박 사 학 위 논 문
Ph.D. Dissertation

실세계를 위한 온-디바이스, 온라인 계속적 학습

On-device, Online Continual Learning
for the Real World

2023

윤 재 홍 (Yoon, Jaehong)

한 국 과 학 기 술 원

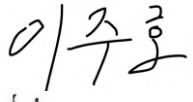
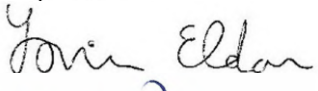
Korea Advanced Institute of Science and Technology

실세계를 위한 온-디바이스, 온라인 계속적 학습

윤 재 홍

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2022년 11월 30일

심사위원장	Sung Ju Hwang	(인)	
심 사 위 원	Eunho Yang	(인)	
심 사 위 원	Juho Lee	(인)	
심 사 위 원	Yonina C. Eldar	(인)	
심 사 위 원	Razvan Pascanu	(인)	

On-device, Online Continual Learning for the Real World

Jaehong Yoon

Advisor: Sung Ju Hwang

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
February 28, 2023

Approved by

Sung Ju Hwang
Professor of Graduate School of AI & School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 윤재홍. 실세계를 위한 온-디바이스, 온라인 계속적 학습. 전산학부 . 2023년. 111+v 쪽. 지도교수: 황성주. (영문 논문)
Jaehong Yoon. On-device, Online Continual Learning for the Real World. School of Computing . 2023. 111+v pages. Advisor: Sung Ju Hwang. (Text in English)

초 록

인간은 시간이 지남에 따라 새로운 지식과 기술을 학습, 병합하며 더 다양하고 큰 문제를 해결할 수 있는 능력을 갖는다. 이러한 사람의 학습 능력을 닮은 시스템을 구축하는 것은 긴 시간동안 지속가능한 학습 능력이 일반 인공지능의 구현에 있어서 핵심적이기에 연구자들의 오랜 소망이었다. 이러한 필요에 따라, 계속적 학습, 평생 기계학습 분야는 다양한 분야에 걸쳐 단일 모델이 순차적으로 들어오는 다양한 테스크를 계속적으로 학습하는 문제를 제안하였다. 본 저자의 장기적 연구 목표는 많은 수의 지속가능한 온-디바이스 온라인 계속적 학습 에이전트들을 포함한 포괄적 학습 시스템을 구축하는 것이며 이 때 임베디드 머신들은 끊임없이 들어오는 비정체 1인칭 비디오 스트림으로 표현된 장비 착용자의 혹은 장비 자체의 경험을 학습하는 것을 목표로 한다. 각각 에이전트들은 독립적으로 그들의 경험을 학습하나 중앙 서버의 중재를 통한 지식 공유 및 확장을 가능케 한다. 이 때 서버는 정보의 축적과 병합을 수행하고 시스템에 참여한 에이전트들의 필요에 따라 적합한 정보를 송신한다. 요약하면, 이러한 학습 시스템에서 본 저자는 세 가지 해결해야할 문제를 제시한다. 첫 번째, 온-디바이스 AI 는 하드웨어 종류와 해결해야하는 문제의 어려움에 따라 적합하게 제한된 메모리/컴퓨팅 리소스를 활용해 문제를 해결해야 한다. 두 번째, 에이전트들은 다른 에이전트들의 경험과 지식을 필요에 따라 선택적으로 전이할 수 있어야 한다. 세 번째, 본 시스템은 실세계 데이터를 실시간으로 학습하는 것을 목표로 하고 있으며, 이 때 실세계 데이터는 많은 수의 중복, 불균형, 노이즈가 포함된 정보를 함유하고 있을 뿐 아니라 레이블 정보가 없을 수도 있다. 즉, 학습 모델은 정제되지 않거나 레이블링되어있지 않은 데이터를 안정적으로 학습할 수 있어야 한다. 본 학위 논문에서는 앞서 말한 문제점을 해결하여 실세계를 위한 지속가능한 온-디바이스 온라인 계속적 학습을 구현하기 위해 다양한 관점에서 연구를 수행하였다.

핵 심 낱 말 계속적 학습, 평생 기계학습, 온-디바이스 학습, 연합 학습, 표현형 학습, 온라인 학습

Abstract

Humans possess the ability to learn a large number of tasks by accumulating knowledge and skills over time. Building a system resembling human learning abilities is a deep-rooted desire since sustainable learning over a long-term period is essential for general artificial intelligence. In light of this need, continual learning (CL), or lifelong learning, tackles a learning scenario where a model continuously learns over a sequence of tasks within a broad research area. For my long-term research goal, I aim to encompass broad research fields to understand humans and impact our real lives through sustainable on-device AI systems with multiple agents. Embedded machines persistently learn human users' experiences, formulated to non-stationary online egocentric video streams. While multiple agents learn their local experiences separately, they can communicate with each other to expand and evolve their knowledge hosted by the server. The server contributes to stacking and merging the local knowledge on various problems and re-distributing them to participating agents on demand. However, this learning paradigm raises several crucial challenges, First, on-device AI has limited computational and memory budgets according to the hardware and difficulties of local sequential tasks. Second, the agent should adaptively transfer the relevant knowledge from other devices' experience. Further, we aim to train on real-world

data in an online manner, and the real-world data contain a large amount of redundant, imbalanced, and noisy, as well as unlabeled instances. That is, the model should cope with not refined and unannotated training data. And this thesis proposes methods from multiple perspectives for solving these challenges of sustainable on-device, online continual learning for the real world.

Keywords continual learning, lifelong machine learning, on-device learning, federated learning, representation learning, online learning

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Lifelong Machine Learning

Humans possess the ability to learn a large number of tasks throughout a lifespan by accumulating and improving knowledge over time. Building a system resembling human learning abilities is essential for *Artificial General Intelligence* (AGI) since sustainable learning over a long-term time period is an important aspect of human intelligence. Suppose that a model trains on ImageNet [78] 22k classes at the beginning. Depending on your needs, you may ask a model to perform more than a million fine-grained classifications. Also, new classes that require classification may be added over time. That is, not all training data might be available when we want to begin training the network in many real-world scenarios, and real-world AGI should manage such irreversible shifts in data distributions (Please see Figure 1.1).

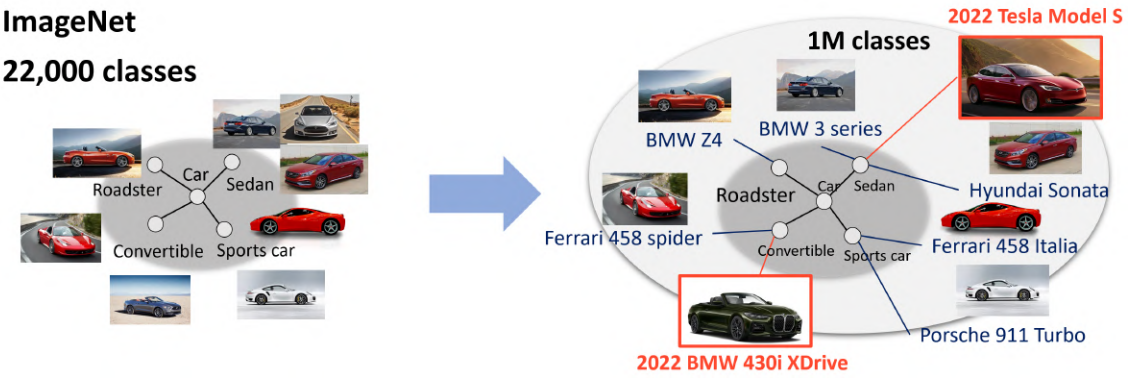


Figure 1.1: **In real-world scenarios, all training data might not be available when we begin to train the network**.

In light of this need, *Continual Learning* (CL) [152] or *lifelong learning*, a paradigm for learning a series of tasks in a sequential manner [80, 97], has been gaining in popularity. One of the major goals of a continual learner is to mimic human cognition, exemplified by the ability to persistently learn new concepts over his/her lifespan. While continual learning is considered a broad field, such as classification [74, 19], image generation [182], language learning [96, 9], clinical application [86, 91], speech recognition [132], and federated learning [174], it is basically organized into the following training scheme: First, a single model receives tasks in sequential order while losing the accessibility of past task data when a model meets new tasks. Next, a model transfers the knowledge from previously learned tasks to solve current task problems. Here, continual learning assumes that the model no longer has access to past data, so the model loses fidelity to past tasks while refining previously learned knowledge to incoming tasks. Conventional continual learning methods have focused on alleviating this knowledge-forgetting issue, which is a well-known challenge, often called *catastrophic forgetting* [104] or *catastrophic interference*.

The mammalian brain avoids catastrophic forgetting by protecting previously-acquired knowledge in neocortical circuits as long-term memory. When external sensory information arrives, the brain encodes it in the hippocampus (initial memory). Neocortical synapses then consolidate the acquired knowledge to be stable through the repeated replay. After that, initial memory is renormalized for future training.

While continual learning [80, 131] is a long-studied topic with a vast literature, we only discuss recent relevant works and broadly categorize them in three directions. Multiple works adopt more than one approach to a single continual learning model. **(1) Regularization-based** approach [74, 90, 137, 109, 153, 108] is to use regularizations that prevent catastrophic forgetting. **(2) Architecture-based** approach [130, 178, 170, 95, 175] introduces additional learnable parameters to overcome the limited expressiveness issue in continual learning. **(3) Rehearsal-based** approach [102, 125, 19, 4, 3, 20] stores a few instances of past task data in a fixed-memory size replay buffer and revisits them during future task training to mitigate forgetting.

1.2 Sustainable On-device Artificial Intelligence

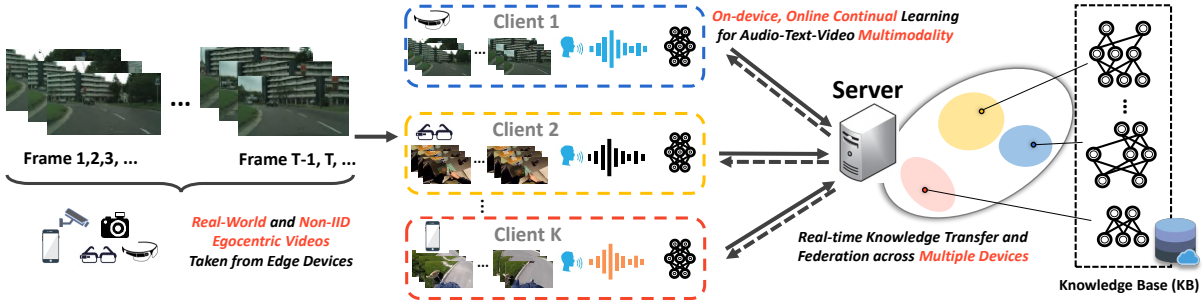


Figure 1.2: **Multi-agent On-device, Online Continual Learning System over Real-world Multimodal Data**.

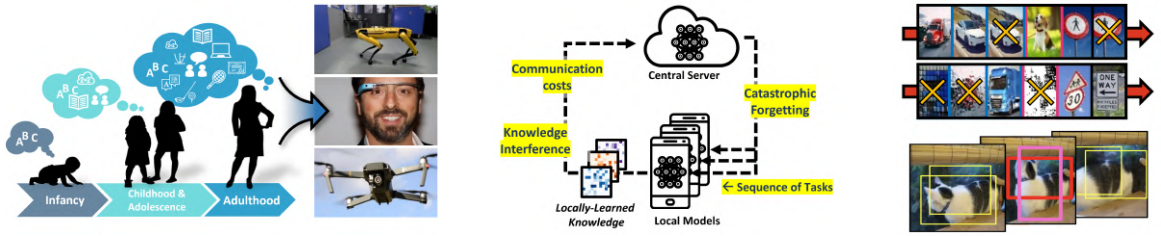
For my long-term research goal, I aim to encompass broad research fields to understand humans and impact our real lives through sustainable on-device AI systems with multiple agents illustrated in Figure 1.2. Embedded machines persistently learn human users' experiences, formulated to non-stationary online egocentric video streams containing audio-video multimodality. While multiple agents learn their local experiences separately, they can communicate with each other to expand and evolve their knowledge hosted by the central server. The server contributes to stacking and merging the local knowledge on various problems and re-distributing them to participating agents on demand. However, this learning paradigm raises several crucial challenges:

Resource requirement/allowance varies according to the devices and tasks. On-device AI has limited computational and memory budgets according to devices and difficulties of local sequential tasks. One of the critical factors in the success of on-device continual learners is effectively utilizing limited hardware resources as models train on new tasks continuously. Moreover, as on-device learning machines often receive an unlimited sequence of tasks in real-time and immediately lose accessibility to the past data corpus, the knowledge obtained from past data/tasks is more susceptible to overwritten with new ones.

Machines limit their training sources from direct experience. Humans can learn from indirect experiences from others via different means (e.g., making conversations and reading books). In a similar vein, allowing communication across multiple on-device continual learners is pragmatic and attractive as they efficiently extend knowledge with data privacy-preserving weight transfer beyond the resource limits in a single device. And the server should transmit knowledge to heterogeneous tasks and devices adaptively.

Training instances are often not refined/annotated. As we aim to train on real-world data in

an online manner, the model should cope with not refined and unannotated training data. While most deep learning algorithms assume that all the instances are sufficiently informative and valuable, data in the wild collected by personal devices include a large amount of redundant and noisy inputs, which may degrade the model adaptation for target problems and yield severe forgetting of past knowledge. Also, as the annotation process is time-consuming and expensive in most on-device, online learning setups, capturing representations on unlabeled data streams is essential.



(a) Efficient knowledge increment with limited memory resource

(b) Communicability across multiple devices

(c) Not refined or unannotated data in the wild

Figure 1.3: **Challenges of on-device continual learning for real-world**.

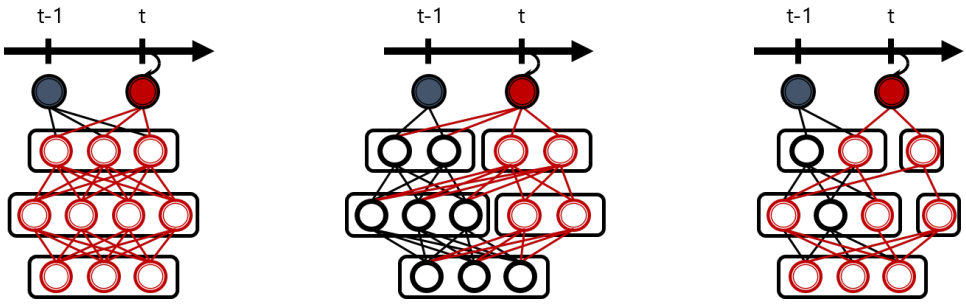
During my Ph.D., I have tackled the practical challenges of deploying on-device, online continual learning methods from varying perspectives in the following chapters. We have presented scalable and efficient continual learning methods [178, 175] to tackle the limited resource allowance issue for on-device continual learning in Chapter 2. To transfer indirect experience across multiple agents, I have worked on solving *Federated Learning* for continual [174] learning tasks or for bitwidth heterogeneous devices [177] in Chapter 3. Lastly, I have addressed data in the wild problem for uncurated [176] and unsupervised [103] continual learning, as well as video representation learning [58] in Chapter 4. Then, I will describe my ongoing and future research plan in Chapter 5.

Chapter 2. Scalable and Efficient Continual Learning

2.1 Lifelong Learning with Dynamically Expandable Networks [178]

2.1.1 Motivation

While many different approaches exist to tackle Lifelong learning [152], we consider it under deep learning to exploit the power of deep neural networks. Fortunately, storing and transferring knowledge can be done straightforwardly for deep learning through the learned network weights. The learned weights can serve as the knowledge for the existing tasks, and the new task can leverage this by simply sharing these weights. Therefore, we can consider lifelong learning simply as a particular case of online or incremental learning in the case of deep neural networks. There are multiple ways to perform such incremental learning [130, 188]. The simplest way is to incrementally fine-tune the network to new tasks by continuing to train the network with new training data. However, such simple retraining of the network can degenerate the performance for both the new and old tasks. If the new task is largely different from the older ones, such as in the case where previous tasks are classifying images of animals, and the new task is to classify images of cars, then the features learned on the previous tasks may not be helpful for the new one. At the same time, the retrained representations for the new task could adversely affect the old tasks, as they may have drifted from their original meanings and are no longer optimal for them. For example, the feature describing the stripe pattern from zebra may change its meaning for the later classification task for classes such as striped t-shirts or fences, which can fit the feature and drastically change its meaning.



(a) Retraining w/o expansion (b) No-retraining w/ expansion (c) Partial retraining w/ expansion

Figure 2.1: **Concept:** (a) Retraining models such as Elastic Weight Consolidation (EWC) [74] retrain the entire network learned on previous tasks while regularizing it to prevent significant deviation from the original model. Retrained units and weights are highlighted in red and fixed ones in black. (b) Non-retraining models such as Progressive Network [130] expand the network for the new task t while withholding modification of network weights for previous tasks. (c) Our DEN selectively retrains the old network, expanding its capacity when necessary and thus dynamically deciding its optimal capacity as it trains on.

Then how can we ensure that the knowledge sharing through the network is beneficial for all tasks in the online/incremental learning of a deep neural network? Existing regularization-based models, like *Elastic Weight Consolidation* [74] or *Synaptic Intelligence* [181], use a regularizer to prevent the parameters from drastically changing their values. They are applicable for on-device AIs with limited memory capacity

but are often suboptimal when tasks increase due to their fixed expressiveness. *Progressive Network* [130] expands the network with a fixed number of neurons for incoming tasks while blocking any changes to the old task weights, resulting in a linear increase in the model size.

Our strategy differs from both approaches since we retrain the network at each task so that each new task utilizes and changes only the relevant part of the previously trained network while allowing it to expand the network capacity when necessary. In this way, each task will use a different subnetwork from the previous tasks while still sharing a considerable part of the subnetwork with them. Figure 2.1 illustrates our model in comparison with existing deep lifelong learning methods. There are a number of challenges that need to be tackled for such incremental deep learning settings with selective parameter sharing and dynamic layer expansion.

1) Achieving scalability and efficiency in training: If the network grows in capacity, training cost per task will increasingly grow as well since the later tasks will establish connections to a much larger network. Thus, we need a way to keep the computational overhead of retraining to be low.

2) Deciding when to expand the network, and how many neurons to add: The network does not need to expand its size if the old network sufficiently explains the new task. On the other hand, it might need to add in many neurons if the task is very different from the existing ones. Hence, the model needs to add only the necessary number of neurons on demand.

3) Preventing *semantic drift*, or *catastrophic forgetting*: Our method might also negatively affect the prior tasks by establishing connections to an old subnetwork even if we partially retrain the network to fit later learned tasks and add new neurons. That is, we need a mechanism to prevent potential semantic drift.

To overcome such challenges, we propose a novel deep network model with an efficient and effective incremental learning algorithm named *Dynamically Expandable Networks* (DEN). In a lifelong learning scenario, DEN maximally utilizes the network learned on all previous tasks to efficiently learn to predict for the new task while dynamically increasing the network capacity by adding in or splitting/duplicating neurons when necessary. Our method is applicable to any deep generic networks, including convolutional networks. We validate our incremental deep neural network for lifelong learning on multiple public datasets, on which it achieves similar or better performance than the model that trains a separate network for each task while using only $11.9\%p - 60.3\%p$ of its parameters. Further, fine-tuning the learned network on all tasks obtains even better performance, outperforming the batch model by as much as $0.05\%p - 4.8\%p$. Thus, our model can be also used for structure estimation to obtain optimal performance over network capacity even when batch training is possible, which is a more general setup.

2.1.2 Related Work

Lifelong learning Lifelong learning is the learning paradigm for continual learning where the model learns from a sequence of tasks while transferring knowledge obtained from earlier tasks to later ones. It has been extensively studied due to its practicality in scenarios where the data arrives in streams, such as autonomous driving or learning of robotic agents. Lifelong learning often tackles an online multi-task learning problem, focusing on efficient training with knowledge transfer. ELLA [131] suggests an online lifelong learning framework based on an existing multi-task learning formulation that efficiently updates latent parameter bases for a sequence of tasks. The method removes the dependency on previous tasks to learn each task predictor and prevent retraining previous task predictors. Some recent work focus on the theoretical aspects of lifelong learning. *Pentina and Lampert* [118] provide a PAC-Bayesian bound for lifelong learning, and they further explored lifelong learning with non-i.i.d. tasks [119]. *Pentina et*

al. [120] propose an algorithm that predicts each task with the majority votes over all previous task predictors. Recently, lifelong learning has been studied in deep learning frameworks; since lifelong learning of a deep neural network can be straightforwardly done by simple retraining, the primary focus of research is overcoming catastrophic forgetting [74, 130, 181, 90].

Preventing catastrophic forgetting Incremental or lifelong learning of deep networks results in the problem known as catastrophic forgetting, which describes the network forgetting learned for previous tasks when retraining the network for new tasks. One solution to this problem is to use a regularizer that prevents the new model from deviating too much from the previous one, such as ℓ_2-regularizer. However, using the simple ℓ_2-regularizer prevents the model from learning new knowledge for the new tasks, resulting in suboptimal performances on later tasks. To overcome this limitation, Elastic Weight Consolidation (EWC) [74] regularizes the model parameter at each step with the model parameter at the previous iteration via the Fisher information matrix for the current task, which enables us to find a good solution for both tasks. Synaptic Intelligence (SI) [181] proposes a similar approach, but their approach computes the per-synapse consolidation in an online manner and considers the entire learning trajectory rather than the final parameter value. Another way to prevent catastrophic forgetting is to block any previous network modifications completely. At each learning stage, Progressive Network [130] expands the network with a subnetwork with fixed capacity and trains it along with incoming weights from the original network, which is not backpropagated in the process.

