on page 109.

[LKJ+17] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, pages 4652–4662, 2017. Cited on page 15.

[LL19] Yingying Li and Na Li. Online learning for markov decision processes in nonstationary environments: A dynamic regret analysis. In *ACC*, pages 1232–1237. IEEE, 2019. Cited on page 43.

[LLP+19] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *ICLR*, 2019. Cited on pages 18, 94, 95, 98, 99, 100, 102, 116, and 153.

[LLSS23a] Yaoyao Liu, Yingying Li, Bernt Schiele, and Qianru Sun. Online hyperparameter optimization for class-incremental learning. In *AAAI*, 2023. Cited on pages 4, 10, 11, 69, and 143.

[LLSS23b] Zilin Luo, Yaoyao Liu, Bernt Schiele, and Qianru Sun. Class-incremental exemplar compression for class-incremental learning. In *CVPR*, 2023. Cited on page 12.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014. Cited on pages 20, 126, and 132.

[LMRS19] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, pages 10657–10665, 2019. Cited on pages 6, 19, 86, 96, 98, 99, 100, 116, and 117.

[LPR17] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, pages 6467–6476, 2017. Cited on page 15.

[LR17] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, pages 6467–6476, 2017. Cited on pages 3, 87, 91, 92, and 148.

[LRBG16] Jelena Luketina, Tapani Raiko, Mathias Berglund, and Klaus Greff. Scalable gradient-based tuning of continuous regularization hyperparameters. In *ICML*, pages 2952–2960, 2016. Cited on page 110.

[LSA21] Michalis Lazarou, Tania Stathaki, and Yannis Avrithis. Iterative label cleaning for transductive and semi-supervised few-shot learning. In *CVPR*, pages 8751–8760, 2021. Cited on page 107.

[LSH$^+$21] Yaoyao Liu, Qianru Sun, Xiangnan He, An-An Liu, Yuting Su, and Tat-Seng Chua. Generating face images with attributes for free. *IEEE Transactions on Neural Networks and Learning Systems*, 32(6):2733–2743, 2021. Cited on page 11.

[LSL$^+$19] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. In *NeurIPS*, pages 10276–10286, 2019. Cited on pages 6, 27, 57, 86, 94, 95, 102, and 153.

[LSL$^+$20] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *CVPR*, pages 12245–12254, 2020. Cited on pages 3, 4, 5, 9, 11, 14, 25, 44, 50, 51, 56, 57, 58, 60, 61, 62, 64, 65, 66, 73, 74, 79, 86, 88, 110, 124, 125, 126, 146, and 152.

[LSS20] Yaoyao Liu, Bernt Schiele, and Qianru Sun. An ensemble of epoch-wise empirical bayes for few-shot learning. In *ECCV*, pages 404–421, 2020. Cited on pages 10, 57, 107, 142, and 143.

[LSS21a] Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *CVPR*, 2021. Cited on pages 5, 9, 11, 41, 42, 47, 50, 51, 52, 53, 55, 70, 71, 74, 78, 79, 80, 131, 142, 143, 146, 147, 151, and 152.

[LSS21b] Yaoyao Liu, Bernt Schiele, and Qianru Sun. Rmm: Reinforced memory management for class-incremental learning. In *NeurIPS*, pages 3478–3490, 2021. Cited on pages 3, 5, 9, 11, 41, 69, 70, 71, 72, 74, 78, 79, 80, 81, 124, 144, 147, and 152.

[LSVR23] Yaoyao Liu, Bernt Schiele, Andrea Vedaldi, and Christian Rupprecht. Continual detection transformer for incremental object detection. In *CVPR*, 2023. Cited on pages 11 and 123.

[LTG$^+$19] Dawei Li, Serafettin Tasci, Shalini Ghosh, Jingwen Zhu, Junting Zhang, and Larry P. Heck. RILOD: near real-time incremental learning for object detection at the edge. In Songqing Chen, Ryokichi Onishi, Ganesh Ananthanarayanan, and Qun Li, editors, *SEC*, pages 113–126, 2019. Cited on pages 133 and 149.

[LWM$^+$20] Xialei Liu, Chenshen Wu, Mikel Menta, Luis Herranz, Bogdan Raducanu, Andrew D Bagdanov, Shangling Jui, and Joost van de Weijer. Generative feature replay for class-incremental learning. In *CVPR Workshops*, pages 226–227, 2020. Cited on pages 3 and 124.