2.1.3 Approach

We consider the problem of incremental training of a deep neural network under the lifelong learning scenario, where the unknown number of tasks with unknown training data distributions arrive at the model in sequence. Specifically, our goal is to learn models for a sequence of T tasks, $t = 1, \cdots, t, \cdots, T$ for *unbounded* T where the task at time point t comes with training data $\mathcal{D}_t = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N_t}$. Note that each task can be either a single task or a comprised set of subtasks. While our method is generic to any task, we only consider the binary classification task for simplification, that is, $y \in \{0, 1\}$ for input feature $\boldsymbol{x} \in \mathbb{R}^d$. The main challenge in the lifelong learning setting is that all the previous training datasets up to $t-1$ are not available at the current time t and only the model parameters for the previous tasks are accessible. The lifelong learning agent at time t aims to learn the model parameter $\boldsymbol{W}^t$ by solving the following problem:

$$\underset{\boldsymbol{W}^t}{minimize}\ \mathcal{L}\left(\boldsymbol{W}^t; \boldsymbol{W}^{t-1}, \mathcal{D}_t\right) + \lambda\Omega\left(\boldsymbol{W}^t\right), \qquad \forall t \in \{1, \ldots, T\} \tag{2.1}$$

where $\mathcal{L}$ is task-specific loss function, $\boldsymbol{W}^t$ is the parameter for task t, and $\Omega\left(\boldsymbol{W}^t\right)$ is the regularization (e.g., element-wise ℓ_2 norm) to enforce our model $\boldsymbol{W}^t$ appropriately. In the case of a neural network which is our primary interest, $\boldsymbol{W}^t = \{\boldsymbol{W}_l\}_{l=1}^{L}$ is the set of weight tensors for all layers. To tackle these challenges of lifelong learning, we let the network maximally utilize the knowledge obtained from the previous tasks while allowing it to dynamically expand its capacity when the accumulated knowledge alone cannot sufficiently explain the new task. Figure 2.2 and Algorithm 1 describe our incremental learning process.

In the following subsections, we describe each component of our incremental learning algorithm in detail: 1) Selective retraining, 2) Dynamic network expansion, and 3) Network split/duplication.

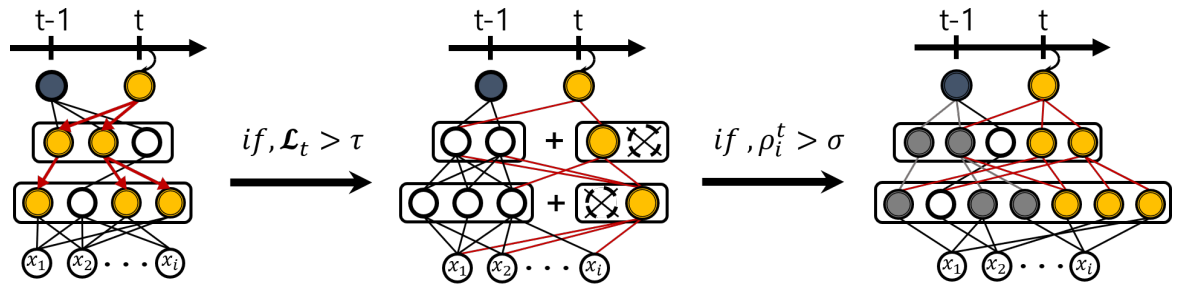


Figure 2.2: **Incremental learning of a dynamically expandable network: Left:** *Selective retraining.* DEN first identifies neurons that are relevant to the new task and selectively retrains the network parameters associated with them. **Center:** *Dynamic network expansion.* If the selective retraining fails to obtain desired loss below the threshold, we expand the network capacity in a top-down manner, while eliminating any unnecessary neurons using group-sparsity regularization. **Right:** *Network split/duplication.* DEN calculates the drift ρ_i^t for each unit to identify units that have drifted too much from their original values during training and duplicate them.

Algorithm 1 Incremental Learning of a Dynamically Expandable Network

Input: Dataset $\mathcal{D} = (\mathcal{D}_1, \ldots, \mathcal{D}_T)$, Thresholds τ, σ
Output: $\boldsymbol{W}^T$
for $t = 1, \ldots, T$ **do**
  if $t = 1$ **then**
    Train the network weights $\boldsymbol{W}^1$ using Equation 2.2
  else
    $\boldsymbol{W}^t = SelectiveRetraining(\boldsymbol{W}^{t-1})$ ▷ Selectively retrain the previous network using Algorithm 2
    if $\mathcal{L}_t > \tau$ **then**
      $\boldsymbol{W}^t = DynamicExpansion(\boldsymbol{W}^t)$ ▷ Expand the network capacity using Algorithm 3
    end if
    $\boldsymbol{W}^t = Split(\boldsymbol{W}^t)$ ▷ Split and duplicate the units using Algorithm 4
  end if
end for

Selective retraining

A most naive way to train the model for a sequence of tasks would be retraining the entire model every time a new task arrives. However, such retraining will be very costly for a deep neural network. Thus, we suggest performing selective retraining of the model by retraining only the weights affected by the new task. Initially (t=1), we train the network with ℓ_1-regularization to promote sparsity in the weights, such that each neuron is connected to only a few neurons in the layer below.

$$\underset{\boldsymbol{W}^{t=1}}{minimize}\ \mathcal{L}\left(\boldsymbol{W}^{t=1}; \mathcal{D}_t\right) + \mu \sum_{l=1}^{L} \left\|\boldsymbol{W}_l^{t=1}\right\|_1, \tag{2.2}$$

where $1 \leq l \leq L$ denotes the l^{th} layer of the network, $\boldsymbol{W}_l^t$ is the network parameter at layer l, and μ is the regularization parameter of the element-wise ℓ_1 norm for sparsity on $\boldsymbol{W}$. For convolutional layers, we apply $(2,1)$-norm on the filters, to select only a few filters from the previous layer. Throughout our incremental learning procedure, we maintain $\boldsymbol{W}^{t-1}$ to be sparse, thus we can drastically reduce the computation overheads if we can focus on the subnetwork connected new task. To this end, when a new

task t arrives at the model, we first fit a sparse linear model to predict task t using the topmost hidden units of the neural network by solving the following problem:

$$\underset{\boldsymbol{W}_{L,t}^{t}}{minimize}\ \mathcal{L}\left(\boldsymbol{W}_{L,t}^{t}\,;\boldsymbol{W}_{1:L-1}^{t-1},\mathcal{D}_t\right)+\mu\left\|\boldsymbol{W}_{L,t}^{t}\right\|_1, \tag{2.3}$$

where $\boldsymbol{W}_{1:L-1}^{t-1}$ denotes the set of all other parameters except $\boldsymbol{W}_{L,t}^{t}$. That is, we solve this optimization to obtain the connections between output unit o_t and the hidden units at layer $L-1$ (fixing all other parameters up to layer $L-1$ as $\boldsymbol{W}^{t-1}$). Once we build the sparse connection at this layer, we can identify all units and weights in the network that are affected by the training, while leaving the part of the network that is not connected to o_t unchanged. Specifically, we perform a breadth-first search on the network starting from those selected nodes, to identify all units (and input feature) that have paths to o_t. Then, we train only the weights of the selected subnetwork S, denoted as $\boldsymbol{W}_S^t$:

$$\underset{\boldsymbol{W}_{S}^{t}}{minimize}\ \mathcal{L}\left(\boldsymbol{W}_{S}^{t}\,;\boldsymbol{W}_{S^c}^{t-1},\mathcal{D}_t\right)+\mu\left\|\boldsymbol{W}_{S}^{t}\right\|_2. \tag{2.4}$$

We use the element-wise ℓ_2 regularizer since the sparse connections have already been established[1]. This partial retraining will result in lower computational overhead and also help with avoiding negative transfer since neurons that are not selected will not get affected by the retraining process. Algorithm 2 describes the selective retraining process.

Algorithm 2 Selective Retraining

1: **Input:** Dataset $\mathcal{D}_t$, Previous parameter $\boldsymbol{W}^{t-1}$
2: **Output:** network parameter $\boldsymbol{W}^{t}$
3: Initialize $l \leftarrow L-1$, $S=\{o_t\}$
4: Solve Equation 2.3 to obtain $\boldsymbol{W}_{L,t}^{t}$
5: Add neuron i to S if the weight between i and o_t in $\boldsymbol{W}_{L,t}^{t}$ is not zero.
6: **for** $l=L-1,\ldots,1$ **do**
7: Add neuron i to S if there exists some neuron $j\in S$ such that $\boldsymbol{W}_{l,ij}^{t-1}\neq 0$.
8: **end for**
9: Solve Equation 2.4 to obtain $\boldsymbol{W}_S^t$

Dynamic network expansion

In cases where the new task is highly relevant to the old ones, or aggregated partial knowledge obtained from each task is sufficient to explain the new task, selective retraining alone will be sufficient for the new task. However, when the learned features cannot accurately represent the new task, additional neurons need to be introduced to the network, in order to account for the features that are necessary for the new task. Some existing works [188, 130] are based on a similar idea. However, they are either inefficient due to iterative training that requires repeated forward pass [188], or adds in the constant number of units at each task t without consideration of the task difficulty [130] and thus are suboptimal in terms of performance and network capacity utilization.

To overcome these limitations, we instead propose an efficient way of using group sparse regularization to dynamically decide how many neurons to add at which layer, for each task without repeated retraining of the network for each unit. Suppose that we expand the l_{th} layer of a network with a constant number of units, say k, inducing two parameter matrices expansions: $\boldsymbol{W}_l^t=[\boldsymbol{W}_l^{t-1};\boldsymbol{W}_l^{\mathcal{N}}]$ and $\boldsymbol{W}_{l-1}^t=[\boldsymbol{W}_{l-1}^{t-1};\boldsymbol{W}_{l-1}^{\mathcal{N}}]$

[1] We can add in ℓ_1-norm here for further identification of necessary weights

for outgoing and incoming layers respectively, where $\boldsymbol{W}^{\mathcal{N}}$ is the expanded weight matrix involved with added neurons. Since we do not always want to add in all k units (depending on the relatedness between the new task and the old tasks), we perform group sparsity regularization on the added parameters as follows:

$$\underset{\boldsymbol{W}_l^{\mathcal{N}}}{minimize}\ \mathcal{L}\left(\boldsymbol{W}_l^{\mathcal{N}}\ ; \boldsymbol{W}_l^{t-1}, \mathcal{D}_t\right) + \mu \left\|\boldsymbol{W}_l^{\mathcal{N}}\right\|_1 + \gamma \sum_g \left\|\boldsymbol{W}_{l,g}^{\mathcal{N}}\right\|_2, \tag{2.5}$$

where $g \in \mathcal{G}$ is a group defined on the incoming weights for each neuron. For convolutional layers, we defined each group as the activation map for each convolutional filter. This group sparsity regularization was used in [160, 5, 172] to find the right number of neurons for a full network, while we apply it to the partial network. Algorithm 3 describes the details of how expansion works.

After selective retraining is done, the network checks if the loss is below a certain threshold. If not, then at each layer, we expand its capacity by k neurons and solve for Equation 2.5. Due to group sparsity regularization in Equation 2.5, hidden units (or convolutional filters) that are deemed unnecessary from the training will be dropped altogether. We expect that from this dynamic network expansion process, the model captures new features that were not previously represented by $\boldsymbol{W}_l^{t-1}$ to minimize residual errors while maximizing the utilization of the network capacity by avoiding adding in too many units.

Algorithm 3 Dynamic Network Expansion

1: **Input:** Dataset $\mathcal{D}_t$, Threshold τ
2: Perform Algorithm 2 and compute $\mathcal{L}$
3: **if** $\mathcal{L} > \tau$ **then**
4: Add k units $\boldsymbol{h}^{\mathcal{N}}$ at all layers
5: Solve for Equation 2.5 at all layers
6: **end if**
7: **for** $l = L-1, \ldots, 1$ **do**
8: Remove useless units in $\boldsymbol{h}_l^{\mathcal{N}}$
9: **end for**

Network split/duplication

A crucial challenge in lifelong learning is the problem of *semantic drift*, or *catastrophic forgetting*, which describes the problem where the model gradually fits the later learned tasks and thus forgets what it learned for earlier tasks, resulting in degraded performance for them. The most popular yet simple way of preventing semantic drift is to regularize the parameters from deviating too much from their original values using ℓ_2-regularization, as follows:

$$\underset{\boldsymbol{W}^t}{minimize}\ \mathcal{L}(\boldsymbol{W}^t; \mathcal{D}_t) + \lambda \|\boldsymbol{W}^t - \boldsymbol{W}^{t-1}\|_2^2, \tag{2.6}$$

where t is the current task, and $\boldsymbol{W}^{t-1}$ is the weight tensor of the network trained for tasks $\{1, \ldots, t-1\}$, and λ is the regularization parameter. This ℓ_2 regularization will enforce the solution $\boldsymbol{W}^t$ to be found close to $\boldsymbol{W}^{t-1}$, by the degree given by λ; if λ is small, then the network will be learned to reflect the new task more while forgetting about the old tasks, and if λ is high, then $\boldsymbol{W}^t$ will try to preserve the knowledge learned at previous tasks as much as possible. Rather than placing simple ℓ_2 regularization, it is also possible to weight each element with Fisher information [74]. Nonetheless, if the number of tasks is large, or if the later tasks are semantically disparate from the previous tasks, it may become difficult to find a good solution for both previous and new tasks.

A better solution in such a case is to split the neuron so that we have optimal features for two different tasks. After performing Equation 2.6, we measure the semantic drift for each hidden unit i, ρ_i^t, as the ℓ_2-distance between the incoming weights at $t-1$ and t. Then if $\rho_i^t > \sigma$, we consider that the meaning of the feature has significantly changed during training, and split this neuron i into two copies. We perform this operation for all hidden units in parallel. The network then trains the weights again by solving Equation 2.6 since split changes the overall structure. However, in practice, this secondary training usually converges fast due to the reasonable parameter initialization from the initial training. Algorithm 4 describes the algorithm for split operation.

Algorithm 4 Network Split/Duplication

Input: Weight $\boldsymbol{W}^{t-1}$, Threshold σ
Perform Equation 2.6 only using weights activated in $t-1$ stage
for all hidden unit i **do**
  $\rho_i^t = \|\boldsymbol{w}_i^t - \boldsymbol{w}_i^{t-1}\|_2$
  if $\rho_i^t > \sigma$ **then**
    Copy i into i' ($\boldsymbol{w}'$ introduction of edges for i')
  end if
end for

Timestamped inference

In both the network expansion and split procedures, we timestamp each newly added unit j by setting $\{\boldsymbol{z}\}_j = t$ to record the training stage t when it is added to the network to prevent knowledge shift caused by introducing new hidden units. At inference time, each task will only use the parameters introduced up to stage t to prevent the old tasks from using new hidden units added in the training process. This timestamping strategy is more flexible than fixing the weights learned up to each learning stage, as in [130], since early tasks can still benefit from the learning at later tasks via units that are further trained but not split.

2.1.4 Experimental Results

Baselines and our model

DNN-STL: A Base Deep Neural Network (DNN) that trains on each task separately.
DNN-MTL: A Base DNN that trains on all tasks at once.
DNN: A Base DNN. All incremental models use weight decay.
DNN-L2: A Base DNN, where at each task t, $\boldsymbol{W}^t$ is initialized as $\boldsymbol{W}^{t-1}$ and continuously trained with ℓ_2-regularization between $\boldsymbol{W}^t$ and $\boldsymbol{W}^{t-1}$.
DNN-EWC: A DNN trained with Elastic Weight Consolidation (EWC) [74] for regularization.
DNN-Progressive: Our implementation of the Progressive Network [130], whose network weights for each task remain fixed for the later arrived tasks.
DEN: Our dynamically expandable network.

Base network settings

Feedforward networks: We use a two-layer network with 312-128 neurons with ReLU activations.
Convolutional networks: For experiments on the CIFAR-100 dataset, we use a modified version of

AlexNet [78] that has five convolutional layers (64-128-256-256-128 depth with 5×5 filter size), and three fully-connected layers (384-192-100 neurons at each layer).

All models and algorithms are implemented using the Tensorflow [1] library. We will release our codes upon acceptance of our paper for the reproduction of the results.

Datasets

MNIST-Variation: This dataset consists of $62,000$ images of handwritten digits from 0 to 9. Unlike MNIST, the handwritten digits are rotated to arbitrary angles and have noise in the background, which makes the prediction task more challenging. We use $1,000/200/5,000$ images for train/val/test split for each class. We form each task to be an one-versus-rest binary classification.

CIFAR-100: This dataset consists of $60,000$ images of 100 generic object classes[77]. Each class has 500 images for training and 100 images for test. We used a CNN as the base network for the experiments on this dataset, to show that our method is applicable to a CNN. Further, we considered each task as a set of 10 subtasks, each of which is a binary classification task on each class.

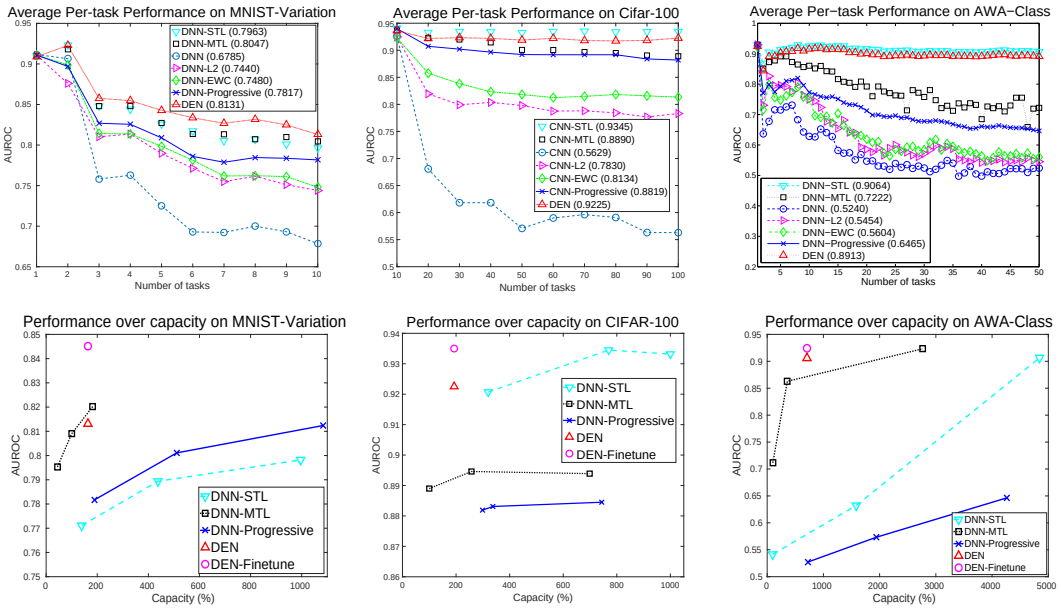
AWA (Animals with Attributes): This dataset consists of $30,475$ images of 50 animals [82]. For features, we use DECAF features provided with the dataset, whose dimensionality is reduced to 500 by PCA. We use random splits of 30/30/30 images for training/validation/test.

Quantitative evaluation

We validate our models for prediction accuracy and efficiency, where we measure the efficiency by network size at the end of training and training time. We first report the average per-task performance of baselines and our models in the top row of Figure 2.3. DNN-STL showed the best performances over AWA and CIFAR-100 datasets as it trained each task separately, finding individual optimal solutions without forgetting. On the other hand, all other models train on given tasks sequentially, which might cause semantic drift. When the number of tasks is small, MTL works best from knowledge sharing via multi-task learning. But when the number of tasks is large, STL works better since it has a larger learning capacity than MTL.

Our proposed DEN performs almost the same as these batch models and even outperforms them on the MNIST-Variation dataset. Retraining models combined with regularization, such as L2 and EWC, do not perform well because these models suffer from limited expressiveness during lifelong learning. Progressive network works better than the two, but it underperforms DEN on all datasets. The performance gap is most significant on AWA, as a larger number of tasks ($T = 50$) may have made it more challenging to find the appropriate network capacity. If the network is too small, then it will not have sufficient learning capacity to represent new tasks, and if too large, it will become prone to overfitting.

We further report the performance of each model over network capacity measured relative to MTL on each dataset in Figure 2.3 (bottom row). For baselines, we report the performance of multiple models with different network capacities. DEN obtains much better performance with a substantially fewer number of parameters than Progressive network or obtains significantly better performance using a similar parameter size. DEN also obtains the same level of performance as STL using only 18.0%, 60.3%, and 11.9% of its capacity on MNIST-Variation, CIFAR-100, and AWA, respectively. The result also shows the main advantage of DEN, which is being able to dynamically find its optimal capacity since it learns a very compact model on MNIST-Variation, whilst learning a substantially large network on CIFAR-100. Further fine-tuning of DEN on all tasks (DEN-Finetune) obtains the best-performing model



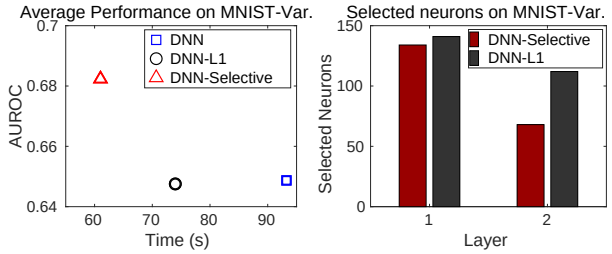
(a) MNIST-Variation (T=10) (b) CIFAR-100 (T=10) (c) AWA (T=50)

Figure 2.3: **Top row:**Average per-task performance of the models over number of task t, averaged over five random splits. The numbers in the legend denote average per-task performance after the model has finished learning ($t = T$). **Bottom row:** Accuracy over network capacity. The network capacity is given relative to the capacity of MTL, which we consider as 100%.

on all datasets, which shows that DEN is useful for lifelong learning and can be used for network capacity estimation when all tasks are available in the first place.

Effect of selective retraining We further examine how efficient and effective selective training is by measuring the training speed and the area under the ROC curve on MNIST-Variation dataset. We compare the model without network expansion, which we refer to as DNN-Selective, against retraining on DNN-L2 and DNN-L1 (Equation 2.2). Section 2.1.4 (a) shows the accuracy over training time measured as actual time spent with GPU computation for each model. We observe that selective retraining takes significantly less training time than the full retraining of the network and even less than DNN-L1, which comes with sparse network weights. Further, whereas DNN-L1 obtained slightly less accuracy than DNN-L2, DNN-Selective improved the accuracy over the base network by 2%p. This accuracy gain may be due to the suppression of catastrophic forgetting, as DEN trains only a partial subnetwork for each newly introduced task. Section 2.1.4 (b) shows the number of selected neurons at each layer with selective retraining. Note that DNN-selective mostly selects less portion of upper-level units, which are more task-specific, while selecting a larger portion of more generic lower-layer units.

Effect of network expansion We also compare the effectiveness of the network expansion against the model with a variant of our model that performs selective retraining and layer expansion without network split. We refer to this model as DNN-Dynamic. We compare DNN-Dynamic with DNN-L2 used in the main experiment and DNN-Constant, a version of our model that expands its capacity at each layer with a fixed number of units, on MNIST-Variation dataset. Section 2.1.4 (c) shows that DNN-Dynamic significantly outperforms all baselines in terms of the mean AU-ROC, including DNN-Constant, while



(a) AUROC over Time (b) Selected neurons

Models	AUROC	Capacity
DNN-L2	0.7440±0.01	1
DNN-L2-Selective	0.7499±0.01	1
DEN-Constant (k=13)	0.7611±0.01	1.55
DEN-Constant (k=20)	0.7580±0.01	1.89
DEN-Dynamic	**0.7684±0.01**	1.56±0.01

(c) Expansion performance

Figure 2.4: **Effect of selective retraining.** (a) shows AUROC over actual training time and (b) shows the number of selected neurons by selective retraining. **(c) Expansion performance.** We report both the prediction AUROC and network capacity measured by the relative number of parameters to that of DNN-MTL on MNIST-Variance dataset. Reported numbers are mean and standard error for five random splits.

(a) Task1 (b) Task4 (c) Task7

Figure 2.5: **Semantic drift experiment on the MNIST-Variation dataset.** We report the AUROC of different models on $t = 1$, $t = 4$, and $t = 7$ at each training stage to see how the model performance changes over time for these tasks. Reported AUROC is the average over five random splits.

increasing the size of the network substantially less than DNN-Constant (k=20). That is, having less number of parameters is beneficial for training efficiency and advantageous in preventing the model from overfitting. We can set the network capacity of DNN-Constant to be similar (k=13) to obtain better accuracy, but it still underperforms DEN, which can dynamically adjust the number of neurons at each layer.

Effect of network split/duplication and timestamped inference To see how network *split and duplication* and *unit timestamping* help prevent semantic drift (or catastrophic forgetting) while allowing to obtain good performances on later tasks, we compare the performance of our model against baselines and also a variant of our DEN without timestamped inference (DEN-No-Stamp) at different learning stages. Each figure in Figure 2.5 (a), (b), and (c) shows how the performance of the model changes at each training stage t, for tasks $t = 1$, $t = 4$, and $t = 7$, respectively.

We observe that DNN-L2 prevents semantic drift of the models learned at early stages but results in increasingly worse performance on later tasks ($t = 4, 7$). DNN-EWC, on the other hand, performs better on later tasks than DNN-L2, as reported in [74]. However, it shows significantly lower performance than both DNN-Progressive and our model, which may be due to its inability to increase network capacity, resulting in limited expressive power. DNN-Progressive shows no semantic drift on old tasks because it does not retrain parameters for them. DEN w/o Timestamping works better than DNN-Progressive on later tasks,

with slight performance degeneration over time. Finally, our full model with timestamped inference, DEN, shows no sign of noticeable performance degeneration at any learning stage while significantly outperforming DNN-Progressive, showing that DEN is highly effective in preventing semantic drift.

2.1.5 Summary

We proposed a novel deep neural network for lifelong learning, Dynamically Expandable Network (DEN). DEN performs partial retraining of the network trained on old tasks by exploiting task-relatedness. Also, DEN dynamically increases its capacity when necessary to account for new knowledge for incoming tasks during lifelong learning. To this end, we find its optimal capacity for solving the problem and effectively preventing semantic drift. We implement both feedforward and convolutional neural network version of our DEN and validate them on multiple classification datasets under lifelong learning scenarios, on which they significantly outperform the existing lifelong learning methods, achieving almost the same performance as the network trained in batch while using as little as $11.9\%p - 60.3\%p$ of its capacity. Further, the fine-tuning of the models on all tasks results in obtaining models that outperform the batch models, which shows that DEN is useful for network structure estimation as well.

2.2 Scalable and Order-robust Continual Learning with Additive Parameter Decomposition [175]

2.2.1 Motivation

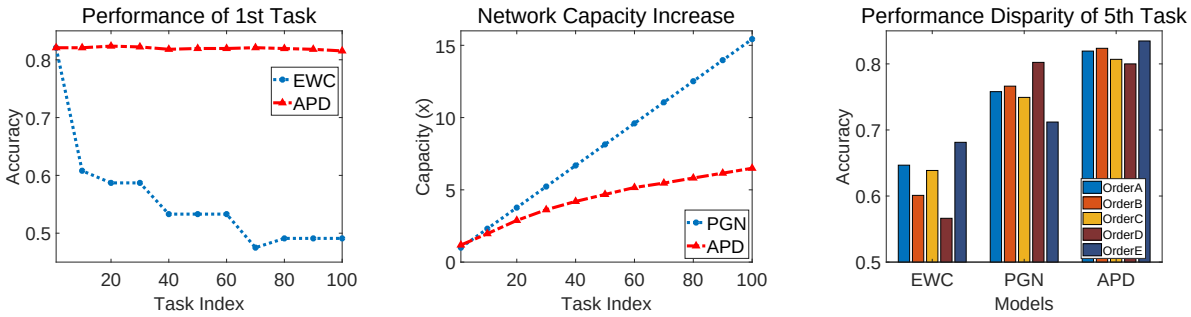
Continual learning [152], or lifelong learning, is a learning scenario where a model is incrementally updated over a sequence of tasks, potentially performing knowledge transfer from earlier tasks to later ones. Building a successful continual learning model may lead us one step further towards developing general artificial intelligence since learning numerous tasks over a long-term time period is an essential aspect of human intelligence. Continual learning is often formulated as incremental/online multi-task learning that models complex task-to-task relationships by sharing basis vectors in linear models [80, 131] or weights in neural networks [97]. One problem that arises here is that as the model learns the new tasks, it could forget what it learned for the earlier tasks, which is known as the problem of *catastrophic forgetting*. Many recent works in continual learning of deep networks [97, 90, 141, 74, 127, 19] tackle this problem by introducing advanced regularizations to prevent the drastic change of network weights. Yet, when the model should adapt to multiple tasks, the interference between task-specific knowledge is inevitable with fixed network capacity. Recently introduced expansion-based approaches handle this problem by expanding the network capacity as they adapt to new tasks [130, 37, 178, 95]. These recent advances have largely alleviated the catastrophic forgetting, at least with a small number of tasks.

However, to deploy continual learning to real-world systems, there are a number of issues that should be resolved. First, in practical scenarios, the number of tasks that the model should train on may be large. In the lifelong learning setting, the model may even have to continuously train on an unlimited number of tasks. Yet, conventional continual learning methods have not been verified for their scalability to a large number of tasks, both in terms of effectiveness in the prevention of catastrophic forgetting, and efficiency as to memory usage and computations (See Figure 2.6 (a) and (b)).

Another critical but relatively less explored problem is the problem of *task order sensitivity*, which describes the performance discrepancy with respect to the task arrival sequence (See Figure 2.6 (c)). The task order that the model trains on significantly impacts the individual task performance as well as the final performance, not only because of the model drift coming from catastrophic forgetting but due to the unidirectional knowledge transfer from earlier tasks to later ones. This order sensitivity could be highly problematic if fairness across tasks is important

To handle these practical challenges, we propose a novel continual learning model with emphAdditive Parameter Decomposition (APD). APD decomposes the network parameters at each layer of the target network into task-shared and sparse task-specific parameters with small mask vectors. At each arrival of a task to a network with APD, which we refer to as *APD-Net*, it will try to maximally utilize the task-shared parameters and learn the incremental difference that cannot be explained by the shared parameters using sparse task-adaptive parameters. Moreover, since having a single set of shared parameters may not effectively utilize the varying degree of knowledge-sharing structure among the tasks, we further cluster the task-adaptive parameters to obtain hierarchically shared parameters (See Figure 2.7).

This decomposition of generic and task-specific knowledge has clear advantages in tackling the previously mentioned problems. APD will greatly alleviate catastrophic forgetting since learning on later tasks will not affect the task-adaptive parameters for the previous tasks and will update the task-shared parameters only with generic knowledge. Since APD does not change the network topology as existing expansion-based approaches do, APD-Net is memory-efficient, even more so with hierarchically shared



(a) Catastrophic forgetting (b) Scalability (c) Task-order sensitivity

Figure 2.6: Description of crucial challenges for continual learning with Omniglot dataset experiment. **Catastrophic forgetting:** Model should not forget what it has learned about previous tasks. **Scalability:** The increase in network capacity with respect to the number of tasks should be minimized. **Order sensitivity:** The model should have a similar final performance regardless of the task order. Our model with Additive Parameter Decomposition effectively solves these three problems.

parameters. It also trains fast since it does not require multiple rounds of retraining. Moreover, it is order-robust since the task-shared parameters can stay relatively static and will converge to a solution rather than drift away upon the arrival of each task. The additional mechanism to retroactively update task-adaptive parameters can further alleviate the order sensitivity from unidirectional knowledge transfer.

We validate our methods on several benchmark datasets for continual learning while comparing them against state-of-the-art continual learning methods to obtain significantly superior performance with minimal increase in network capacity while being scalable and order-robust.

The contribution of this paper is threefold:

- We tackle practically important and novel problems in continual learning that have been overlooked thus far, such as *scalability* and *order robustness*.
- We introduce a novel framework for continual deep learning that effectively prevents catastrophic forgetting. The proposed method is highly scalable and order-robust by decomposing the network parameters into shared and sparse task-adaptive parameters with small mask vectors.
- We perform extensive experimental validation of our model on multiple datasets against recent continual learning methods, whose results show that our method is significantly superior to them in terms of accuracy, efficiency, scalability, as well as order-robustness.

2.2.2 Related Work

Continual learning The literature on continual (lifelong) learning [152, 131] is vast as it is a long-studied topic, but we only mention the recent and relevant works. Most continual deep learning approaches have focused on preventing catastrophic forgetting, in which case the retraining of the network for new tasks shifts the distribution of the learned representations. Most continual deep learning approaches are focused on preventing *catastrophic forgetting*, in which case the retraining of the network for new tasks shifts the distribution of the learned representations. A simple yet effective regularization is to enforce the representations learned at the current task to be closer to ones from the network trained on

previous tasks [97]. EWC [74] and P&C [135] propose to regularize the model parameter for the current tasks with parameters for the previous task via a Fisher information matrix to find a solution that works well for both tasks. *Lee et al.* [90] introduce a moment-matching technique with a similar objective. HAT [137] presents a new binary masking approach to minimize drift for important prior knowledge. The model learns a pseudo-step function to promote hard attention, then builds a compact network with marginal forgetting. But the model cannot expand the network capacity and performs unidirectional knowledge transfer, thus suffering from the order-sensitivity. DGR [141] employs deep generative models to compactly encode task knowledge and later generate samples from the model as it learns new tasks. GEM [102] and A-GEM [19] introduce new directions for efficient continual learning with weighted updates according to the gradients of episodic memory under single-epoch learning scenarios. VCL [113] formulates continual learning as a sequential Bayesian update. The method utilizes the replay buffer to mitigate forgetting when estimating the posterior distribution over weights for the new task. *Riemer et al.* [127] addresses the stability-plasticity dilemma maximizing knowledge transfer to later tasks while minimizing their interference on earlier tasks, using optimization-based meta-learning with experience replay.

Dynamic network expansion Continual learners of fixed size are infeasible to immune catastrophic forgetting even with well-defined regularizers as they may encounter an unlimited number of tasks. An effective way to tackle this challenge is by dynamically expanding the network capacity to handle new tasks. *Zhou et al.* [188] present an iterative algorithm to train a denoising autoencoder while adding in new neurons one by one and merging similar units. Progressive [130] Network keeps the old weights fixed to avoid catastrophic forgetting and expands the network by augmenting each layer with a fixed number of additional neurons per task. Yet, this framework often results in a network with excessive size. DEN [178] proposes to overcome this limitation via selective retraining of the old network while expanding each layer with only the necessary number of neurons via neuronwise group sparsification. With a similar objective, RCL [170] uses reinforcement learning to decide how many neurons to add. *Li et al.* [95] propose to perform an explicit network architecture search to decide how much to reuse the existing network weights and how much to add. Our model also performs dynamic network expansion as the previous expansion-based methods, but instead of adding in new units, it additively decomposes the network parameters into task-shared and task-specific parameters. Further, we minimize the increase of model size at the arrival of each task by constraining the sparsity on the task-specific parameters and adopting the hierarchical structure of shared parameters.

2.2.3 Approach

Assuming that a deep network receives a sequence of tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_T\}$ in random order in the continual learning setting. We denote the dataset of the t^{th} task as $\mathcal{D}_t = \{\boldsymbol{x}_t^i, \boldsymbol{y}_t^i\}_{i=1}^{N_t}$, where $\boldsymbol{x}_t^i$ and $\boldsymbol{y}_t^i$ are i^{th} instance and label among N_t examples. We further assume that they become inaccessible after step t. The set of parameters for the network at step t is given as $\boldsymbol{\Theta}_t = \{\boldsymbol{\theta}_t^l\}_{l=1}^{L}$, where $\boldsymbol{\theta}_t^l$ represents the weights for layer l; we omit the layer index l when the context is clear. Then the training objective at the arrival of task t can be defined as follows: $\underset{\boldsymbol{\Theta}_t}{minimize} \mathcal{L}\left(\boldsymbol{\Theta}_t; \boldsymbol{\Theta}_{t-1},\ \mathcal{D}_t\right) + \lambda\mathcal{R}(\boldsymbol{\Theta}_t)$, where $\mathcal{R}(\cdot)$ is a regularization term on the model parameters. In the next paragraph, we introduce our continual learning framework with task-adaptive parameter decomposition and hierarchical knowledge consolidation.

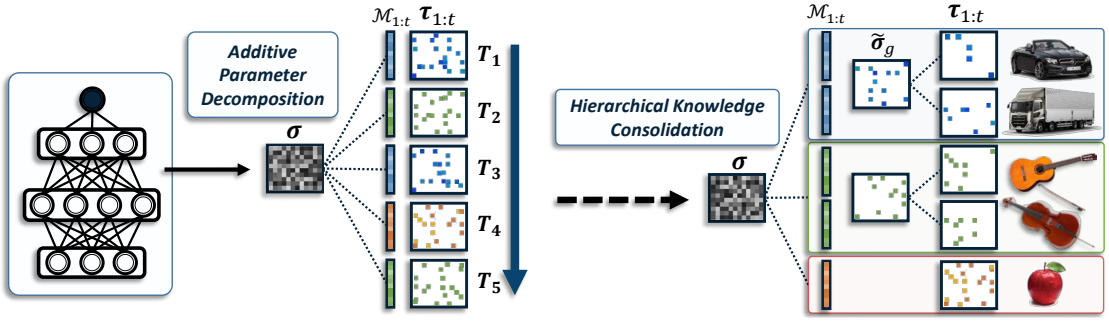


Figure 2.7: **An illustration of Additive Parameter Decomposition (APD) for continual learning.** APD effectively prevents catastrophic forgetting and suppresses order-sensitivity by decomposing the model parameters into shared $\boldsymbol{\sigma}$ and sparse task-adaptive $\boldsymbol{\tau}_t$, which will let later tasks only update shared knowledge. $\mathcal{M}_t$ is the task-adaptive mask on $\boldsymbol{\sigma}$ to transform generic knowledge to adapt the corresponding task. APD enhances scalability through weight sparsification on $\boldsymbol{\tau}_t$ and hierarchical knowledge consolidation.

Additive Parameter Decomposition

To minimize the impact of catastrophic forgetting and the amount of newly introduced parameters by network expansion, we propose to decompose $\boldsymbol{\theta}$ into a *task-shared* parameter matrix $\boldsymbol{\sigma}$ and a *task-adaptive* parameter matrix $\boldsymbol{\tau}$, that is, $\boldsymbol{\theta}_t = \boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t$ for task t. The masking variable $\mathcal{M}_t$ acts as an *attention* on the task-shared parameter to guide the learner to focus only on the parts relevant to each task. This decomposition allows us to easily control the trade-off between semantic drift and predictive performance of a new task by imposing separate regularizations on decomposed parameters. When a new task arrives, we encourage the shared parameters $\boldsymbol{\sigma}$ to be properly updated, but not deviate far from the previous shared parameters $\boldsymbol{\sigma}^{(t-1)}$. At the same time, we enforce the capacity of $\boldsymbol{\tau}_t$ to be as small as possible by making it sparse. The objective function for this decomposed parameter model is given as follows:

$$\underset{\boldsymbol{\sigma}, \boldsymbol{\tau}_t, \boldsymbol{v}_t}{minimize} \quad \mathcal{L}\left(\{\boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t\}; \mathcal{D}_t\right) + \lambda_1 \left\|\boldsymbol{\tau}_t\right\|_1 + \lambda_2 \left\|\boldsymbol{\sigma} - \boldsymbol{\sigma}^{(t-1)}\right\|_2^2, \tag{2.7}$$

where $\mathcal{L}$ denotes a loss function, $\boldsymbol{\sigma}^{(t-1)}$ denotes the shared parameter before the arrival of the current task t, $\|\cdot\|_1$ indicates an element-wise ℓ_1 norm defined on the matrix, and λ_1, λ_2 are hyperparameters balancing efficiency catastrophic forgetting. We use ℓ_2 transfer regularization to prevent catastrophic forgetting, but we could use other types of regularizations as well, such as Elastic Weight Consolidation [74]. The masking variable $\mathcal{M}_t$ is a sigmoid function with a learnable parameter $\boldsymbol{v}_t$, which is applied to output channels or neurons of $\boldsymbol{\sigma}$ in each layer. We name our model with decomposed network parameters, *Additive Parameter Decomposition (APD)*.

The proposed decomposition in Equation 2.7 makes continual learning efficient since, at each task, we only need to learn an extremely sparse $\boldsymbol{\tau}_t$ that accounts for task-specific knowledge that cannot be explained with the transformed shared knowledge $\boldsymbol{\sigma} \otimes \mathcal{M}_t$. Thus, we are doing residual learning with $\boldsymbol{\tau}_t$. Further, it helps the model achieve robustness to the task arrival order because semantic drift occurs only through the task-shared parameter corresponding to *generic* knowledge, while the *task-specific* knowledge learned from previous tasks is kept intact. Next, we introduce additional techniques to achieve even more task-order robustness and efficiency.

Algorithm 5 Continual learning with Additive Parameter Decomposition

input Dataset $\mathcal{D}_{1:T}$ and hyperparameter $\lambda, m, s, K = k$

output $\boldsymbol{\sigma}^{(T)}$, $\boldsymbol{v}_{1:T}$, $\widetilde{\boldsymbol{\sigma}}_{1:K}$, and $\boldsymbol{\tau}_{1:T}$

Let $\boldsymbol{\sigma}^{(1)} = \boldsymbol{\theta}_1$, and optimize for the task 1
for $t = 2, ..., T$ **do**
  for $i = 1, ..., t-1$ **do**
    Restore $\boldsymbol{\theta}_i^* = \boldsymbol{\sigma}^{(t-1)} \otimes \mathcal{M}_i^{(t-1)} + \widetilde{\boldsymbol{\tau}}_i^{(t-1)}$
  end for
  Minimize Equation 2.9 to update $\boldsymbol{\sigma}$ and $\{\boldsymbol{\tau}_i, \boldsymbol{v}_i\}_{i=1}^t$
  if $t \mod s = 0$ **then**
    Initialize k new random centroids, $\{\boldsymbol{\mu}_g\}_{g=K-k+1}^K$
    Group all tasks into K disjoint sets, $\{\mathcal{G}_g\}_{g=1}^K$
    for $g = 1, ..., K$ **do**
      Decompose $\{\widetilde{\boldsymbol{\tau}}_i\}_{i \in \mathcal{G}_g}$ into $\widetilde{\boldsymbol{\sigma}}_g$ and $\{\boldsymbol{\tau}_i\}_{i \in \mathcal{G}_g}$
    end for
    Delete old $\widetilde{\boldsymbol{\sigma}}$ and $K = K + k$
  end if
end for

Order robust continual learning with retroactive parameter updates

We observe that a naive update of the shared parameters may induce semantic drift in parameters for the previously trained tasks, yielding an order-sensitive model since we do not have access to previous task data. In order to provide a high degree of order-robustness, we impose an additional regularization to further prevent *parameter-level* drift without explicitly training on the previous tasks.

To achieve order-robustness in Equation 2.7, we need to *retroactively* update task-adaptive parameters of the past tasks to reflect the updates in the shared parameters at each training step so that all previous tasks are able to maintain their original solutions. Toward this objective, when a new task t arrives, we first recover all previous parameters ($\boldsymbol{\theta}_i$ for task $i < t$): $\boldsymbol{\theta}_i^* = \boldsymbol{\sigma}^{(t-1)} \otimes \mathcal{M}_i^{(t-1)} + \boldsymbol{\tau}_i^{(t-1)}$ and then update $\boldsymbol{\tau}_{1:t-1}$ by constraining the combined parameter $\boldsymbol{\sigma} \otimes \mathcal{M}_i + \boldsymbol{\tau}_i$ to be close to $\boldsymbol{\theta}_i^*$. The learning objective for the current task t is then described as follows:

$$\underset{\boldsymbol{\sigma}, \boldsymbol{\tau}_{1:t}, \boldsymbol{v}_{1:t}}{minimize}\ \mathcal{L}\left(\{\boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t\}; \mathcal{D}_t\right) + \lambda_1 \sum_{i=1}^{t} \|\boldsymbol{\tau}_i\|_1 + \lambda_2 \sum_{i=1}^{t-1} \|\boldsymbol{\theta}_i^* - (\boldsymbol{\sigma} \otimes \mathcal{M}_i + \boldsymbol{\tau}_i)\|_2^2 . \tag{2.8}$$

Compared to Equation 2.7, the task-adaptive parameters of previous tasks can now be retroactively updated to minimize the parameter-level drift. This formulation also constrains the update of the task-shared parameters to consider order-robustness.