[LWW$^+$20] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS*, 2020. Cited on pages 20 and 21.

[LWZ+21] Fanfan Liu, Haoran Wei, Wenzhe Zhao, Guozhen Li, Jingquan Peng, and Zihao Li. WB-DETR: Transformer-based detector without backbone. In *ICCV*, pages 2979–2987, 2021. Cited on page 126.

[LYR+20] Xialei Liu, Hao Yang, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Multi-task incremental learning for object detection. *arXiv*, 2002.05347, 2020. Cited on pages 21, 131, and 136.

[LZCL17] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few shot learning. *arXiv*, 1707.09835, 2017. Cited on pages 7, 87, and 110.

[LZQL19] Yingying Li, Aoxiao Zhong, Guannan Qu, and Na Li. Online markov decision processes with time-varying transition probabilities and rewards. In *ICML workshop on Real-world Sequential Decision Making*, 2019. Cited on pages 43 and 72.

[MBB13] 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. Cited on page 56.

[MC89] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989. Cited on pages 26, 42, 56, 70, 87, 91, 92, 123, and 148.

[MD17] Akshay Mehrotra and Ambedkar Dukkipati. Generative adversarial residual pairwise networks for one shot learning. *arXiv*, 1703.08033, 2017. Cited on pages 86 and 99.

[MDA15] Dougal Maclaurin, David K. Duvenaud, and Ryan P. Adams. Gradient-based hyper-parameter optimization through reversible learning. In *ICML*, pages 2113–2122, 2015. Cited on page 110.

[MH93] K. McRae and P. Hetherington. Catastrophic interference is eliminated in pre-trained networks. In *CogSci*, 1993. Cited on pages 26, 42, 56, 70, and 123.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. Cited on pages 26, 38, and 145.

[Mit97] TM Mitchell. Machine learning, mcgraw-hill higher education. *New York*, 1997. Cited on page 110.

[MLT+20] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification. *arXiv*, 2010.15277, 2020. Cited on pages 14 and 25.

[MMCSD19] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. In *ICLR*, 2019. Cited on page 110.

[MRCA18] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Snail: A simple neural attentive meta-learner. In *ICLR*, 2018. Cited on pages 19, 88, 96, and 99.

[MVL+19] Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *ICLR*, 2019. Cited on pages 27 and 28.

[MY17] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *ICML*, pages 2554–2563, 2017. Cited on pages 7, 19, 87, and 99.

[MYMT18] Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *ICML*, pages 3661–3670, 2018. Cited on pages 95, 96, and 99.

[NM92] Devang K Naik and RJ Mammone. Meta-neural networks that learn by learning. In *IJCNN*, pages 437–442, 1992. Cited on pages 6 and 86.

[ORL18] Boris N. Oreshkin, Pau Rodríguez, and Alexandre Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, pages 719–729, 2018. Cited on pages 7, 17, 18, 19, 87, 89, 95, 96, 98, 99, 100, 109, 115, 116, and 117.

[OV12] Mete Ozay and Fatos T Yarman Vural. A new fuzzy stacked generalization technique and analysis of its performance. *arXiv*, 1204.0171, 2012. Cited on pages 108 and 110.

[PDH18] Hugo Prol, Vincent Dumoulin, and Luis Herranz. Cross-modulation networks for few-shot learning. *arXiv*, 1812.00273, 2018. Cited on pages 18, 19, and 99.

[PSdV$^+$18] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, pages 3942–3951, 2018. Cited on pages 31, 34, and 88.

[PSL15] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H. Lampert. Curriculum learning of multiple tasks. In *CVPR*, pages 5492–5500, 2015. Cited on page 89.

[PTKY11] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 22(2):199–210, 2011. Cited on page 88.

[PZL20] Can Peng, Kun Zhao, and Brian C. Lovell. Faster ILOD: incremental learning for object detectors based on faster RCNN. *Pattern Recognition Letter*, 140:109–115, 2020. Cited on pages 21 and 123.

[PZM$^+$21] Can Peng, Kun Zhao, Sam Maksoud, Meng Li, and Brian C. Lovell. SID: incremental learning for anchor-free object detection via selective and inter-related distillation. *CVIU*, 210:103229, 2021. Cited on pages 132, 133, and 149.