Hierarchical knowledge consolidation

The objective function in Equation 2.8 does not directly consider local sharing among the tasks, and thus it will inevitably result in the redundancy of information in the task-adaptive parameters. To further minimize the capacity increase, we perform *hierarchical knowledge consolidation* to group relevant task-adaptive parameters into task-shared parameters (See Figure 2.7). We first cluster all tasks into K disjoint sets $\{\mathcal{G}_g\}_{g=1}^K$ using K-means clustering on $\{\boldsymbol{\tau}_i\}_{i=1}^t$, then decompose the task-adaptive parameters in the same group into locally-shared $\widetilde{\boldsymbol{\sigma}}_g$ and task-adaptive parameters $\{\boldsymbol{\tau}_i\}_{i \in \mathcal{G}_g}$ (with higher sparsity) by simply computing the amount of value discrepancy in each parameter as follows:

- If $\max\{\boldsymbol{\tau}_{i,j}\}_{i\in\mathcal{G}_g} - \min\{\boldsymbol{\tau}_{i,j}\}_{i\in\mathcal{G}_g} \leq \beta$, then $\{\boldsymbol{\tau}_{i,j}\}_{i\in\mathcal{G}_g} = 0$ and $\widetilde{\boldsymbol{\sigma}}_{g,j} = \boldsymbol{\mu}_{g,j}$
- Else, $\widetilde{\boldsymbol{\sigma}}_{g,j} = 0$,

where $\boldsymbol{\tau}_{i,j}$ denotes the jth element of the ith task-adaptive parameter matrix, and $\boldsymbol{\mu}_g$ is the cluster center of group $\mathcal{G}_g$. We update the locally-shared parameters $\widetilde{\boldsymbol{\sigma}}_g$ after the arrival of every s tasks for efficiency, by performing K-means clustering while initializing the cluster centers with the previous locally-shared parameters $\widetilde{\boldsymbol{\sigma}}_g$ for each group. At the same time, we increase the number of centroids to $K+k$ to account for the increase in the variance among the tasks.

Our final objective function is then given as follows:

$$\underset{\boldsymbol{\sigma},\boldsymbol{\tau}_{1:t},\boldsymbol{v}_{1:t}}{minimize}\ \mathcal{L}\left(\{\boldsymbol{\sigma}\otimes\mathcal{M}_t+\boldsymbol{\tau}_t\};\mathcal{D}_t\right)+\lambda_1\sum_{i=1}^{t}\|\boldsymbol{\tau}_i\|_1+\lambda_2\sum_{i=1}^{t-1}\|\boldsymbol{\theta}_i^*-(\boldsymbol{\sigma}\otimes\mathcal{M}_i+\widetilde{\boldsymbol{\tau}}_i)\|_2^2, \quad \text{where } \widetilde{\boldsymbol{\tau}}_i=\boldsymbol{\tau}_i+\widetilde{\boldsymbol{\sigma}}_g \text{ for } i\in\mathcal{G}_g. \tag{2.9}$$

Algorithm 5 describes the training of our APD model.

Selective task forgetting

In real-world scenarios, some tasks learned at earlier stages may become irrelevant as we continue to train the model. For example, you don't need to recognize discontinued products when training a product identification model. In such situations, we may want to forget the earlier tasks in order to secure network capacity for later task learning. Unfortunately, existing continual learning methods cannot effectively handle this problem since removing some features or parameters will also negatively affect the remaining tasks as their parameters are entangled. Yet, with APDs, forgetting a task t can be done by dropping out the task-adaptive parameters $\boldsymbol{\tau}_t$. Trivially, this will have absolutely no effect on the task-adaptive parameters of the remaining tasks.

2.2.4 Experimental Results

We now validate APD-Net on multiple datasets against state-of-the-art continual learning methods. We use a modified version of LeNet-5 [85] and VGG16 network [143] with batch normalization as base networks.

Baselines and Our Models

L2-Transfer: Deep neural networks trained with the $L2$-transfer regularizer $\lambda\|\boldsymbol{\theta}_t-\boldsymbol{\theta}_{t-1}\|_F^2$ when training for task t.
EWC: Deep neural networks regularized with Elastic Weight Consolidation [74].
P&C: Deep neural networks with two-step training: Progress, and Compresss [135].
PGN: Progressive Neural Networks [130] which constantly increase the network size by k neurons with each task.
DEN: Dynamically Expandable Networks [178] that selectively retrain and dynamically expand the network size by introducing new units and duplicating neurons with semantic drift.
RCL: Reinforced Continual Learning proposed in [170] which adaptively expands units at each layer using reinforcement learning.
APD-Fixed: APD-Net without the retroactive update of the previous task-adaptive parameters (Eq. equation 2.7).

APD(1): Additive Parameter Decomposition Networks with depth 1, whose parameter is decomposed into task-shared and task-adaptive parameters.
APD(2): APD-Net with depth 2, which also has locally shared parameters from hierarchical knowledge consolidation.

Datasets

CIFAR-100 Split [77] consists of images from 100 generic object classes. We split the classes into 10 groups and consider 10-way multi-class classification in each group as a single task. We use 5 random training/validation/test splits of $4,000/1,000/1,000$ samples.
CIFAR-100 Superclass consists of images from 20 superclasses of the CIFAR-100 dataset, where each superclass consists of 5 different but semantically related classes. For each task, we use 5 random training/validation/test splits of $2,000/500/500$ samples.
3) Omniglot-rotation [81] contains OCR images of $1,200$ characters (we only use the training set) from various writing systems for training, where each class has 80 images, including 0, 90, 180, and 270 degree rotations of the original images. We use this dataset for large-scale continual learning experiments by considering the classification of 12 classes as a single task, obtaining 100 tasks in total. For each class, we use 5 random training/test splits of 60/20 samples.

Quantitative Evaluation

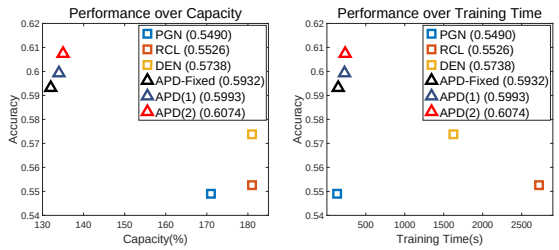
Task-average performance We first validate the final task-average performance after the completion of continual learning. To perform a fair evaluation of performance that is not order-dependent, we report the performance on three random trials over five different task sequences over all experiments. Table 2.1 shows that APD-Nets outperform all baselines by large margins in accuracy. We attribute this performance gain to two features. First, an APD-Net uses neuron(filter)-wise masking on the shared parameters, allowing it to focus only on parts relevant to the task at the current training stage. Secondly, APD-Net updates the previous task-adaptive parameters to reflect the changes made to the shared parameters to perform retroactive knowledge transfer. APD-Fixed, without these retroactive updates, performs slightly worse. APD(2) outperforms APD(1) since it further allows local knowledge transfer with hierarchically shared parameters. Moreover, our methods yield considerably higher accuracy with lower capacity than expansion-based baselines (Figure 2.8). This efficiency comes from task-adaptive learning performing only residual learning for each task with a minimal capacity increase while maximally utilizing the task-shared parameters.

We further validate the efficiency of our methods in terms of training time. Existing approaches with network expansion require excessive training time. DEN performs its training process with multiple sequential steps, namely selective retraining, dynamic network expansion, and split/duplication, each of which requires retraining of the network. RCL uses reinforcement learning techniques, which is inherently slow since the agent should determine how many neurons to add at each layer in a discrete space. PGN trains much faster, but the model increases the fixed number of neurons at each layer when a new task arrives, resulting in overly large networks. On the contrary, although APD-Net requires updates to the previous task-adaptive parameters, it is achieved in a single training step. Figure 2.8 shows that APD(1) and APD(2) perform training within competitive training time to the base model.

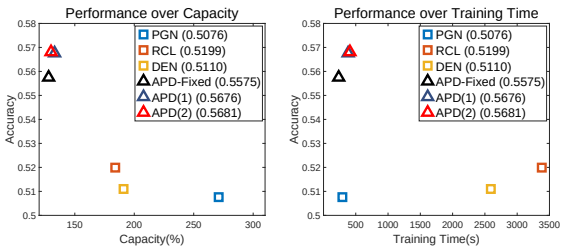
Order fairness in continual learning We now evaluate the order-robustness of our model in comparison to the existing approaches. We first define an evaluation metric for order-sensitivity for each

Table 2.1: **Experiment results on CIFAR-100 Split and CIFAR-100 Superclass datasets.** The results are the mean accuracies over 3 runs of experiments with random splits, performed with 5 different task order sequences. STL is the single-task learning model that trains a separate network for each task independently.

	CIFAR-100 Split				CIFAR-100 Superclass			
Methods	**Capacity**	**Accuracy**	**AOPD**	**MOPD**	**Capacity**	**Accuracy**	**AOPD**	**MOPD**
STL	1,000%	63.75%	0.98%	2.23%	2,000%	61.00%	2.31%	3.33%
L2T	100%	48.73%	8.62%	17.77%	100%	41.40%	8.59%	20.08%
EWC [74]	100%	53.72%	7.06%	15.37%	100%	47.78%	9.83%	16.87%
P&C [135]	100%	53.54%	6.59%	11.80%	100%	48.42%	9.05%	20.93%
PGN [130]	171%	54.90%	8.08%	14.63%	271%	50.76%	8.69%	16.80%
DEN [178]	181%	57.38%	8.33%	13.67%	191%	51.10%	5.35%	10.33%
RCL [170]	181%	55.26%	5.90%	11.50%	184%	51.99%	4.98%	14.13%
APD-Fixed	132%	59.32%	2.43%	4.03%	128%	55.75%	3.16%	6.80%
	175%	61.02%	2.26%	**2.87%**	191%	57.98%	**2.58%**	**4.53%**
APD(1)	134%	59.93%	2.12%	3.43%	133%	56.76%	3.02%	6.20%
	170%	**61.30%**	**1.57%**	**2.77%**	191%	**58.37%**	**2.64%**	**5.47%**
APD(2)	135%	60.74%	**1.79%**	3.43%	130%	56.81%	2.85%	5.73%
	153%	**61.18%**	1.86%	3.13%	182%	**58.53%**	2.75%	5.67%



(a) CIFAR-100 Split (T=10)



(b) CIFAR-100 Superclass (T=20)

Figure 2.8: Accuracy over the efficiency of expansion-based continual learning methods and our methods. We report performance over capacity and performance over training time on both datasets.

task t, which we name as *Order-normalized Performance Disparity (OPD)*, as the disparity between its performance on R random task sequences:

$$OPD_t = \max(\overline{P}_t^1, ..., \overline{P}_t^R) - \min(\overline{P}_t^1, ..., \overline{P}_t^R) \tag{2.10}$$

where $\overline{P}_t^r$ denotes the performance of task t to the task sequence r. Then we define the *Maximum OPD* as $MOPD = \max(OPD_1, ..., OPD_t)$, and the *Average OPD* as $AOPD = \frac{1}{T}\sum_{t=1}^{T} OPD_t$, to evaluate order-robustness on the entire task set. A model that is sensitive to the task sequence order will have high MOPD and AOPD, and an order-robust model will have low values for both metrics.

In Table 2.1, we show the experimental results on order-robustness for all models, obtained on five random sequences. We observe that expansion-based continual learning methods are more order-robust than fixed-capacity methods, owing to their ability to introduce task-specific units. However, they still suffer from a large degree of performance disparity due to the asymmetric direction of knowledge transfer from earlier tasks to later ones. On the other hand, APD-Nets obtain significantly lower MOPD and AOPD than baseline models, which have a high disparity in performance between task sequences given in

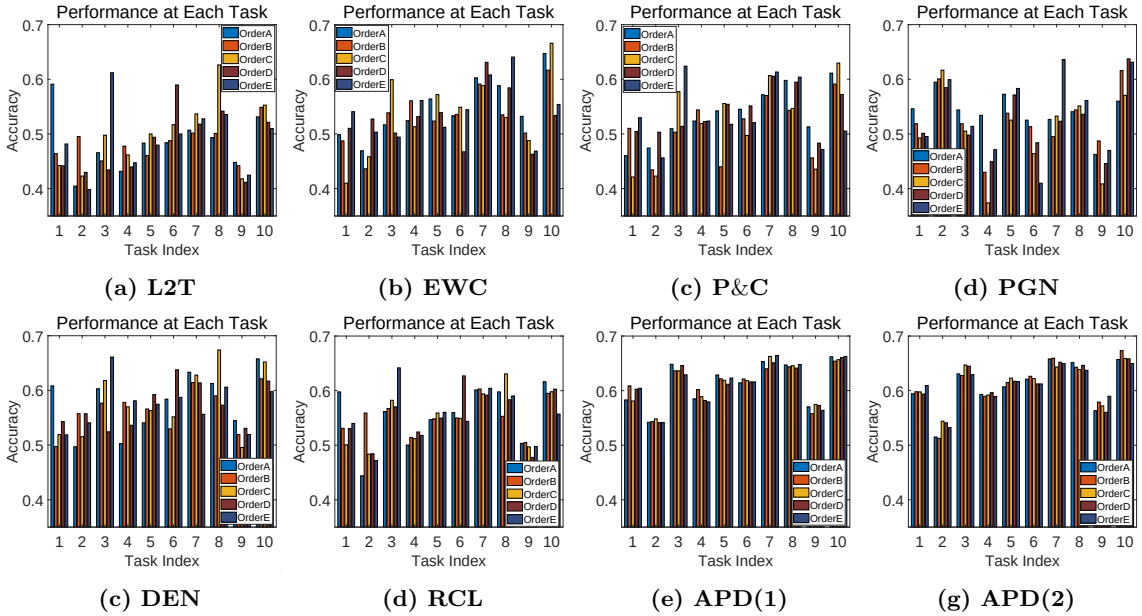


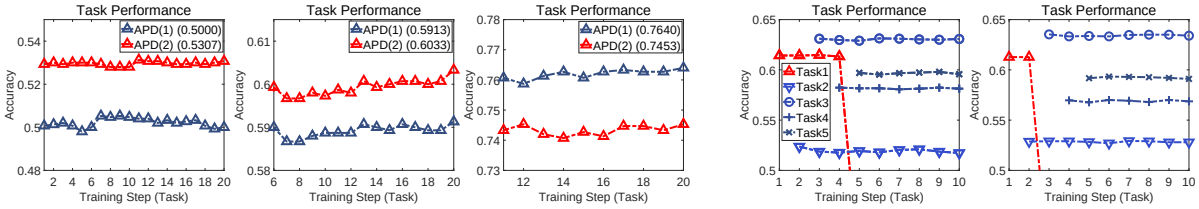
Figure 2.9: Per-task accuracy for each task sequence of continual learning baselines and our models on CIFAR-100 Split, on 5 task sequences of different order. Large amount of disparity among task performance of different orders implies that the model is task-order sensitive, that is less confident in terms of fairness in continual learning.

different orders. APD(1) and APD(2) are more order-robust than APD-Fixed, suggesting the effectiveness of the retroactive updates of $\boldsymbol{\tau}_{1:t-1}$. Figure 2.9 further shows how the per-task performance of each model changes to task sequences of five different orders. We observe that our models show the least disparity in performance to the order of the task sequence.

Preventing catastrophic forgetting We show the effectiveness of APD in preventing catastrophic forgetting by examining how the model performance on earlier tasks changes as new tasks arrive. Figure 2.10 (a)-(c) show the results on task $1, 6, 11$ from CIFAR-100 Superclass, which has 20 tasks in total. APD-Nets do not show any sign of catastrophic forgetting, although their performances marginally change with the arrival of each task. Indeed, APD(2) even improves on task 6 (by $0.40\%p$) as it learns on later tasks, which is possible both due to the update of the shared parameters and the retroactive update of the task-adaptive parameters for earlier tasks, which leads to better solutions.

Selective task forgetting. To show that APD-Net can perform selective task forgetting without any harm to the performance of non-target tasks, in Figure 2.10, (d)-(e), we report the performance change in Task 1-5 when removing parameters for Task 3 and 5. As shown, there is no performance degeneration on non-target tasks, which is expected since dropping out a task-adaptive parameter for a specific task will not affect the task-adaptive parameters for the remaining tasks. This ability to selectively forget is another important advantage of our model that makes it practical in lifelong learning scenarios.

Scalability to a large number of tasks We further validate the scalability of our model with large-scale continual learning experiments on the Omniglot-Rotation dataset, which has 100 tasks. Regardless of random rotations, tasks may share specific features such as circles, curves, and straight lines. [44] showed that we could learn generic representations even with rotated images. They proposed a popular



(a) Task 1 (b) Task 6 (c) Task 11 (d) F. in Step 5 (e) F. in Step 3

Figure 2.10: **(a)-(c) Catastrophic Forgetting** on CIFAR-100 Superclass: Performance of our models on the 1^{st}, 6^{th},and 11^{th} task during continual learning. **(d)-(e) Task Forgetting** on CIFAR-100 Split: Per-task Performance of APD(1) ($T_{1:5}$) when 1^{st} task is dropped during continual learning.

Models	Capacity	Accuracy	AOPD	MOPD
STL	10,000%	82.13% (0.08)	2.79%	5.70%
L2T	100%	63.46% (1.58)	13.35%	24.43%
	1,599%	64.65% (1.76)	11.35%	27.23%
EWC	100%	67.48% (1.39)	14.92%	32.93%
	1,599%	68.66% (1.92)	15.19%	40.43%
PGN	1,045%	73.65% (0.27)	6.79%	19.27%
	1,543%	79.35% (0.12)	4.52%	10.37%
APD(2)	**649%**	**81.20% (0.62)**	**4.09%**	**9.44%**
	943%	**81.60% (0.53)**	**3.78%**	**8.19%**

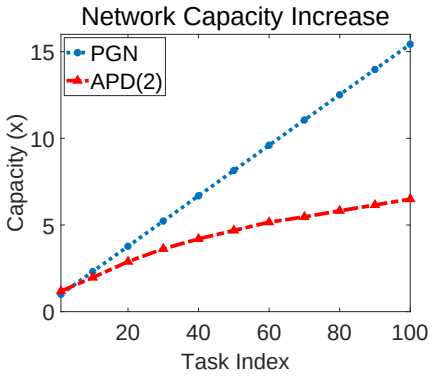


Figure 2.11: **Left:** Performance comparison with several benchmarks on Omniglot-rotation (standard deviation into parenthesis). **Right:** The number of the parameters which is obtained during course of training on Omniglot-rotation.

self-supervised learning technique where they trained the model to predict the rotation angle of randomly rotated images. We do not compare against DEN or RCL for this experiment since they are impractically slow to train. Figure 2.11 Left shows the results of this experiment. For PGN, we restrict the maximum number of links to the adapter to 3 in order to avoid establishing exponentially many connections. We observe that continual learning models achieve significantly lower performance and higher OPDs than single-task learning.

On the contrary, our model outperforms them by a large amount, obtaining performance almost equal to STL, which uses 100 times more network parameters. To show that our model scales well, we plot the number of parameters for our models as a function of the number of tasks in Figure 2.11 Right. The plot shows that our APD-Net scales well, showing logarithmic growth in network capacity (the number of parameters), while PGN shows linear growth. This result suggests that our model is highly efficient, especially in large-scale continual learning scenarios.

Continual learning with heterogeneous datasets We further consider a more challenging continual learning scenario where we train on a series of heterogeneous datasets. For this experiment, we use CIFAR-10, CIFAR100, and the Street View House Numbers (SVHN) dataset, in two different task arrival sequences (SVHN→CIFAR-10→CIFAR-100, CIFAR-100→CIFAR-10→SVHN). We use VGG-16 as the base network, and compare against an additional baseline, Piggyback, which handles a newly arrived task

Table 2.2: Accuracy comparison on diverse datasets according to two opposite task orders (arrows). The results are the mean accuracies over 3 runs of experiments. VGG16 with batch normalization is used for a base network.

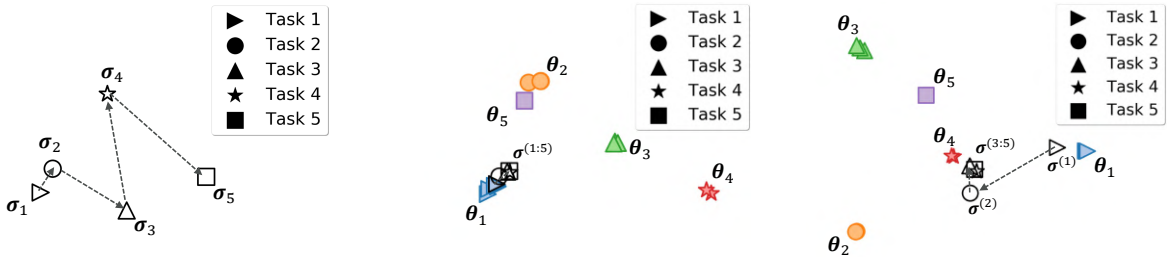
	STL	**L2T**		**Piggyback**		**PGN**		**APD(1)**	
Task Order	None	↓	↑	↓	↑	↓	↑	↓	↑
SVHN	96.8%	10.7%	88.4%	**96.8**%	96.4%	**96.8**%	96.2%	**96.8**%	**96.8**%
CIFAR10	91.3%	41.4%	35.8%	83.6%	90.8%	85.8%	87.7%	**90.1**%	**91.0**%
CIFAR100	67.2%	29.6%	12.2%	41.2%	**67.2**%	41.6%	**67.2**%	**61.1**%	**67.2**%
Average	85.1%	27.2%	45.5%	73.9%	84.8%	74.7%	83.7%	**83.0**%	**85.0**%
Model Size	171MB	57 MB	57 MB	59 MB	59 MB	64 MB	64 MB	63 MB	65 MB

Table 2.3: Ablation study results on APD(1) with average of five different orders depicted in **A.1**. We show the validity of APD as compared with several architectural variants. All experiments were performed on CIFAR-100 split dataset.

Models	**Capacity**	**Accuracy**	**AOPD**	**MOPD**
STL	1,000%	63.75%	0.98%	2.23%
APD(1)	170%	**61.30%**	**1.57%**	**2.77%**
w/o Sparsity	1,084%	**63.47%**	3.20%	5.40%
w/o Adaptive Mask	168%	59.09%	1.83%	3.47%
Fixed $\boldsymbol{\sigma}$	167%	58.55%	2.31%	3.53%

by learning a task-specific binary mask on a network pretrained on ImageNet; since we cannot assume the availability of such large-scale datasets for pretraining in a general setting, we pre-train it on the initial task. Table 2.2 shows the results, which show that existing models obtain suboptimal performance in this setting and are order-sensitive. While Piggyback and PGN are immune to catastrophic forgetting since they freeze the binary masks and hidden units trained on previous tasks, they still suffer from performance degeneration since their performances largely depend upon the pretrained network and the similarity of the later tasks to earlier ones. On the contrary, APD obtains performance close to STL without much increase to the model size and is also order-robust.

Architectural choices for additive parameter decomposition We also evaluate various ablation experiments on Additive Parameter Decomposition in Table 2.3. First, we build the dense APD without sparsity-inducing constraints for task-adaptive parameters while maintaining the essential architecture depicted as **w/o Sparsity**. It significantly outperforms APD in terms of accuracy but is impractical since it requires huge network capacity. We also measure the performance of APD without adaptive masking variables to observe the performance change when the flexibility of APD for newly arriving tasks is limited, referred to as **w/o Adaptive Masking** in the table. Naturally, it underperforms with respect to both accuracy and OPDs. Freezing $\boldsymbol{\sigma}$ after training the first task, referred to as **Fixed $\boldsymbol{\sigma}$** in the table, describes an APD variant that the task-shared knowledge is not properly captured by $\boldsymbol{\sigma}$. Interestingly, this shows significantly lower performance than the others, suggesting the importance of properly learning task-shared knowledge for efficient continual learning.



(a) L2-Transfer (b) APD-Fixed (c) APD(1)

Figure 2.12: **Visualizations of the model parameters during continual learning.** The colored markers denote the parameters for each task i, and the empty markers with black outlines denote the task-shared parameters. Dashed arrows indicate the drift in the parameter space as the model trains on a sequence of tasks.

Qualitative analysis

As a further qualitative analysis of the effect of APD, we visualize the parameters using our method and baselines by projecting them onto a 2D space (Figure 2.12). For this experiment, we use a modified MNIST-split dataset whose images are cropped in the center by 8×8 pixels and create 5 tasks, where each task is the binary classification between two classes. As for the base network, we use a 2-layer multi-layer perceptron with 10 units at each layer. Then we use Principle Component Analysis (PCA) to reduce the dimensionality of the parameters to two. We visualize the 2D projections of both the task-shared and task-adaptive parameters for each step of continual learning. For example, for task 3, we plot three green markers which visualize the parameters when training on task 4 and 5. We only have a single marker for the last task (Task 5) as there is no future task to solve.

We observe that the model parameters using $L2$-Transfer drift away in a new direction as it trains on a sequence of tasks, which brings in catastrophic forgetting. APD-Fixed (Figure 2.12 (b)) largely alleviates the semantic drift, as the update on later tasks only affects the task-shared parts while the task-adaptive parameters are kept intact. However, the update to the task-shared parameters could result in a small drift in the combined task-specific parameters. On the other hand, APD-Net with the retroactive update of task-adaptive parameters successfully prevents the drift in the task-specific parameters (Figure 2.12 (c)).

2.2.5 Summary

We proposed a novel continual learning model with Additive Parameter Decomposition, where the task-shared parameters capture knowledge generic across tasks, and the task-adaptive parameters capture incremental differences over them to capture task-specific idiosyncrasies. This knowledge decomposition naturally solves the catastrophic forgetting problem since the task-adaptive parameters for earlier tasks will remain intact and is significantly more efficient compared to expansion-based approaches since the task-adaptive parameters are additive and do not increase the number of neurons or filters. Moreover, we also introduce and tackle a novel problem we refer to as *task-order sensitivity*, where the performance for each task varies sensitively to the order of task arrival sequence. With our model, the shared parameters will stay relatively static regardless of the task order, and retroactive updates of the task-adaptive parameters prevent them from semantic drift. With extensive experimental validation, we showed that

our model obtains impressive accuracy gains over the existing continual learning approaches while being memory- and computation-efficient, scalable to a large number of tasks, and order-robust. We hope our paper initiates new research directions for continual learning on the relatively unexplored problems of scalability, task-order sensitivity, and selective task forgetting.

Chapter 3. Leverage Indirect Experiences from Different On-device Clients

3.1 Federated Continual Learning with Weighted Inter-client Transfer [174]

3.1.1 Motivation

Continual learning [152, 80, 131, 74, 135] describes a learning scenario where a model continuously trains on a sequence of tasks; it is inspired by the human learning process, as a person learns to perform numerous tasks with large diversity over his/her lifespan, making use of the past knowledge to learn about new tasks without forgetting previously learned ones. Continual learning is a long-studied topic since having such an ability leads to the potential of building general artificial intelligence. However, there are crucial challenges in implementing it with conventional models such as deep neural networks (DNNs), such as *catastrophic forgetting*, which describes the problem where parameters or semantic representations learned for past tasks drift to the direction of new tasks during training. Various prior works have tackled the problem [74, 141, 127], and some recent works tackle other issues, such as scalability or order-robustness [135, 57, 175].

However, all of these models are fundamentally limited in that they can only learn from direct experience - they only learn from the sequence of the tasks they have trained on. Contrarily, humans can learn from *indirect experiences* from others through different means (e.g., verbal communication, books, or various media). Then wouldn't it be beneficial to implement such an ability to a continual learning framework, such that multiple models learning on different machines can learn from the knowledge of the tasks that other clients have already experienced? One problem that arises here is that due to data privacy on individual clients and exorbitant communication costs, it may not be possible to communicate data directly between the clients or between the server and clients.

Federated learning [105, 94, 179] is a learning paradigm that tackles this issue by communicating the parameters instead of the raw data itself. We may have a server that receives the parameters locally trained on multiple clients, aggregates them into a single model parameter, and sends it back to the clients. Motivated by our intuition on learning from indirect experience, we tackle the problem of *Federated Continual Learning (FCL)* where we perform continual learning with multiple clients trained on private task sequences, which communicate their task-specific parameters via a global server. Figure 3.1 depicts an example scenario of FCL.

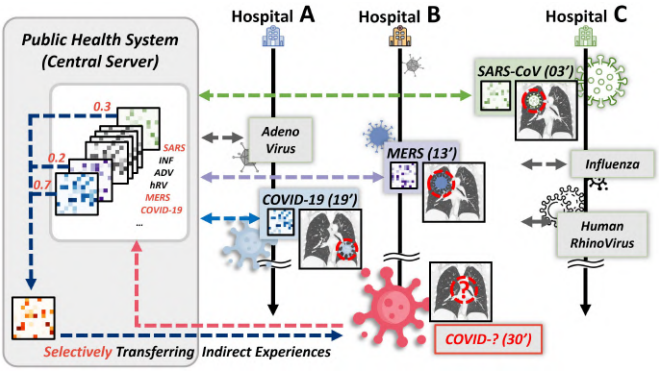


Figure 3.1: **Concept**. A continual learner at a hospital who learns on a sequence of disease prediction tasks may want to utilize relevant task parameters from other hospitals. FCL allows such inter-client knowledge transfer via the communication of task-decomposed parameters.

Suppose that we are building a network of hospitals, each of which has a disease diagnosis model which continuously learns to perform diagnosis given CT scans, for new types of diseases. Then, under our framework, any diagnosis model which has learned about a new type of disease (e.g., COVID-19) will transmit the task-specific parameters to the server, which will redistribute them to other hospitals for the local models to utilize. This allows all participants to benefit from the new task knowledge without compromising data privacy.

Yet, the problem of federated continual learning also brings new challenges. First, there is not only catastrophic forgetting from continual learning but also the **threat of potential interference from other clients**. Figure 3.2 describes this challenge with the results of a simple experiment. Here, we train a model for MNIST digit recognition while communicating the parameters from another client trained on a different dataset. When the knowledge transferred from the other client is relevant to the target task (SVHN), the model converges faster and reaches higher accuracy (**green line**), whereas the model underperforms the base model if the transferred knowledge is highly different from the target task (CIFAR-10, **red line**). Thus, we need to *selectively utilize* knowledge from other clients to minimize *inter-client interference* and maximize *inter-client knowledge transfer*. Another problem with federated learning is efficient communication, as communication costs could become enormous when utilizing the knowledge of the other clients since the communication cost could be the main bottleneck in practical scenarios when working with edge devices. Thus we want the knowledge to be represented as compactly as possible.

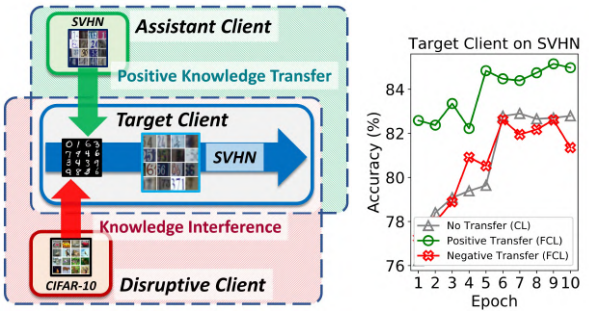


Figure 3.2: **Challenge of Federated Continual Learning.** Interference from other clients, resulting from sharing irrelevant knowledge, may hinder the optimal training of target clients (*Red*), while relevant knowledge from other clients will be beneficial for their learning (*Green*).

To tackle these challenges, we propose a novel framework for federated continual learning, *Federated Weighted Inter-client Transfer (FedWeIT)*, which decomposes the local model parameters into a dense base and sparse task-adaptive parameters. Local clients participating in FedWeIT encode task-specific knowledge into task-adaptive parameters while sharing task-generic knowledge through the base parameters. When we utilize generic knowledge, we also want the client to transfer task-specific knowledge obtained from other clients if it is beneficial. To this end, we allow each model to take a weighted combination of the task-adaptive parameters broadcast from the server such that it can select task-specific knowledge helpful for the task at hand. FedWeIT is communication-efficient since the task-adaptive parameters are *highly sparse* and only need to be communicated once created. Moreover, when communication efficiency is not a critical issue, as in cross-silo federated learning [65], we can use our framework to incentivize each client based on the attention weights on its task-adaptive parameters. We validate our method on multiple scenarios with varying degrees of task similarity across clients against various federated learning and local continual learning models. The results show that our model obtains significantly superior performance over all baselines and adapts faster to new tasks, with largely reduced communication costs.

The main contributions of this paper are as follows:

- We introduce **a new problem of Federated Continual Learning (FCL)**, where multiple models continuously learn on distributed clients, which poses new challenges such as the prevention of

inter-client interference and inter-client knowledge transfer.

- We propose **a novel and communication-efficient framework for federated continual learning**, which allows each client to adaptively update the federated parameter and selectively utilize the past knowledge from other clients by communicating sparse parameters.

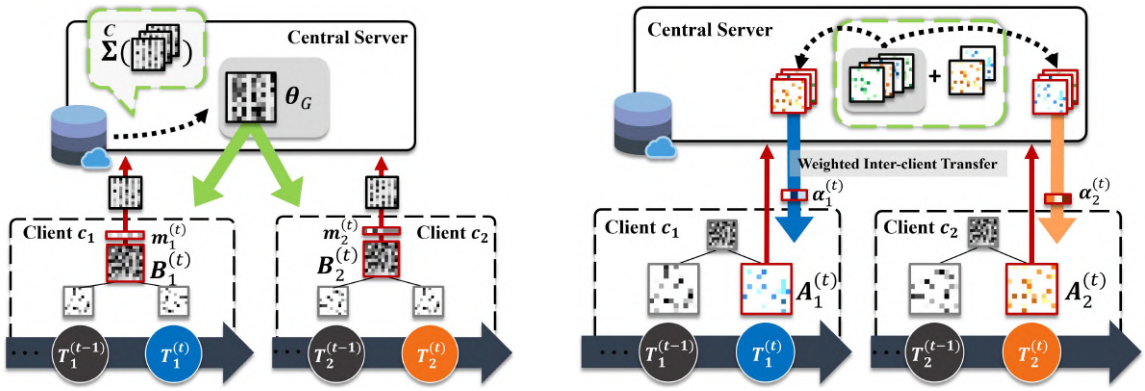
3.1.2 Related Work

Continual learning While continual learning [80, 131] is a long-studied topic with a vast literature, we only discuss recent relevant works. **Regularization-based:** EWC [74] leverages the Fisher Information Matrix to restrict the change of the model parameters such that the model finds the solution that is good for both previous and the current task, and IMM [90] proposes to learn the posterior distribution for multiple tasks as a mixture of Gaussians. Stable SGD [109] shows impressive performance gain through controlling essential hyperparameters and gradually decreasing the learning rate each time a new task arrives. **Architecture-based:** DEN [178] tackles this issue by expanding the network size that is necessary via iterative neuron/filter pruning and splitting, and RCL [170] tackles the same problem using reinforcement learning. APD [175] additively decomposes the parameters into shared and task-specific parameters to minimize the increase in the network complexity. **Coreset-based:** GEM variants [102, 19] minimize the loss on both of actual dataset and stored episodic memory. FRCL [153] memorizes approximated posteriors of previous tasks with sophisticatedly constructed inducing points. To the best of our knowledge, none of the existing approaches considered the communicability for continual learning of deep neural networks, which we tackle. CoLLA [129] aims at solving multi-agent lifelong learning with sparse dictionary learning, it does not have a central server to guide collaboration among clients and is formulated by a simple dictionary learning problem, thus not applicable to modern neural networks. Also, CoLLA is restricted to synchronous training with homogeneous clients.

Federated learning Federated learning is a distributed learning framework under differential privacy, which aims to learn a global model on a server while aggregating the parameters learned by the clients on their private data. FedAvg [105] aggregates the model trained across multiple clients by computing a weighted average of them based on the number of data points trained. FedProx [94] trains the local models with a proximal term which restricts their updates to be close to the global model. FedCurv [142] aims to minimize the model disparity across clients during federated learning by adopting a modified version of EWC. Recent works [179, 156] introduce well-designed aggregation policies by leveraging Bayesian non-parametric methods. A crucial challenge of federated learning is **the reduction of communication cost**. TWAFL [26] tackles this problem by performing layer-wise parameter aggregation, where shallow layers are aggregated at every step, but deep layers are aggregated in the last few steps of a loop. [69] suggests an algorithm for rapid convergence, which minimizes the interference among discrepant tasks at clients by sacrificing the local optimality. This is the opposite direction from personalized federated learning methods [36, 83, 29], which put more emphasis on the performance of local models. FCL is a parallel research direction to both, and to the best of our knowledge, ours is the first work that considers task-incremental learning of clients under the federated learning framework.

3.1.3 Approach

Motivated by the human learning process from indirect experiences, we introduce novel continual learning under the federated learning setting, which we refer to as *Federated Continual Learning (FCL)*.



(a) Communication of General Knowledge (b) Communication of Task-adaptive Knowledge

Figure 3.3: **Updates of *FedWeIT*. (a)** A client sends sparsified federated parameter $\boldsymbol{B}_c \odot \boldsymbol{m}_c^{(t)}$. After that, the server redistributes aggregated parameters to the clients. **(b)** The knowledge base stores previous tasks-adaptive parameters of clients, and each client selectively utilizes them with an attention mask.

FCL assumes that multiple clients are trained on a sequence of tasks from the private data stream while communicating the learned parameters with a global server. We first formally define the problem and then propose naive solutions that straightforwardly combine the existing federated learning and continual learning methods. Then, we discuss two novel challenges that are introduced by federated continual learning and propose a novel framework, *Federated Weighted Inter-client Transfer (FedWeIT)* which can effectively handle the two problems while also reducing the client-to-sever communication cost.

Problem definition

In the standard continual learning (on a single machine), the model iteratively learns from a sequence of tasks $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, ..., \mathcal{T}^{(T)}\}$ where $\mathcal{T}^{(t)}$ is a labeled dataset of t^{th} task, $\mathcal{T}^{(t)} = \{\boldsymbol{x}_i^{(t)}, \boldsymbol{y}_i^{(t)}\}_{i=1}^{N_t}$, which consists of N_t pairs of instances $\boldsymbol{x}_i^{(t)}$ and their corresponding labels $\boldsymbol{y}_i^{(t)}$. Assuming the most realistic situation, we consider the case where the task sequence is a task stream with an unknown arriving order, such that the model can access $\mathcal{T}^{(t)}$ only at the training period of task t which becomes inaccessible afterward. Given $\mathcal{T}^{(t)}$ and the model learned so far, the learning objective at task t is as follows: $\underset{\boldsymbol{\theta}^{(t)}}{\text{minimize}}\ \mathcal{L}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\theta}^{(t-1)}, \mathcal{T}^{(t)})$, where $\boldsymbol{\theta}^{(t)}$ is a set of the model parameters at task t.

We now extend the conventional continual learning to the federated learning setting with multiple clients and a global server. Let us assume that we have C clients, where at each client $c_c \in \{c_1, \dots, c_C\}$ trains a model on a *privately accessible* sequence of tasks $\{\mathcal{T}_c^{(1)}, \mathcal{T}_c^{(2)}, ..., \mathcal{T}_c^{(t)}\} \subseteq \mathcal{T}$. Please note that there is no relation among the tasks $\mathcal{T}_{1:C}^{(t)}$ received at step t, across clients. Now the goal is to effectively train C continual learning models on their own private task streams via communicating the model parameters with the global server, which aggregates the parameters sent from each client, and redistributes them to clients.

Communicable continual learning

In conventional federated learning settings, the learning is done with multiple rounds of local learning and parameter aggregation. At each round of communication r, each client c_c and the server s perform the following two procedures: *local parameter transmission* and *parameter aggregation & broadcasting*. In the local parameter transmission step, for a randomly selected subset of clients at round r,

$\mathcal{C}^{(r)} \subseteq \{c_1, c_2, ..., c_C\}$, each client $c_c \in \mathcal{C}^{(r)}$ sends updated parameters $\boldsymbol{\theta}^{(r)}$ to the server. The server-clients transmission is not done at every client because some of the clients may be temporarily disconnected. Then the server aggregates the parameters $\boldsymbol{\theta}_c^{(r)}$ sent from the clients into a single parameter. The most popular frameworks for this aggregation are FedAvg [105] and FedProx [94]. However, naive federated continual learning with these two algorithms on local sequences of tasks may result in catastrophic forgetting. One simple solution is to use a regularization-based, such as Elastic Weight Consolidation (EWC) [74], which allows the model to obtain a solution that is optimal for both the previous and the current tasks. There exist other advanced solutions [113, 19] that successfully prevent catastrophic forgetting. However, the prevention of catastrophic forgetting at the client level is an orthogonal problem from federated learning.

Thus we focus on challenges that newly arise in this federated continual learning setting. In the federated continual learning framework, the aggregation of the parameters into a global parameter $\boldsymbol{\theta}_G$ allows inter-client knowledge transfer across clients, since a task $\mathcal{T}_i^{(q)}$ learned at client c_i at round q may be similar or related to $\mathcal{T}_j^{(r)}$ learned at client c_j at round r. Yet, using a single aggregated parameter $\boldsymbol{\theta}_G$ may be suboptimal in achieving this goal since knowledge from irrelevant tasks may not be helpful or even hinder the training of each client by altering its parameters into incorrect directions, which we describe as *inter-client interference*. Another problem that is also practically important is *communication efficiency*. The parameter transmission between the server and clients will incur high communication costs, which will be problematic for the continual learning setting since the clients may train on unlimited streams of tasks.

Federated weighted inter-client transfer

How can we then maximize the *knowledge transfer* between clients while minimizing the *inter-client interference* and communication cost? We now describe our model, *Federated Weighted Inter-client Transfer (FedWeIT)*, which can resolve these two problems that arise with a naive combination of continual learning approaches with the federated learning framework.

As briefly alluded to earlier, the main cause of the problems is that the knowledge of all tasks learned at multiple clients is stored in a single set of parameters $\boldsymbol{\theta}_G$. However, for the knowledge transfer to be effective, each client should selectively utilize only the knowledge of the relevant tasks that other clients train. This **selective transfer** is also the key to minimizing inter-client interference and disregarding the knowledge of irrelevant tasks that may interfere with learning.

We tackle this problem by decomposing the parameters into three different types of parameters with different roles: *global parameters* ($\boldsymbol{\theta}_G$) that capture the global and generic knowledge across all clients, *local base parameters* ($\boldsymbol{B}$) which capture generic knowledge for each client, and *task-adaptive parameters* ($\boldsymbol{A}$) for each specific task per client, motivated by [175]. A set of the model parameters $\boldsymbol{\theta}_c^{(t)}$ for task t at continual learning client c_c is then defined as follows:

$$\boldsymbol{\theta}_c^{(t)} = \boldsymbol{B}_c^{(t)} \odot \boldsymbol{m}_c^{(t)} + \boldsymbol{A}_c^{(t)} + \sum_{i \in \mathcal{C}_{nc}} \sum_{j<|t|} \alpha_{i,j}^{(t)} \boldsymbol{A}_i^{(j)} \tag{3.1}$$

where $\boldsymbol{B}_c^{(t)} \in \{\mathbb{R}^{I_l \times O_l}\}_{l=1}^{L}$ is the set of base parameters for c^{th} client shared across all tasks in the client, $\boldsymbol{m}_c^{(t)} \in \{\mathbb{R}^{O_l}\}_{l=1}^{L}$ is the set of sparse vector masks which allows to adaptively transform $\boldsymbol{B}_c^{(t)}$ for the task t, $\boldsymbol{A}_c^{(t)} \in \{\mathbb{R}^{I_l \times O_l}\}_{l=1}^{L}$ is the set of sparse task-adaptive parameters at client c_c. Here, L is the number of the layer in the neural network, and I_l, O_l are the input and output dimensions of the weights at layer l, respectively.