[PZS22] Mozhgan PourKeshavarzi, Guoying Zhao, and Mohammad Sabokrou. Looking back on learned experiences for class/task incremental learning. In *ICLR*, 2022. Cited on page 14.

[QLSY18] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L. Yuille. Few-shot image recognition by predicting parameters from activations. In *CVPR*, pages 7229–7238, 2018. Cited on pages 88, 96, and 116.

[Rat90] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97:285–308, 1990. Cited on pages 3, 26, 42, 56, and 70.

[RCA$^+$19] 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. In *ICLR*, 2019. Cited on page 57.

[RCAZ16] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016. Cited on page 43.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. Cited on pages 34, 50, 56, 62, 79, 95, and 115.

[RES13] Marcus Rohrbach, Sandra Ebert, and Bernt Schiele. Transfer learning in a transductive setting. In *NIPS*, pages 46–54, 2013. Cited on page 88.

[RHGS17] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. Cited on pages 20 and 21.

[RHK$^+$19] Jathushan Rajasegaran, Munawar Hayat, Salman H Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for continual learning. In *NeurIPS*, pages 12669–12679, 2019. Cited on pages 16 and 17.

[RKH$^+$20] Jathushan Rajasegaran, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. itaml: An incremental task-agnostic meta-learning approach. In *CVPR*, pages 13588–13597, 2020. Cited on page 57.

[RKH$^+$21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *ICML*, pages 8748–8763, 2021. Cited on page 142.

[RKSL17] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, pages 5533–5542, 2017. Cited on pages 3, 4, 5, 7, 14, 15, 17, 20, 26, 27, 28, 30, 32, 34, 35, 36, 37, 41, 42, 43, 44, 49, 50, 51, 56, 57, 60, 61, 62, 65, 66, 70, 71, 73, 74, 75, 78, 79, 81, 124, 125, 126, 131, 135, 136, 145, 146, 147, 151, and 152.

[RL17] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. Cited on pages 17, 18, 19, 89, 95, 96, 99, and 110.

[RMM$^+$17] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *CVPR*, pages 7008–7024, 2017. Cited on page 48.

[RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014. Cited on page 112.

[RRD$^+$16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv*, 1606.04671, 2016. Cited on page 16.

[RRS$^+$19] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019. Cited on pages 6, 17, 18, 19, 20, 86, 88, 89, 96, 99, 100, and 117.

[RTR$^+$18] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018. Cited on pages 6, 7, 86, 87, 94, 95, 98, 100, 102, 109, 115, and 153.

[SAN16] Russell Stewart, Mykhaylo Andriluka, and Andrew Y Ng. End-to-end people detection in crowded scenes. In *CVPR*, pages 2325–2333, 2016. Cited on page 128.

[SB20] Fátima Saiz and Iñigo Barandiaran. Covid-19 detection in chest x-ray images using a deep learning approach. *International Journal of Interactive Multimedia and Artificial Intelligence*, 2020. Cited on page 86.

[SBB+16] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016. Cited on page 19.

[SCC20] Jae Woong Soh, Sunwoo Cho, and Nam Ik Cho. Meta-transfer learning for zero-shot super-resolution. In *CVPR*, pages 3516–3525, 2020. Cited on page 86.

[SCD+17] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *CVPR*, pages 618–626, 2017. Cited on pages 64, 66, 80, 81, 146, and 147.

[SCH+16] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. In *ACL*, 2016. Cited on page 43.

[SCYK21] Zhiqing Sun, Shengcao Cao, Yiming Yang, and Kris M Kitani. Rethinking transformer-based set prediction for object detection. In *ICCV*, pages 3611–3620, 2021. Cited on pages 125 and 126.

[SDC+19] Benoit Steiner, Zachary DeVito, Soumith Chintala, Sam Gross, Adam Paszke, Francisco Massa, Adam Lerer, Gregory Chanan, Zeming Lin, Edward Yang, et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019. Cited on pages 31 and 114.

[SE18] Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018. Cited on pages 18 and 99.

[SGG16] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, pages 761–769, 2016. Cited on pages 87, 89, and 104.

[SGNK17] Nikolaos Sarafianos, Theodore Giannakopoulos, Christophoros Nikou, and Ioannis A. Kakadiaris. Curriculum learning for multi-task classification of visual attributes. In *ICCV Workshops*, 2017. Cited on page 88.

[SK96] Peter Sollich and Anders Krogh. Learning with ensembles: How overfitting can be useful. In *NIPS*, pages 190–196, 1996. Cited on page 110.

[SKH21] Christian Simon, Piotr Koniusz, and Mehrtash Harandi. On learning the geodesic path for incremental learning. In *CVPR*, pages 1591–1600, 2021. Cited on page 14.

[SKS+18] Eli Schwartz, Leonid Karlinsky, Joseph Shtok, Sivan Harary, Mattias Marder, Rogério Schmidt Feris, Abhishek Kumar, Raja Giryes, and Alexander M. Bronstein. Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *NeurIPS*, pages 2850–2860, 2018. Cited on pages 86 and 99.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012. Cited on page 109.

[SLC+22] Qianru Sun, Yaoyao Liu, Zhaozheng Chen, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning through hard tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1443–1456, 2022. Cited on pages 10, 11, 19, 20, 27, 61, 86, 115, 116, and 143.

[SLCS19] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, pages 403–412, 2019. Cited on pages 5, 6, 7, 10, 19, 20, 27, 31, 34, 58, 61, 64, 86, 96, 102, 108, 109, 112, 113, 114, 115, 116, 117, and 118.

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017. Cited on page 86.

[SLKK17] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, pages 2990–2999, 2017. Cited on pages 15, 16, 26, and 56.

[SMD18] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2018. Cited on pages 27 and 56.

[SRM18] Tyler R. Scott, Karl Ridgeway, and Michael C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *NeurIPS*, pages 76–85, 2018. Cited on page 88.

[SRS$^+$15] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, pages 2171–2180, 2015. Cited on page 110.

[SSA17] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, pages 3420–3429, 2017. Cited on pages 7, 20, 123, 124, 125, 126, and 142.

[SSF17] Qianru Sun, Bernt Schiele, and Mario Fritz. A domain based approach to social relation recognition. In *CVPR*, pages 435–444, 2017. Cited on page 88.

[SSZ17] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4077–4087, 2017. Cited on pages 18, 74, 87, 91, 94, 98, 99, 100, 101, 102, 116, 117, 148, and 153.

[SSZA14] Jasper Snoek, Kevin Swersky, Richard S. Zemel, and Ryan P. Adams. Input warping for bayesian optimization of non-stationary functions. In *ICML*, pages 1674–1682, 2014. Cited on page 109.

[SW99] Padhraic Smyth and David Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83, 1999. Cited on page 110.

[SXH$^+$21] Xi Shen, Yang Xiao, Shell Xu Hu, Othman Sbai, and Mathieu Aubry. Re-ranking for image retrieval and transductive few-shot classification. *NeruIPS*, pages 25932–25943, 2021. Cited on page 107.

[SYZ$^+$18] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, pages 1199–1208, 2018. Cited on pages 18, 87, 91, 94, 98, 99, 100, 101, 102, 115, 148, and 153.

[SZL$^+$22] Yujun Shi, Kuangqi Zhou, Jian Liang, Zihang Jiang, Jiashi Feng, Philip HS Torr, Song Bai, and Vincent YF Tan. Mimicking the oracle: An initial phase decorrelation approach for class incremental learning. In *CVPR*, pages 16722–16731, 2022. Cited on page 55.

[TABT17] Amal Rannen Triki, Rahaf Aljundi, Matthew B. Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *ICCV*, pages 1329–1337, 2017. Cited on page 15.

[TCH+20] Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. Topology-preserving class-incremental learning. In *ECCV*, pages 254–270, 2020. Cited on pages 14, 15, 51, and 65.

[TP98] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998. Cited on pages 6 and 86.

[VBL+16] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, pages 3630–3638, 2016. Cited on pages 7, 17, 18, 87, 89, 90, 94, 95, 96, 98, 99, 101, 102, 109, 110, 115, 116, 117, and 153.

[VDR21] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. In *ICLR*, 2021. Cited on page 44.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, pages 5998–6008, 2017. Cited on page 125.

[VSV52] Heinrich Von Stackelberg and Stackelberg Heinrich Von. *The theory of the market economy*. Oxford University Press, 1952. Cited on pages 27 and 57.

[VVPL17] Ragav Venkatesan, Hemanth Venkateswara, Sethuraman Panchanathan, and Baoxin Li. A strategy for an uncompromising incremental learner. *arXiv*, 1705.00744, 2017. Cited on page 16.