The first term allows selective utilization of global knowledge. We want the base parameter $\boldsymbol{B}_c^{(t)}$ at each client to capture generic knowledge across all tasks across all clients.

As illustrated in Figure 3.3 (a), we initialize the base parameter at each round t with the global parameter from the previous iteration ($\boldsymbol{\theta}_G^{(t-1)}$) allowing $\boldsymbol{B}_c^{(t)}$ to also benefit from *global* knowledge about all the tasks. However, since $\boldsymbol{\theta}_G^{(t-1)}$ also contains knowledge irrelevant to the current task, we learn that the sparse mask selects only the relevant parameters for the given task instead of using it as is. This weighted selection helps minimize inter-client interference and also allows for efficient communication. The second term indicates task-adaptive parameters. Since we additively decompose the parameters, $\boldsymbol{A}_c^{(t)}$ aims to capture relevant knowledge to task $\mathcal{T}_c^{(t)}$, which is not captured from the first term. The final term describes weighted inter-client knowledge transfer. We have a set of parameters transmitted from the server, containing all task-adaptive parameters from all the clients. To selectively utilize these indirect experiences from other clients, we further allocate attention $\boldsymbol{\alpha}_c^{(t)}$ to these parameters to take a weighted combination of them. By learning this attention, each client can select only the relevant task-adaptive parameters that help learn the given task. Although we design $\boldsymbol{A}_i^{(j)}$ to be highly sparse, using about $2-3\%$ of memory of full parameter in practice, sending all task knowledge is not desirable. Thus we transmit the randomly sampled task-adaptive parameters across all time steps from the knowledge base, which we empirically find to achieve good results in practice.

Training of local clients We learn the decomposable parameter $\boldsymbol{\theta}_c^{(t)}$ by optimizing for the following objective:

$$\begin{aligned}\underset{\boldsymbol{B}_c^{(t)},\, \boldsymbol{m}_c^{(t)},\, \boldsymbol{A}_c^{(1:t)},\, \boldsymbol{\alpha}_c^{(t)}}{\text{minimize}} \; & \mathcal{L}\left(\boldsymbol{\theta}_c^{(t)}; \mathcal{T}_c^{(t)}\right) + \lambda_1 \Omega(\{\boldsymbol{m}_c^{(t)}, \boldsymbol{A}_c^{(1:t)}\}) \\ & + \lambda_2 \sum_{i=1}^{t-1} \|\Delta \boldsymbol{B}_c^{(t)} \odot \boldsymbol{m}_c^{(i)} + \Delta \boldsymbol{A}_c^{(i)}\|_2^2,\end{aligned} \tag{3.2}$$

where $\mathcal{L}$ is a loss function and $\Omega(\cdot)$ is a sparsity-inducing regularization term for all task-adaptive parameters and the masking variable (we use ℓ_1-norm regularization) to make them sparse. The second regularization term is used for the retroactive update of the past task-adaptive parameters, which helps the task-adaptive parameters to maintain the original solutions for the target tasks by reflecting the change of the base parameter. Here, $\Delta \boldsymbol{B}_c^{(t)} = \boldsymbol{B}_c^{(t)} - \boldsymbol{B}_c^{(t-1)}$ is the difference between the base parameter at the current and previous timestep, and $\Delta \boldsymbol{A}_c^{(i)}$ is the difference between the task-adaptive parameter for task i at the current and previous timestep. This regularization is essential for preventing catastrophic forgetting. λ_1 and λ_2 are hyperparameters controlling the effect of the two regularizers.

Efficient communication via sparse parameters FedWeIT learns via server-to-client communication. As discussed earlier, a crucial challenge here is to reduce communication costs. We describe what happens at the client and the server at each step.

Client: At each round r, each client c_c partially updates its base parameter with the nonzero components of the global parameter sent from the server; that is, $\boldsymbol{B}_c(n) = \boldsymbol{\theta}_G(n)$ where n is a nonzero element of the global parameter. After training the model using Equation 3.2, it obtains a sparsified base parameter $\widehat{\boldsymbol{B}}_c^{(t)} = \boldsymbol{B}_c^{(t)} \odot \boldsymbol{m}_c^{(t)}$ and task-adaptive parameter $\boldsymbol{A}_c^{(t)}$ for the new task, both of which are sent to the server, at the smaller cost compared to naive FCL baselines. While naive FCL baselines require $|\mathcal{C}| \times R \times |\boldsymbol{\theta}|$ for client-to-server communication, FedWeIT requires $|\mathcal{C}| \times (R \times |\widehat{\boldsymbol{B}}| + |\boldsymbol{A}|)$ where R is the number of communication round per task and $|\cdot|$ is the number of parameters.

Algorithm 6 Federated Weighted Inter-client Transfer

input Dataset $\{\mathcal{D}_c^{(1:t)}\}_{c=1}^{C}$, global parameter $\boldsymbol{\theta}_G$, hyperparameters λ_1, λ_2, knowledge base $kb \leftarrow \{\}$
output $\{\boldsymbol{B}_c,\ \boldsymbol{m}_c^{(1:t)},\ \boldsymbol{\alpha}_c^{(1:t)},\ \boldsymbol{A}_c^{(1:t)}\}_{c=1}^{C}$

Initialize $\boldsymbol{B}_c$ to $\boldsymbol{\theta}_G$ for all clients $\mathcal{C} \equiv \{c_1, ..., c_C\}$
for task $t = 1, 2, ...$ **do**
  Randomly sample knowledge base $kb^{(t)} \sim kb$
  for round $r = 1, 2, ...$ **do**
    Collect communicable clients $\mathcal{C}^{(r)} \sim \mathcal{C}$
    Distribute $\boldsymbol{\theta}_G$ and $kb^{(t)}$ to client $c_c \in \mathcal{C}^{(r)}$ **if** c_c meets $kb^{(t)}$ first, **otherwise** distribute only $\boldsymbol{\theta}_G$
    Minimize Equation 3.2 for solving local CL problems
    $\boldsymbol{B}_c^{(t,r)} \odot \boldsymbol{m}_c^{(t,r)}$ are transmitted from $\mathcal{C}^{(r)}$ to the server
    Update $\boldsymbol{\theta}_G \leftarrow \frac{1}{|\mathcal{C}^{(r)}|} \sum_c \boldsymbol{B}_c^{(t,r)} \odot \boldsymbol{m}_c^{(t,r)}$
  end for
  Update knowledge base $kb \leftarrow kb \cup \{\boldsymbol{A}_j^{(t)}\}_{j \in \mathcal{C}}$
end for

Server: The server first aggregates the base parameters sent from all the clients by taking a weighted average of them: $\boldsymbol{\theta}_G = \frac{1}{\mathcal{C}} \sum_{\mathcal{C}} \widehat{\boldsymbol{B}}_i^{(t)}$. Then, it broadcasts $\boldsymbol{\theta}_G$ to all the clients. Task adaptive parameters of $t-1$, $\{\boldsymbol{A}_i^{(t-1)}\}_{i=1}^{\mathcal{C}_{nc}}$ are broadcast at once per client during training task t. While naive FCL baselines requires $|\mathcal{C}| \times R \times |\boldsymbol{\theta}|$ for server-to-client communication cost, FedWeIT requires $|\mathcal{C}| \times (R \times |\boldsymbol{\theta}_G| + (|\mathcal{C}|-1) \times |\boldsymbol{A}|)$ in which $\boldsymbol{\theta}_G$, $\boldsymbol{A}$ are highly sparse. We describe the FedWeIT algorithm in Algorithm 6.

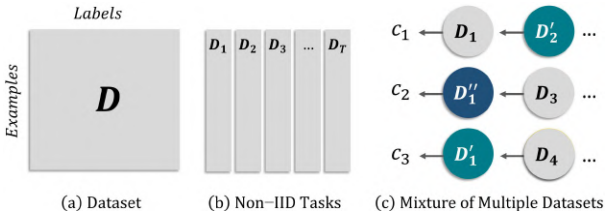


Figure 3.4: **Configuration of task sequences:** We first split a dataset D into multiple sub-tasks in non-IID manner ((a) and (b)). Then, we distribute them to multiple clients ($C_{\#}$). Mixed tasks from multiple datasets (colored circles) are distributed across all clients ((c)).

3.1.4 Experimental Results

We use a modified version of LeNet [85] for the experiments with both Overlapped-CIFAR-100 and NonIID-50 datasets. Further, we use ResNet-18 [55] with NonIID-50 dataset. We followed other experimental setups from [137] and [175].

Baselines and our model

STL: Single Task Learning at each arriving task.
EWC: Individual continual learning with *EWC* [74] per client.
Stable-SGD: Individual continual learning with *Stable-SGD* [109] per client.
APD: Individual continual learning with *APD* [175] per client.
FedProx: Federated continual learning using *FedProx* [94] algorithm.
Scaffold: Federated continual learning using *Scaffold* [69] algorithm.

FedCurv: Federated continual learning using *FedCurv* [142] algorithm.
FedProx-[*model*]: FCL, that is trained using *FedProx* algorithm with [*model*].
FedWeIT: Our FedWeIT algorithm.

Datasets

We validate our **FedWeIT** under different configurations of task sequences against baselines which are namely Overlapped-CIFAR-100 and NonIID-50.
Overlapped-CIFAR-100: We group 100 classes of CIFAR-100 dataset into 20 non-iid superclasses tasks. Then, we randomly sample 10 tasks out of 20 tasks and split instances to create a task sequence for each of the clients with overlapping tasks.
NonIID-50: We use the following eight benchmark datasets: MNIST [85], CIFAR-10/-100 [77], SVHN [110], Fashion-MNIST [165], Not-MNIST [14], FaceScrub [112], and TrafficSigns [150].
We split the classes in the 8 datasets into 50 non-IID tasks, each of which is composed of 5 classes that are disjoint from the classes used for the other tasks. This is a large-scale experiment containing $280,000$ images of 293 classes from 8 heterogeneous datasets. After generating and processing tasks, we randomly distribute them to multiple clients as illustrated in Figure 3.4. We followed metrics for accuracy and forgetting from recent works [18, 109, 108].

Quantitative analysis

We first validate our model on both Overlapped-CIFAR-100 and NonIID-50 task sequences against single task learning (STL), continual learning (EWC, APD), federated learning (FedProx, Scaffold, FedCurv), and naive federated continual learning (FedProx-based) baselines. Table 3.1 shows the final average per-task performance after the completion of (federated) continual learning on both datasets. We observe that FedProx-based federated continual learning (FCL) approaches degenerate the performance of continual learning (CL) methods over the same methods without federated learning. This is because the aggregation of all client parameters learned on irrelevant tasks results in severe interference in the learning for each task, leading to catastrophic forgetting and suboptimal task adaptation. Scaffold performs poorly on FCL, as its regularization on the local gradients is harmful to FCL, where all clients learn from different task sequences. While FedCurv reduces the inter-task disparity in parameters, it cannot minimize inter-task interference, resulting in underperforming single-machine CL methods. On the other hand, FedWeIT significantly outperforms single-machine CL and naive FCL baselines on both datasets. Even with a larger number of clients ($C = 100$), FedWeIT consistently outperforms all baselines (Figure 3.5). This improvement largely owes to FedWeIT's ability to selectively utilize the knowledge from other clients to rapidly adapt to the target task and obtain better final performance.

The fast adaptation to a new task is another clear advantage of inter-client knowledge transfer. To further demonstrate the practicality of our method with larger networks, we experiment on Non-IID dataset with ResNet-18 (Table 3.3), on which FedWeIT still significantly outperforms the strongest baseline (FedProx-APD) while using fewer parameters.

Efficiency of FedWeIT We also report the accuracy as a function of network capacity in Tables 3.1 to 3.3, which we measure by the number of parameters used. We observe that FedWeIT obtains much higher accuracy while utilizing less number of parameters than FedProx-APD. This efficiency mainly

Table 3.1: Averaged Per-task performance on both datasets during FCL with 5 clients (fraction=1.0). We measured task accuracy and model size after completing all learning phases over 3 individual trials. We also measured C2S/S2C communication cost for training each task.

		NonIID-50 Dataset (F=1.0, R=20)				
Methods	FCL	Accuracy	Forgetting	Model Size	Client to Server Cost	Server to Client Cost
EWC [74]	×	74.24 (± 0.11)	0.10 (± 0.01)	61 MB	N/A	N/A
Stable SGD [109]	×	76.22 (± 0.26)	0.14 (± 0.01)	61 MB	N/A	N/A
APD [175]	×	81.42 (± 0.89)	0.02 (± 0.01)	90 MB	N/A	N/A
FedProx [94]	✓	68.03 (± 2.14)	0.17 (± 0.01)	61 MB	1.22 GB	1.22 GB
Scaffold [69]	✓	30.84 (± 1.41)	0.11 (± 0.02)	61 MB	2.44 GB	2.44 GB
FedCurv [142]	✓	72.39 (± 0.32)	0.13 (± 0.02)	61 MB	1.22 GB	1.22 GB
FedProx-EWC	✓	68.27 (± 0.72)	0.12 (± 0.01)	61 MB	1.22 GB	1.22 GB
FedProx-Stable-SGD	✓	75.02 (± 1.44)	0.12 (± 0.01)	79 MB	1.22 GB	1.22 GB
FedProx-APD	✓	81.20 (± 1.52)	0.01 (± 0.01)	79 MB	1.22 GB	1.22 GB
FedWeIT (Ours)	✓	**84.11** (± 0.27)	**0.00** (± 0.00)	78 MB	0.37 GB	1.08 GB
Single Task Learning	×	85.78 (± 0.17)	–	610 MB	N/A	N/A
		Overlapped CIFAR-100 Dataset (F=1.0, R=20)				
Methods	FCL	Accuracy	Forgetting	Model Size	Client to Server Cost	Server to Client Cost
EWC [74]	×	44.26 (± 0.53)	0.13 (± 0.01)	61 MB	N/A	N/A
Stable SGD [109]	×	43.31 (± 0.44)	0.08 (± 0.01)	61 MB	N/A	N/A
APD [175]	×	50.82 (± 0.41)	0.02 (± 0.01)	73 MB	N/A	N/A
FedProx [94]	✓	38.96 (± 0.37)	0.13 (± 0.02)	61 MB	1.22 GB	1.22 GB
Scaffold [69]	✓	22.80 (± 0.47)	0.09 (± 0.01)	61 MB	2.44 GB	2.44 GB
FedCurv [142]	✓	40.36 (± 0.44)	0.15 (± 0.02)	61 MB	1.22 GB	1.22 GB
FedProx-EWC	✓	41.53 (± 0.39)	0.13 (± 0.01)	61 MB	1.22 GB	1.22 GB
FedProx-Stable-SGD	✓	43.29 (± 1.45)	0.07 (± 0.01)	61 MB	1.22 GB	1.22 GB
FedProx-APD	✓	52.20 (± 0.50)	0.02 (± 0.01)	75 MB	1.22 GB	1.22 GB
FedWeIT (Ours)	✓	**55.16** (± 0.19)	**0.01** (± 0.00)	75 MB	0.37 GB	1.07 GB
Single Task Learning	×	57.15 (± 0.07)	–	610 MB	N/A	N/A

comes from reusing task-adaptive parameters from other clients, which is impossible with single-machine CL or naive FCL methods.

We also examine the communication cost (the size of non-zero parameters transmitted) of each method. Table 3.1 reports both the *client-to-server* (C2S) / *server-to-client* (S2C) communication cost at training each task. FedWeIT uses only 30% and 3% of parameters for $\widehat{\boldsymbol{B}}$ and $\boldsymbol{A}$ of the dense models, respectively. We observe that FedWeIT is significantly more communication-efficient than FCL baselines, although it broadcasts task-adaptive parameters due to the high sparsity of the parameters. Figure 3.7 shows the accuracy as a function of C2S cost according to the transmission of top-κ% informative parameters. Since FedWeIT selectively utilizes task-specific parameters learned from other clients, it results in superior performance over APD baselines, especially with sparse communication of model parameters.

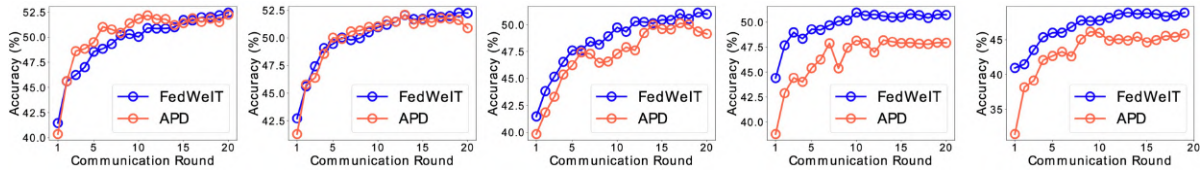
Catastrophic forgetting Further, we examine how the performance of the past tasks changes during continual learning, to see the severity of catastrophic forgetting with each method. Figure 3.6 shows the performance of FedWeIT and FCL baselines on all tasks at the end of training for later tasks. We observe that naive FCL baselines suffer from more severe catastrophic forgetting than local continual learning with EWC because of *inter-client interference*, where the knowledge of irrelevant tasks from other clients overwrites the knowledge of the past tasks. Contrarily, our model shows no sign of catastrophic forgetting.

Table 3.2: Average Per-task Performance on Overlapped-CIFAR-100 during FCL with 100 clients.

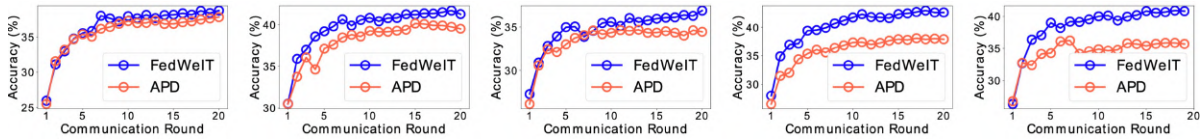
100 clients (F=0.05, R=20, **1,000 tasks** in total)			
Methods	Accuracy	Forgetting	Model Size
FedProx	24.11 (± 0.44)	0.14 (± 0.01)	1.22 GB
FedCurv	29.11 (± 0.20)	0.09 (± 0.02)	1.22 GB
FedProx-SSGD	22.29 (± 0.51)	0.14 (± 0.01)	1.22 GB
FedProx-APD	32.55 (± 0.29)	0.02 (± 0.01)	6.97 GB
FedWeIT (Ours)	**39.58** (± 0.27)	**0.01** (± 0.00)	4.03 GB
STL	32.96 (± 0.23)	–	12.20 GB

Table 3.3: FCL results on NonIID-50 dataset with ResNet-18.

	ResNet-18 (F=1.0, R=20)		
Methods	Accuracy	Forgetting	Model Size
APD	92.44 (± 0.17)	0.02 (± 0.00)	1.86 GB
FedProx-APD	92.89 (± 0.22)	0.02 (± 0.01)	2.05 GB
FedWeIT (Ours)	**94.86** (± 0.13)	**0.00** (± 0.00)	**1.84** GB



(a) Federated continual learning with 20 clients



(b) Federated continual learning with 100 clients

Figure 3.5: **Task adaptation comparison with FedWeIT and APD** using 20 clients and 100 clients. We visualize the last 5 tasks out of 10 tasks per client. *Overlapped-CIFAR-100* datasets are used after splitting instances according to the number of clients (20 and 100).

This is mainly due to the selective utilization of the prior knowledge learned from other clients through the global/task-adaptive parameters, which allows it to effectively alleviate *inter-client interference*. FedProx-APD also does not suffer from catastrophic forgetting but yields inferior performance due to ineffective knowledge transfer.

Weighted inter-client knowledge transfer By analyzing the attention $\boldsymbol{\alpha}$ in Equation 3.1, we examine which task parameters from other clients each client selected. Figure 3.8 shows examples of the attention weights learned for the 0^{th} split of *MNIST* and the 10^{th} split of *CIFAR-100*. We observe that strong attention is allocated to the task parameters from the same (CIFAR-100 utilizes parameters from CIFAR-100 tasks with disjoint classes) or similar datasets (MNIST utilizes parameters from Traffic Sign and SVHN). As shown, FedWeIT effectively selects beneficial parameters to maximize inter-client knowledge transfer, and this is impressive since it does not know which datasets the parameters are trained on.

Asynchronous federated continual learning We now consider FedWeIT under the asynchronous federated continual learning scenario, where there is no synchronization across clients for each task. This is a more realistic scenario since each task may require different training rounds to converge during federated continual learning. Here, asynchronous implies that each task requires different training costs (i.e., time, epochs, or rounds). Under the asynchronous federated learning scenario, FedWeIT transfers

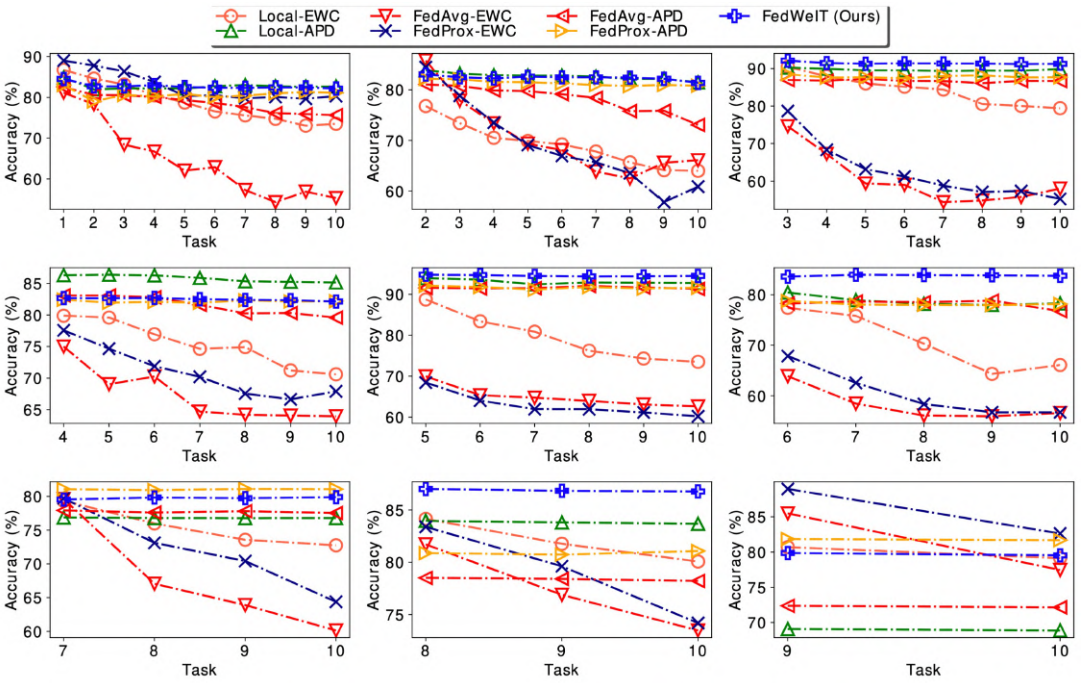


Figure 3.6: **Forgetting Analysis** Performance change over the increasing number of tasks for all tasks except the last task (1^{st} to 9^{th}) during federated continual learning on *NonIID-50*. We observe that our method does not suffer from task forgetting any tasks.

any available task-adaptive parameters from the knowledge base to each client. We plot the average test accuracy in Figure 3.9 over all tasks during synchronous/asynchronous federated continual learning. In asynchronous FedWeIT, each task requires different training rounds and receives new tasks and task-adaptive parameters in an asynchronous manner. The performance of asynchronous FedWeIT is almost similar to that of synchronous FedWeIT.

Effect of the communication frequency We analyze the effect of the communication frequency by comparing the performance, measured by the number of training epochs per communication round. We run the 4 different FedWeIT with $1, 2, 5,$, and 20 training epochs per round. Figure 3.10 shows the performance of our FedWeIT variants. As clients frequently update the model parameters through communication with the central server, the model gets higher performance while maintaining a smaller network capacity since the model with frequent communication efficiently updates the model parameters

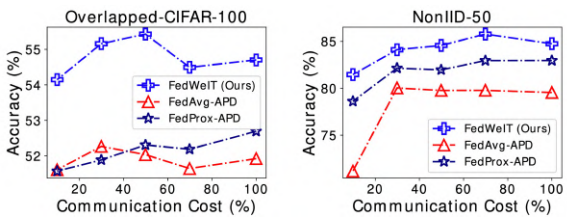


Figure 3.7: **Accuracy over client-to-server cost.** We report the relative communication cost to the original network. All results are averaged over the 5 clients.

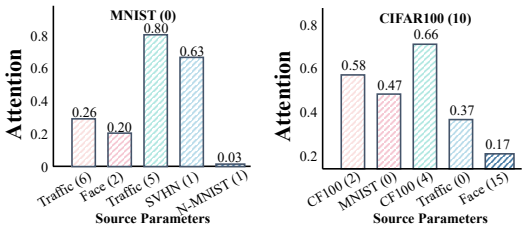


Figure 3.8: **Inter-client transfer** for NonIID-50. We compare the scale of the attentions at the first FC layer which gives the weights on transferred task-adaptive parameters from other clients.

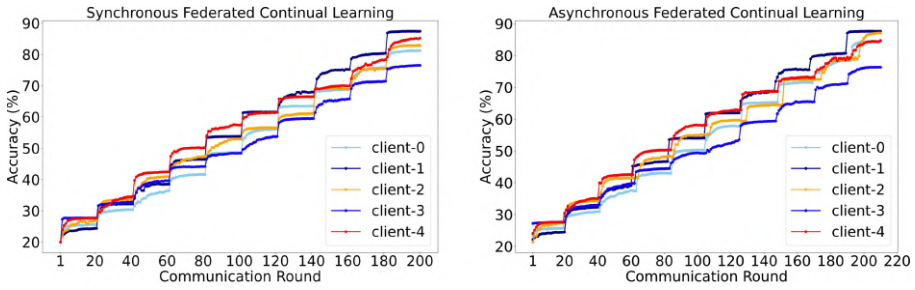
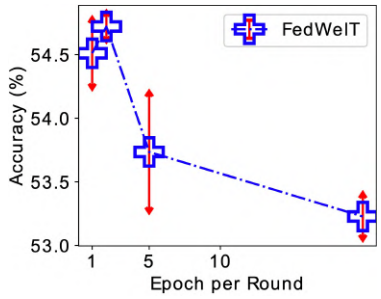


Figure 3.9: **FedWeIT with asynchronous federated continual learning** on *Non-iid 50* dataset. We measure the test accuracy of all tasks per client.



	Overlapped-CIFAR-100			
Methods	**Accuracy**	**Model Size**	**C2S/S2C Cost**	**Epochs / Round**
FedWeIT	**55.16** ± 0.19	**0.075** GB	0.37 / 1.07 GB	**1**
FedWeIT	**55.18** ± 0.08	0.077 GB	0.19 / 0.53 GB	**2**
FedWeIT	53.73 ± 0.44	0.083 GB	0.08 / 0.22 GB	**5**
FedWeIT	53.22 ± 0.14	0.088 GB	0.02 / 0.07 GB	**20**

Figure 3.10: **Number of Epochs per Round** We show error bars over the number of training epochs per communication rounds on *Overlapped-CIFAR-100* with 5 clients. All models transmit full of local base parameters and highly sparse task-adaptive parameters. All results are the mean accuracy over 5 clients and we run 3 individual trials. Red arrows at each point describes the standard deviation of the performance.

by transferring the inter-client knowledge. However, it requires much higher communication costs than the model with sparser communication. For example, the model trained for one epoch at each round may require about 16.9 times higher entire communication cost than the model trained for 20 epochs at each round. Hence, there is a trade-off between the model performance of federated continual learning and communication efficiency, whereas FedWeIT variants consistently outperform (federated) continual learning baselines.

Ablation study with model components We perform an ablation study to analyze the role of each component of our FedWeIT. We compare the performance of four different variations of our model. **w/o B communication** describes the model that does not transfer the base parameter **B** and only

Table 3.4: Ablation studies to analyze the effectiveness of parameter decomposition on WeIT. All experiments were performed on NonIID-50 dataset.

	NonIID-50		
Methods	**Acc.**	**M Size**	**C2S/S2C Cost**
FedWeIT	**84.11%**	**0.078** GB	**0.37 / 1.07** GB
w/o B comm.	77.88%	0.070 GB	**0.01 / 0.01** GB
w/o A comm.	79.21%	0.079 GB	0.37 / 1.04 GB
w/o A	65.66%	0.061 GB	0.37 / 1.04 GB
w/o m	78.71%	0.087 GB	1.23 / 1.25 GB

Table 3.5: Ablation Study on Knowledge Transfer (NonIID-50).

Method	Avg. Accuracy	Memory size	BwT
FedWeIT	**84.43% (± 0.50)**	68.93 MB	-0.0014
FedWeIT w/o ℓ_1	87.12% (± 0.24)	354.41 MB	**-0.0007**
FedWeIT w/o ℓ_2	56.76% (± 0.84)	**63.44** MB	-0.3203

communicates task-adaptive ones. **w/o A communication** is the model that does not communicate task-adaptive parameters. **w/o A** is the model which trains the model only with sparse transmission of local base parameter, and **w/o m** is the model without the sparse vector mask.

As shown in Table 3.4, without communicating **B** or **A**, the model yields significantly lower performance compared to the full model since they do not benefit from *inter-client knowledge transfer*. The model **w/o A** obtains very low performance due to catastrophic forgetting, and the model **w/o** sparse mask **m** achieves lower accuracy with larger capacity and cost, which demonstrates the importance of performing selective transmission. We also perform the additional analysis by eliminating the proposed regularization terms in Table 3.5. Without ℓ_1 term, the method achieves even better performance but requires significantly larger memory. On the other hand, the method removing ℓ_2 term suffers from forgetting.

3.1.5 Summary

We tackled a novel problem of federated continual learning, which continuously learns local models at each client while allowing it to utilize indirect experience (task knowledge) from other clients. This poses new challenges, such as inter-client knowledge transfer and prevention of inter-client interference between irrelevant tasks. To tackle these challenges, we additively decomposed the model parameters for each client into the global parameters shared across all clients and the sparse local task-adaptive parameters specific to each task. Further, we allowed each model to selectively update the global task-shared parameters and utilize beneficial task-adaptive parameters from other clients. The experimental validation of our model under various task similarities across clients against existing federated learning and continual learning baselines shows that our model obtains significantly outperforms baselines with reduced communication cost. We believe that federated continual learning is a practically important topic of large interest to both research communities of continual learning and federated learning that will lead to new research directions.

3.2 Bitwidth Heterogeneous Federated Learning with Progressive Weight Dequantization [177]

3.2.1 Motivation

In recent decades, the drastic evolution of hardware technologies for edge devices has changed our lives from the root. Intelligent edge devices, which have the ability to process sensory inputs and communicate, such as embedded sensors, drones, phones, smartwatches, and augmented reality glasses, are now being used in our everyday lives. Such accessibility of edge devices has led to the emergence of *Federated Learning* (FL) [105, 187, 26], a learning framework in which multiple clients collaboratively train on private local data while periodically communicating the trained models across them, often through a server that aggregates and broadcasts the local models. Many previous works have investigated the potential and applicability of FL in various learning frameworks, such as semi-supervised learning [61], graph neural networks [106, 162, 6], meta-learning [63, 36], and continual learning [174].

A crucial challenge in FL is that there could be a large discrepancy among participants in their data distribution, tasks, model architectures, and devices which often leads to incompatibility of the models that are being aggregated. This problem is often referred to as Heterogeneous Federated Learning [62, 99, 52, 31] problem. Many existing works have successfully alleviated the adverse effect of data-, model-, and device-heterogeneity. However, the most basic assumption, even in such heterogeneous FL scenarios, is that all models have the same bitwidths.

Yet, in real-world FL scenarios, participating devices may have heterogeneous bitwidth specifications. Suppose that we are building a federated network of various healthcare providers, such as hospitals, community health centers, and clinics, as well as end-users, and even wearable devices, i.e., smartwatches. Each client has a health disorder prediction model for the users themselves or their patients that continuously learns to perform a diagnosis given the heart rates or bioelectric signals. The scenario allows the participation of local clients with various hardware infrastructures, some of them using models built under lightweight devices based on low-bitwidth hardware operations using FPGA, ASIC, Raspberry Pi, or Edge GPUs. Here, BHFL enables local devices with different hardware specifications to participate in a single federated learning framework without the need for uniformity of the infrastructure, enhancing the pool of devices that could participate in collaborative learning. We refer to this practical FL scenario as Bitwidth Heterogeneous Federated Learning (BHFL), which we illustrate in Figure 3.11.

Tackling BHFL is a nontrivial problem, as it poses new challenges such as (i) suboptimal loss convergence due to distributional shift after aggregating the weights of bitwidth-heterogeneous models and (ii) inherent limitation of expressive power in low-bitwidth weights. As shown in Table 3.6, existing

Table 3.6: Categorization of existing methods for bitwidth heterogeneous federated learning.

Methods	FL Type	$\text{Bits}_{\text{Server}}$	$\text{Bits}_{\text{Clients}}$	$\text{Bits}_{\text{Uplink}}$	$\text{Bits}_{\text{Downlink}}$	Communication
FedAvg [105]	FL	Float32	Float32	Float32	Float32	Weights
FedProx [94]	FL	Float32	Float32	Float32	Float32	Weights
FedPAQ [126]	QPC[1]	Float32	Float32	Target bits	Float32	$Q_{\text{Target\_bits}}(\text{Diffs})$[2]
FedCOM [49]	QPC	Float32	Float32	Target bits	Float32	$Q_{\text{Target\_bits}}(\text{Diffs})$
FedCOMGATE [49]	QPC	Float32	Float32	Target bits	Float32×2	$Q_{\text{Target\_bits}}(\text{Diffs})$
ProWD (Ours)	BHFL	Float32	Client-specific	Client-specific	Client-specific	W_Q[3]

[1] Quantized Parameter Communication (QPC) [2] $Q_{\text{Target\_bits}}(w)$: A function quantizes input w to the target bitwidth
[3] Weights obtained from clients' bit-dependent training

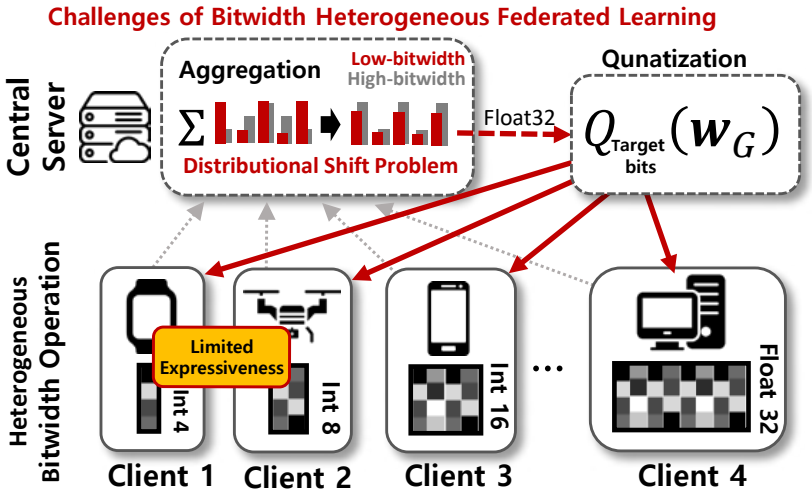


Figure 3.11: **Bitwidth Heterogeneous Federated Learning.** We consider the FL setting where the participating devices have heterogeneous bitwidths. FL with different bitwidth models may cause detrimental side effects due to (i) the distributional shift of model weights, and (ii) the limited expressiveness in low-bit weights.

methods cannot appropriately handle the new setting due to these challenges. While a line of works proposes quantizing model weights to reduce the communication cost when transmitting them to the server, they assume that full-precision weights are being used at the local devices. However, processing full-precision weights may not be possible for resource-limited hardware devices.

In this paper, we propose a novel framework that can successfully deal with the new challenges posed by the BHFL problem, which we name **Pro**gressive **W**eight **D**equantization (**ProWD**). The ProWD framework enhances the compatibility across bit-heterogeneous weights with selective weight aggregation and dequantization. The selective weight aggregation discards outliers from the low-precision weights, and the trainable dequantizer at the server recovers high-precision weight information from the given low-bitwidth weights. These two methods collaboratively alleviate the information loss resulting from aggregating incompatible weights with heterogeneous bitwidths.

We evaluate the performance of our ProWD on various benchmark datasets and show that our method significantly outperforms previous FL methods that consider weight quantization while also outperforming naive heuristics to tackle the bitwidth heterogeneity, such as group averaging. We also provide comprehensive analyses which show that existing FL methods suffer from poor convergence and adaptation under the bit-heterogeneous federated learning scenario, while ProWD consistently increases the performance of all local models regardless of their bitwidths.

In summary, our contributions are threefold:

- We define a practical federated learning scenario where the participating devices may have largely different bitwidths, which brings in new challenges such as degenerating distributional shifts of federated weights and limited expressive power of low-bitwidth models.
- We propose *ProWD*, a novel framework that effectively tackles the bitwidth heterogeneous FL problem by selectively aggregating the weights and hierarchically dequantizing the weights prior to aggregation.
- We demonstrate the efficacy of our ProWD framework by validating it on diverse compositions of bitwidth specifications in local clients against recent FL methods as well as naive remedies.

3.2.2 Related Work

Quantization for federated learning While no existing work considers the problem of federated learning across devices with different innate bitwidths, several works propose to quantize weights or gradients to reduce the communication cost, often referred to as Quantized Parameter Communication (QPC). FedPAQ [126] proposed to send the quantized changes of the weights at each client, instead of the full weights. FedCOMGATE [49] extends this idea by adopting a global learning rate and guiding the update direction to stay close to each other with an accumulated correction vector, to further address data heterogeneity. While we empirically observed that QPC approaches alleviate the performance degeneration of the high-bitwidth models under the BHFL scenario since they do not drastically change the weights as simple averaging does, they are suboptimal since they do not explicitly tackle the challenges posed by the BHFL problem.

Low-bitwidth training Low-bitwidth training is an approach to train a model using only low-bitwidth operations and data types at training time, which enables on-device learning with lightweight edge devices. BNN [56] and XNOR-Net [124] are approaches to accelerate the training of the convolutional neural networks using binary operations in the forward pass with binarized weights and activations, with floating-point operations to compute gradients in the backward pass. Except for that, diverse approaches have been proposed that focus on using 8-bit floating point numbers for the weights, activations, and gradients [190], devising ternary gradients to reduce communication cost (yet, it allows float32 operation for accumulating the gradients) [161], training without floating-point operations [163], adopting direction-sensitive gradient clipping [191], training for mixed-bitwidth models [185, 186, 151]. Most existing neural quantization approaches quantize only the weights or the gradients while preserving parts of the network that has a large impact on the performance, such as activations, in full-precision, and thus are not applicable to training on devices with limited bitwidths.

Neural dequantization The dequantization of low-bitwidth signals into high-bitwidth signals has been studied for diverse applications. For image reconstruction, [168] aims to recover the resolution of quantized sRGB images to full-dynamic-range RAW image data via an invertible dequantizer function. In generative flow models, [114] dequantizes the discrete-valued data by adding a uniform noise to guarantee that the data is able to have any value in the continuous domain. Dequantizatoin is also used for the process of converting the low-bit representation of the weights to high-bit without changing their values [43], or to decode the encoded vectors using a codebook [87]. In this paper, we refer to "dequantization" to describe the process of recovering the original high-bitwidth weights from the low-bitwidth quantized weights received from local clients so that a server can utilize the high-performing recovered models for aggregation during FL. To our knowledge, our work is the first work that provides an appropriate method adopting the weight dequantization approach for solving practical FL scenarios.

3.2.3 Approach

We now introduce the problem setup for standard federated learning and extends it to the Bitwidth Heterogeneous Federated Learning (BHFL) scenario, where a subset of the participating devices train the local models with low-bitwidth operations according to their hardware specifications. We then describe the quantized computational flow for the training of low-bitwidth clients.

Problem statements

Federated learning In a standard Federated Learning (FL) scenario [105, 26], each client trains the local model on the private data and periodically transmits the model parameters to the central server, where the models are aggregated and broadcasted back to the clients. Let N different clients $\mathcal{C} = \{c_1, ..., c_N\}$ participate in an FL system. Given training samples $\boldsymbol{x}_n$ and its corresponding labels $\boldsymbol{y}_n$, we suppose that a client c_n solves a local optimization problem $\mathcal{L}_n = \text{CE}(f(\boldsymbol{x}_n; \boldsymbol{w}_n); \boldsymbol{y}_n)$, where CE is a cross-entropy loss and $f(\cdot; \boldsymbol{w}_n)$ is a neural network of client c_n parameterized by $\boldsymbol{w}_n$. At each communication round r, clients $\mathcal{C}^{(r)} \subseteq \mathcal{C}$ send the model parameters to the central server, and the server aggregates received weights, for example, by averaging their weights.

Bitwidth heterogeneous federated learning The majority of existing FL methods assume full-precision operations for local clients, even when considering device heterogeneity. However, the participating devices have largely heterogeneous bitwidths according to their hardware specifications. To this end, we introduce a practical FL scenario, named Bitwidth Heterogeneous Federated Learning (BHFL), in which we relax the strong assumption that all clients are capable of full-precision floating-point operations. We represent n-th local client as a tuple of the model weights $\boldsymbol{w}_n$ and the corresponding hardware bitwidth information $s_n \in \mathcal{S}$, where $\mathcal{S}$ is a set of available bitwidth specifications for local hardware devices. Then, the set of N clients, $\mathcal{C}$ can be represented as follows: $\mathcal{C} = \{(\boldsymbol{w}_1, s_1), \ldots, (\boldsymbol{w}_N, s_N)\}$. At each round of communication, a central server receives client tuples and aggregates model parameters which might be quantized in various levels depending on hardware specifications. We assume that the server allows the full-precision computation, which redistributes $\boldsymbol{w}_G^{(r)}$ to all clients after quantizing the aggregated weights according to the hardware specifications for each client; that is, $\boldsymbol{w}_n^{(r+1)} \leftarrow Q(\boldsymbol{w}_G^{(r)}, s_n), \forall n$.

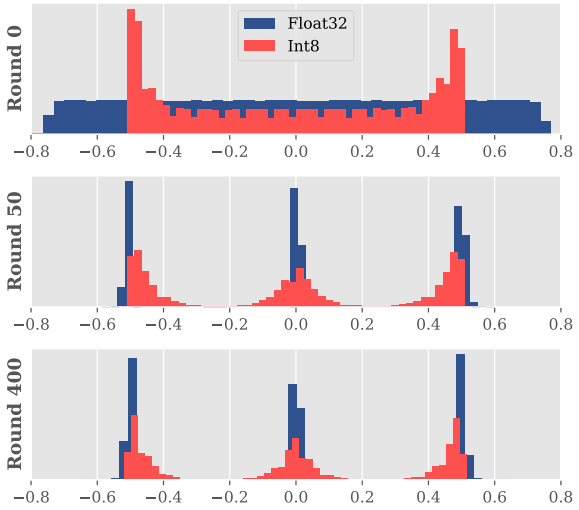


Figure 3.12: **Skewed weight distribution after the aggregation of mixed bitwidth weights**. The distribution of the last layer's weights of the full-precision and low-bitwidth models at the initial, after 50, and 400 aggregation rounds.