[VYP+22] Eli Verwimp, Kuo Yang, Sarah Parisot, Hong Lanqing, Steven McDonagh, Eduardo Pérez-Pellitero, Matthias De Lange, and Tinne Tuytelaars. Re-examining distillation for continual object detection. In *BMVC*, 2022. Cited on page 123.

[WCA18] Daphna Weinshall, Gad Cohen, and Dan Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *ICML*, pages 5235–5243, 2018. Cited on page 88.

[WCW+19] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, pages 374–382, 2019. Cited on pages 15, 26, 28, 30, 34, 35, 36, 37, 50, 56, 57, 65, 79, 131, 145, and 151.

[Wel09] Max Welling. Herding dynamical weights to learn. In *ICML*, pages 1121–1128, 2009. Cited on pages 15, 26, 57, and 146.

[WGHH18] Yu-Xiong Wang, Ross B. Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *CVPR*, pages 7278–7286, 2018. Cited on page 86.

[WHD+20] Xin Wang, Thomas E. Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. In *ICML*, 2020. Cited on page 108.

[WHL+18] Chenshen Wu, Luis Herranz, Xialei Liu, Yaxing Wang, Joost van de Weijer, and Bogdan Raducanu. Memory replay gans: Learning to generate new categories without forgetting. In *NeurIPS*, pages 5966–5976, 2018. Cited on page 15.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. Cited on pages 43, 44, 45, and 48.

[WYKN20] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020. Cited on page 5.

[WZHY18] Ying Wei, Yu Zhang, Junzhou Huang, and Qiang Yang. Transfer learning via learning to transfer. In *ICML*, pages 5085–5094, 2018. Cited on page 88.

[WZTE18] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv*, 1811.10959, 2018. Cited on pages 27 and 57.

[WZY$^+$22] Liyuan Wang, Xingxing Zhang, Kuo Yang, Longhui Yu, Chongxuan Li, Lanqing Hong, Shifeng Zhang, Zhenguo Li, Yi Zhong, and Jun Zhu. Memory replay with data compression for continual learning. In *ICLR*, 2022. Cited on pages 3, 15, 17, and 124.

[WZYZ22] Fu-Yun Wang, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. Foster: Feature boosting and compression for class-incremental learning. In *ECCV*, 2022. Cited on pages 3, 4, 14, 16, 78, and 152.

[Xia19] Liu Xialei. *Visual recognition in the wild: learning from rankings in small domains and continual learning in new domains*. PhD thesis, Universitat Autònoma de Barcelona, Spain, 2019. Cited on page 14.

[Xia20] Yongqin Xian. *Learning from limited labeled data - Zero-Shot and Few-Shot Learning*. PhD thesis, Saarland University, Saarbrücken, Germany, 2020. Cited on pages 1, 6, and 17.

[XL22] Jingyi Xu and Hieu Le. Generating representative samples for few-shot classification. In *CVPR*, pages 9003–9013, 2022. Cited on page 107.

[XSSA19] Yongqin Xian, Saurabh Sharma, Bernt Schiele, and Zeynep Akata. f-VAEGAN-D2: A feature generating framework for any-shot learning. In *CVPR*, pages 10275–10284, 2019. Cited on pages 86 and 88.

[XZ18] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *NeurIPS*, pages 899–908, 2018. Cited on pages 4, 16, 17, 43, 44, 45, 48, and 72.

[Yav20] Najib Yavari. *Few-Shot Learning with Deep Neural Networks for Visual Quality Control: Evaluations on a Production Line*. PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY, 2020. Cited on pages 2 and 139.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, pages 3320–3328, 2014. Cited on pages 59 and 68.

[YDS$^+$22] Binbin Yang, Xinchi Deng, Han Shi, Changlin Li, Gengwei Zhang, Hang Xu, Shen Zhao, Liang Lin, and Xiaodan Liang. Continual object detection via prototypical task correlation guided gating mechanism. In *CVPR*, pages 9255–9264, 2022. Cited on page 123.

[YHZS18] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Learning embedding adaptation for few-shot learning. *arXiv*, 1812.03664, 2018. Cited on page 96.

[YHZS20] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *CVPR*, 2020. Cited on pages 18, 96, and 116.

[YKD$^+$18] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *NeurIPS*, pages 7343–7353, 2018. Cited on pages 109, 110, and 113.

[YTL$^+$20] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *CVPR*, pages 6982–6991, 2020. Cited on page 15.

[YXH21] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, pages 3014–3023, 2021. Cited on pages 16, 78, and 152.

[YYG15] LeCun Yann, Bengio Yoshua, and Hinton Geoffrey. Deep learning. *Nature*, 521(7553):436, 2015. Cited on page 86.

[YYH07] Jun Yang, Rong Yan, and Alexander G. Hauptmann. Adapting SVM classifiers to data with shifted distributions. In *ICDM Workshops*, 2007. Cited on page 88.

[YZZ$^+$22] Dongbao Yang, Yu Zhou, Aoting Zhang, Xurui Sun, Dayan Wu, Weiping Wang, and Qixiang Ye. Multi-view correlation distillation for incremental object detection. *Pattern Recognition*, 131:108863, 2022. Cited on pages 7 and 21.

[ZCG$^+$18] Ruixiang Zhang, Tong Che, Zoubin Grahahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *NeurIPS*, pages 2371–2380, 2018. Cited on pages 6, 19, 86, 99, and 108.

[ZCLS20] Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers. In *CVPR*, pages 12203–12213, 2020. Cited on pages 27, 57, and 108.

[ZCLS22] Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. Deepemd: Differentiable earth mover's distance for few-shot learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. Cited on page 19.

[ZCS$^+$20] Wang Zhou, Shiyu Chang, Norma Sosa, Hendrik Hamann, and David Cox. Lifelong object detection. *arXiv*, 2009.01129, 2020. Cited on pages 7 and 21.

[ZCT$^+$19] Mengyao Zhai, Lei Chen, Frederick Tung, Jiawei He, Megha Nawhal, and Greg Mori. Lifelong gan: Continual learning for conditional image generation. In *ICCV*, pages 2759–2768, 2019. Cited on page 16.

[ZL17] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. Cited on pages 43, 45, and 48.

[ZLL$^+$19a] Chi Zhang, Guosheng Lin, Fayao Liu, Jiushuang Guo, Qingyao Wu, and Rui Yao. Pyramid graph networks with connection attentions for region-based one-shot semantic segmentation. In *ICCV*, pages 9587–9595, 2019. Cited on page 108.

[ZLL$^+$19b] Chi Zhang, Guosheng Lin, Fayao Liu, Rui Yao, and Chunhua Shen. Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning. In *CVPR*, pages 5217–5226, 2019. Cited on pages 27 and 108.

[ZLL$^+$22] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. DINO: DETR with improved denoising anchor boxes for end-to-end object detection. *arXiv*, 2203.03605, 2022. Cited on page 126.

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Doina Precup and Yee Whye Teh, editors, *ICML*, pages 3987–3995, 2017. Cited on page 15.

[ZSC$^+$19] Le Zhang, Zenglin Shi, Ming-Ming Cheng, Yun Liu, Jia-Wang Bian, Joey Tianyi Zhou, Guoyan Zheng, and Zeng Zeng. Nonlinear regression via deep negative correlation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. Cited on page 110.

[ZSL+21] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. In *ICLR*, 2021. Cited on pages 7, 8, 21, 123, 124, 125, 126, 127, 132, 134, 135, 137, 142, 149, and 154.

[ZSS+18] Amir Roshan Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *CVPR*, pages 3712–3722, 2018. Cited on page 88.

[ZWYZ21] Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, and De-Chuan Zhan. Pycil: A python toolbox for class-incremental learning. *arXiv*, 2112.12533, 2021. Cited on page 41.

[ZXG+20] Bowen Zhao, Xi Xiao, Guojun Gan, Bin Zhang, and Shu-Tao Xia. Maintaining discrimination and fairness in class incremental learning. In *CVPR*, pages 13208–13217, 2020. Cited on pages 3, 79, and 124.

[ZZC+22] Kai Zhu, Wei Zhai, Yang Cao, Jiebo Luo, and Zhengjun Zha. Self-sustaining representation expansion for non-exemplar class-incremental learning. In *CVPR*, pages 9286–9295, 2022. Cited on page 142.

[ZZW+21] Fei Zhu, Xu-Yao Zhang, Chuang Wang, Fei Yin, and Cheng-Lin Liu. Prototype augmentation and self-supervision for incremental learning. In *CVPR*, pages 5871–5880, 2021. Cited on page 19.