In BHFL, what aggregation method we use could strongly impact the overall performance of the model being learned. A simple averaging technique to aggregate local clients' model parameters hinders convergence. This is because the model falls into a suboptimal local minimum due to the incompatibility across the model weights with heterogeneous bitwidths due to large discrepancies in their distributions. The detrimental effect is more severe for larger bitwidth models, as aggregating the low-bitwidth model weights will result in the loss of expressiveness in the model. In Figure 3.12, we observe that the distribution of the full-precision weights quickly degenerates into three peaks that correspond to the ternary quantization values of the low-bitwidth models.

Local computation for limited bitwidth clients

We assume that the local clients are edge devices that perform limited bitwidth computations according to their hardware specifications. Float32 clients perform regular full-precision training, but quantized clients, such as Int6, Int8, and Int16, perform training using low-precision integer operations, following [163]. Let $\boldsymbol{q}^l$ and $\boldsymbol{a}^l$ be s-bit quantized weights and activations at layer l for a client, respectively. We denote the convolution operator as $*$. Since convolution involves multiplication between the weights and the activations, naively computing $\boldsymbol{q}^l * \boldsymbol{a}^{l-1}$ requires the hardware to support fast multiplication of two s-bit integers. To relax this requirement, we ternarize the quantized weights before the convolution operation:

$$\boldsymbol{a}^l = \mathrm{ReLU}\left(Q_s\left(Q_{\mathrm{Int2}}\left(\boldsymbol{q}^l\right) * \boldsymbol{a}^{l-1}/\alpha^l\right)\right). \tag{3.3}$$

Thus, Equation 3.3 only needs s-bit addition and subtraction operations for training and inference, and the intermediate results can be stored in $(s+1)$-bits while achieving a good tradeoff between precision and computation [92]. To prevent the activations from clipping out of range, we rescale the activations with a pre-defined layerwise scalar coefficient α^l so that they are within the range for s-bit integers. This rescaling can be implemented efficiently with a hardware bit-shift operation. After a few local training steps, limited-bitwidth clients broadcast original low-bitwidth weights $\boldsymbol{q}$ or the ternarized weights $Q_{\mathrm{Int2}}\left(\boldsymbol{q}\right)$ to the server, like other QPC methods [126, 49].

Low-bitwidth training for bit-limited hardware devices

Weight initialization In order to prevent weights from vanishing due to the intermediate ternarization, we adjust the scale of the initialization as follows:

$$\boldsymbol{w}_q \sim \mathcal{U}(-L, L), \tag{3.4}$$

where $L = \max\{0.75, \sqrt{3/\mathrm{fan\_in}_l}\}$. Note that when $L = \sqrt{3/\mathrm{fan\_in}_l}$, it is equivalent to the Kaiming uniform initialization. The layer-wise scaling factor α^l is defined as follows:

$$\alpha^l = \mathrm{Shift}(0.75/\sqrt{3/\mathrm{fan\_in}_l}). \tag{3.5}$$

This modified initialization strategy has consequences for the full-precision model since the scale of the weights aggregated is not the same across bitwidths. We alleviate this problem by initializing the full-precision model weights with the same distribution as the quantized model weights and using Weight Normalization [133] in the full-precision clients to compensate for the scale difference.

Backward pass The errors are calculated by using the chain rule, except we normalize the errors at each layer to prevent saturation. Specifically, we apply the following before propagating the error value to every subsequent layer in the chain rule:

$$\boldsymbol{e}_q = Q_s(\boldsymbol{e}/\mathrm{Shift}(\max\{|\boldsymbol{e}|\})), \tag{3.6}$$

where $\mathrm{Shift}(x) = 2^{\lceil \log_2 x \rfloor}$ finds the nearest power-of-two to the input, and $\max\{|\boldsymbol{e}|\})$ represents the layer-wise maximum absolute value among the elements of the error $\boldsymbol{e}$, and s is the bitwidth of the client.

The quantizer function is defined as

$$Q_s(x) := \text{clip}_s(\lceil (2^{s-1} \cdot x) \rfloor / 2^{s-1}), \tag{3.7}$$

$$\text{clip}_s(x) := \max(\min(x, (2^{s-1} - 1)/2^{s-1}), (-2^{s-1} + 1)/2^{s-1}). \tag{3.8}$$

For the weight update, we similarly apply the following rescaling operation to the gradient value before applying the weight update:

$$\boldsymbol{q} \leftarrow \text{clip}_s(\boldsymbol{q} - Q_s^{\text{stoch}}(\eta \cdot \boldsymbol{g} / \text{Shift}(max\{|\boldsymbol{g}|\}))), \tag{3.9}$$

where $Q_{s_n}^{\text{stoch}}(\cdot)$ is a stochastic quantization function defined elementwise as follows:

$$Q_s^{\text{stoch}}(x) = \begin{cases} 2^{1-s} \cdot \lceil |x| \rceil & \text{w.p. } |x| - \lfloor |x| \rfloor \\ 2^{1-s} \cdot \lfloor |x| \rfloor & \text{otherwise.} \end{cases} \tag{3.10}$$

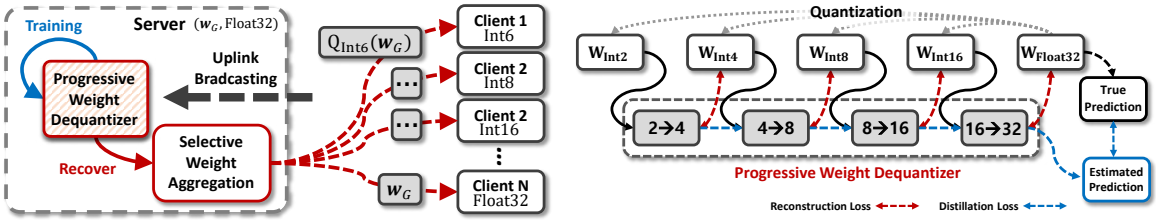
Note that the scale of the learning rate η is different from regular full-precision network training because of the rescaling of the gradient values.

Bitwidth heterogeneous federated learning with ProWD

We first introduce our naive remedies for alleviating the challenges posed by the BHFL problem, and their limitations. Then, we propose a novel FL framework to properly handle BHFL scenarios, named *ProWD*, which consists of two core components: progressive weight dequantization and score-based selective weight aggregation.

Naive remedies for bitwidth heterogeneity in federated learning BHFL deteriorates the performance of higher-bitwidth clients' models, while the local models from low-bitwidth clients often enjoy clear benefits due to knowledge transfer from the high-bitwidth models (See Figure 3.12). We first suggest a bitwidth-dependent averaging technique for BHFL, preventing interference across different bitwidth weights during aggregation, referred to as FedGroupedAvg. However, this method suffers from poor transferability, as the rich knowledge obtained by expressive high-bitwidth models is not transferred to low-bitwidth ones, while it preserves the performance of higher-bitwidth clients. Thus, we additionally suggest a modified version of FedGroupedAvg, which allows the knowledge transfer from higher-bitwidth clients to lower ones, but not vice versa, named as FedGroupedAvg-Asymmetric. These two simple baselines can improve the performance of the local model but are suboptimal in that it only utilizes part of the knowledge from participating models. Thus, we propose a novel method to effectively tackle the challenges in BHFL, which overcomes the limitations of such naive remedies.

Progressive weight dequantization Low-bitwidth models, while hardware-friendly, severely limit the expressiveness of the model. Aggregating the parameters with such limited information thus may degrade the quality of high-bitwidth models during federated learning. To this end, we propose a trainable dequantizer that reconstructs the full-precision weights for given low-precision weights. Yet, directly reconstructing the low-bit weights to the high-bit may be less effective, especially when the bitwidth gap is large since there exists a significant disparity between their distributions. To alleviate this issue during single-step reconstruction, our dequantizer breaks down the problem into block-wise weight dequantization problems by adopting a stack of network blocks.



(a) Bitwidth heterogenrous federated learning with ProWD (b) An example of progressive weight dequantizer

Figure 3.13: **(a) Illustration of our ProWD framework.** Clients send their local models and hardware bitwidth specifications to the server. We reduce the distributional disparity among weights from different bitwidth devices during BHFL by introducing weight dequantization and selective aggregation. **(b) Progressive weight dequantizer** recovers low-bit weights into the high-bit via minimizing two loss terms.

Let a $\Pi = \{\pi_0, \cdots, \pi_k\} \supseteq \mathcal{S}$ be an ordered set of bitwidth precisions, where π_j is a lower precision bitwidth than π_{j+1} $\forall j$. The lowest and the higest bitwidths of Π correspond to those of $\mathcal{S}$, i.e., $\pi_0 = s_1$ and $\pi_k = s_m$. We denote a progressive weight dequantization function ϕ as a stack of k decomposable neural network blocks:

$$\phi := \phi^{\pi_0 \to \pi_k} = \phi^{\pi_0 \to \pi_1} \circ \phi^{\pi_1 \to \pi_2} \circ \cdots \circ \phi^{\pi_{k-1} \to \pi_k}, \tag{3.11}$$

where $\circ$ denotes the function composition and $\phi^{\pi_j \to \pi_{j+1}}$ indicates a block that recovers π_{j+1}-bit weights from π_j-bit weights. That is, we design the set of bitwidths for dequantizer blocks Π to include the client bitwidths so that the dequantizer can directly reconstruct the desired bitwidths from the received low-bit weights. We implement each block using a dimensionality-preserving function $h(\cdot; \boldsymbol{\theta})$ with a residual connection to its input. When the model $(\boldsymbol{w}, \pi_{j>0})$ arrives from a local client, the server quantizes $\boldsymbol{w}$ into j lower-bitwidth weights $\mathcal{Q}_{\boldsymbol{w}} = \{\boldsymbol{q}_{\pi_0}, ..., \boldsymbol{q}_{\pi_{j-1}}\}$. Given $\pi_{j<k}$-bit quantized weights $\boldsymbol{q}_{\pi_j}$, the block operation is formulated as follows:

$$\widehat{\boldsymbol{q}}_{\pi_{j+1}} = \phi^{\pi_j}(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j) = \boldsymbol{q}_{\pi_j} + h(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j). \tag{3.12}$$

The central server constructs the weight dataset out of received local model weights by segmenting them into uniformly-sized blocks (e.g., $64 \times 24 \times 24$). Thus, our dequantizer can utilize differently-shaped weights from different layers or across heterogeneous neural architectures. For the training of the weight dequantizer, we introduce two different loss terms. The first term is the reconstruction loss, which is defined as the average difference of the blockwise weight reconstruction of quantized input weights $\boldsymbol{q}_{\pi_j}$ and its higher bitwidth ground truth weights $\boldsymbol{q}_{\pi_{j+1}}$:

$$\mathcal{L}_{recon} = \sum_{j=0}^{k-1} \left\| \boldsymbol{q}_{\pi_{j+1}} - \phi^{\pi_j \to \pi_{j+1}}(\boldsymbol{q}_{\pi_j}; \boldsymbol{\theta}_j) \right\|_1. \tag{3.13}$$

Minimizing the blockwise reconstruction error allows the model to progressively extrapolate missing links between the low-bitwidth weights and high-bitwidth weights and not to stray far from the intermediate reconstructions. We note that there is no strict rule for the choices of the target bitwidths of dequantizer blocks, but we design the dequantizer in which the bitwidth difference between consecutive blocks is maintained to a similar degree until reaching the target high-bitwidth, avoiding the drastic increase in bitwidth for each reconstruction step.

The second term is a distillation loss for minimizing the discrepancy in the predictions from the

model with high-bitwidth ground truth weights and the model with dequantized low-bitwidth weights. To this end, the central server utilizes a tiny buffer $\mathcal{U}$ independent of the local data to compare the prediction between the high-bit model and the recovered model from the low-bit weights. Given the $\boldsymbol{w}$ and its quantized low-precision $\boldsymbol{q}_{\pi_0} = Q_{\pi_0}(\boldsymbol{w})$, we compute a distillation loss using a randomly sampled minibatch $\boldsymbol{u} \sim \mathcal{U}$ from the buffer as follows:

$$\mathcal{L}_{distill} = -\text{Sim}\Big(f(\boldsymbol{u}; \boldsymbol{w}), f\big(\boldsymbol{u}; \phi^{0\to k}(\boldsymbol{q}_{\pi_0}; \Theta)\big)\Big), \tag{3.14}$$

where $\text{Sim}(\cdot)$ is a similarity metric for the two output distributions (we use cosine similarity) and $\Theta = \{\boldsymbol{\theta}_0, \cdots, \boldsymbol{\theta}_k\}$. Note that $f(\boldsymbol{u}; \boldsymbol{w})$ is a softmax class probability of a neural network parameterized by $\boldsymbol{w}$ given input $\boldsymbol{u}$. This distillation loss allows ProWD to directly tackle the prediction task with the reconstructed weights, which helps it recover full-precision weights while considering the downstream task performance. The final objective for training ProWD at the server is given as follows:

$$\mathcal{L} = \mathcal{L}_{recon} + \lambda \mathcal{L}_{distill}, \tag{3.15}$$

where λ is a hyperparameter to balance the two loss terms. Note that the model is not very sensitive to the choice of λ, and thus we set $\lambda = 1$ in all our experiments. The training steps of our dequantizer are described in Algorithm 7.

Algorithm 7 Training of progressive weight dequantizer

input set of bitwidths $\Pi = \{\pi_0, \ldots, \pi_k\}$, dequantizer $\phi_\Theta^{\pi_0 \to \pi_k}$, input weights and corresponding bitwidth $(\boldsymbol{w}, \pi_j)$, neural network f, unsupervised buffer $\mathcal{U}$, balancing coefficient λ.

$\boldsymbol{q}_{i<j} = \{Q_{\pi_i}(\boldsymbol{w})\}_{i=0}^{j-1}$ ▷ Quantize into each of the lower bitwidths
Construct data loader $\mathcal{D}_{\boldsymbol{w}}$ using $\{\boldsymbol{q}_{i<j}, \boldsymbol{w}\}$
for $(\boldsymbol{q}_0, \ldots, \boldsymbol{q}_{j-1},\ \boldsymbol{w}) \sim \mathcal{D}_{\boldsymbol{w}}$ **do**
  Sample $\boldsymbol{u} \sim \mathcal{U}$ augmented with gaussian noise
  $\mathcal{L}_{recon} = \sum_{i=0}^{j-1} \left\| \boldsymbol{q}_{i+1} - \phi^{\pi_i \to \pi_{i+1}}(\boldsymbol{q}_i; \boldsymbol{\theta}_i) \right\|_1$
  $\mathcal{L}_{distill} = -\text{Sim}(f(\boldsymbol{u}; \boldsymbol{w}), f(\boldsymbol{u}; \phi_\Theta(\boldsymbol{q}_0)))$
  $\mathcal{L} = \mathcal{L}_{recon} + \lambda \mathcal{L}_{distill}$
  Update weight dequantizer ϕ_Θ to minimize $\mathcal{L}$
end for

Training details of progressive weight dequantizer

Construction of the weights dataset Given the local model weights $\boldsymbol{w}$, we construct the weight datasets to learn the progressive weight dequantizer. Since layers in deep neural networks are often composed of weights with different dimensionality each other, we split them into uniformly-sized sub-weights. As the following common structures for CNN models are mainly composed of multiple convolution layers, such as VGG [144] and ResNet [55], we utilize convolution weights with a filter size of 3×3, and the input dimension is 64 or larger to construct the weight dataset. That is, we use all

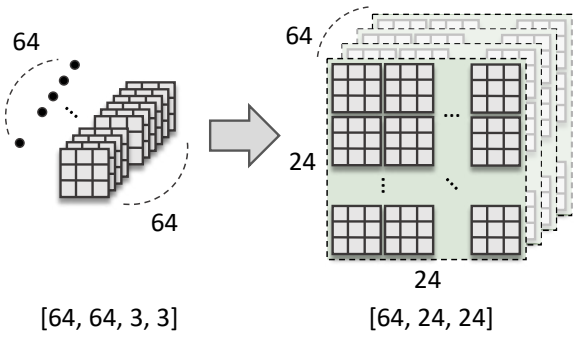


Figure 3.14: **Illustration of a reshaping process on the weight module.**

convolution weights except the weights from the first layer (the input dimension is the image channel, 3). We split the weights at each convolution layer into partial modules with the shape of $64 \times 64 \times 3 \times 3$ (e.g., the weights with the shape of $256 \times 128 \times 3 \times 3$ are split into $4 \cdot 2 = 8$ different modules. Next, we reshape each module sized by 24×24 with 64 channels (i.e., $64 \times 24 \times 24$) as illustrated in Figure 3.14.

Block design for progressive weight dequantizer We implement our dimensionality-preserving network block ϕ for our dequantizer using two layered affine coupling layers, and the design of each layer ρ is as follows:

$$\begin{aligned} \widehat{\boldsymbol{w}}_{1:d} &= \boldsymbol{w}_{1:d} + \alpha(\boldsymbol{w}_{d+1:D}), \\ \widehat{\boldsymbol{w}}_{d+1:D} &= \boldsymbol{w}_{d+1:D} \odot \exp\left(\beta(\widehat{\boldsymbol{w}}_{1:d})\right) + \gamma(\widehat{\boldsymbol{w}}_{1:d}), \end{aligned} \tag{3.16}$$

where $\widehat{\boldsymbol{w}} = \rho(\boldsymbol{w})$, α, β, and γ are DenseNet [59] blocks while we omit the notation of weights in each layer for readability. To this end, a j^{th} network block of progressive weight dequantizer is formulated as follows:

$$\widehat{\boldsymbol{w}} = \phi^{\pi_j \to \pi_{j+1}}(\boldsymbol{w}; \boldsymbol{\theta}_j) = \boldsymbol{w} + \tau\rho(\rho(\widehat{\boldsymbol{w}})), \tag{3.17}$$

where τ is a scaling coefficient and we set $\tau = 0.1$ for all experiments. We want to note that there is a huge potential to further develop the design of our dequantizer. We intend to suggest a better design for the dequantization function for recovering high-bitwidth weights from low-bitwidth weights in future work.

Score-based selective weight aggregation

As described in Figure 3.12, using a naive aggregation technique, such as simple averaging, may lead the model training to fall into a suboptimal local minimum. To prevent such distribution shifts of weights during BHFL, we assert that the low-bitwidth models should share similar gradient directions as the full-precision model, as inspired by the observation in [191] that there exists a strong correlation between the training stability and the deviation of the quantized model's gradient directions from the full-precision model, measured by the cosine similarity. Let us consider a simple two-client federated learning framework where a server communicates with a full-precision model parameterized with $\boldsymbol{w}_{\text{High}}$ and a low-bitwidth model parameterized with $\boldsymbol{w}_{\text{Low}}$. The goal of the low-bitwidth model then is to distill the knowledge of $\boldsymbol{w}_{\text{High}}$:

$$\left\langle \frac{\partial \ell(f(B; \boldsymbol{w}_{\text{High}}))}{\partial \boldsymbol{w}_{\text{High}}}, \frac{\partial_Q \ell(f_Q(B; \boldsymbol{w}_{\text{Low}}))}{\partial_Q \boldsymbol{w}_{\text{Low}}} \right\rangle \geq 0, \tag{3.18}$$

where ℓ is a task loss and $f(\cdot; \boldsymbol{w})$ is a neural network parameterized by $\boldsymbol{w}$. The subscript Q denotes the quantized operations on the weights, activations, and gradients.

However, unlike the setting of [191], under BHFL scenarios, it is impossible to preserve full-precision knowledge on low-bitwidth local devices as they have no means to represent them. Thus, we impose a selective weight aggregation technique based on the relevancy among the weights from the local clients. When a central server receives the local models from different bitwidth clients, we select sparse sub-weights from lower-bitwidth models that are compatible with the full-precision aggregated weights.

Let $\overline{\boldsymbol{w}}^{(r)}_{\text{High}}$ and $\overline{\boldsymbol{w}}^{(r)}_{\text{Low}}$ denote average high-bit weights and the low-bit weights that the server received at communication round r, respectively. When $\Delta\overline{\boldsymbol{w}}_{\text{High}} = \overline{\boldsymbol{w}}^{(r)}_{\text{High}} - \overline{\boldsymbol{w}}^{(r-1)}_{\text{High}}$ and $\Delta\overline{\boldsymbol{w}}_{\text{Low}} = \overline{\boldsymbol{w}}^{(r)}_{\text{Low}} - \overline{\boldsymbol{w}}^{(r-1)}_{Low}$,

we encourage the high-bit and low-bit model weights to have similar update directions by disregarding a few outliers in the low-bit models. That is, Given a sparsity ratio τ, we encourage a server to obtain the binary mask c^* as follows:

$$\boldsymbol{c}^* = \underset{\boldsymbol{c}}{\operatorname{argmax}} \frac{(\boldsymbol{c} \odot \Delta\overline{\boldsymbol{w}}_{\text{Low}})^\top \Delta\overline{\boldsymbol{w}}_{\text{High}}}{\|\boldsymbol{c} \odot \Delta\overline{\boldsymbol{w}}_{\text{Low}}\| \|\Delta\overline{\boldsymbol{w}}_{\text{High}}\|}, \text{ s.t. } |\boldsymbol{c}^*| \leq \tau. \tag{3.19}$$

For BHFL scenarios with multiple bitwidths, we split high-/low-bitwidths weights based on the mean bit-width of the given bitwidth set. We formulate the equation as a simple optimization problem to obtain a desired binary mask c^* to maximize the cosine similarity between sparsified averaged weight movement of low-bitwidth models and the averaged movement of the high-bitwidth models. The process is rapidly optimized within a few steps (e.g., 10) and a marginal training time ($\sim 10ms$ per client). To this end, we perform selective weight aggregation as follows:

$$\boldsymbol{w}_G \leftarrow \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{c}_n \odot \boldsymbol{w}_n, \tag{3.20}$$

where $\boldsymbol{c}_n = \boldsymbol{c}^*$, if s_n is one of low-bitwidth specifications, otherwise, $\boldsymbol{c}_n$ is a all-one tensor with the same shape as $\boldsymbol{w}_n$. The overall procedure of our BHFL framework is described in Algorithm 8. Note that the dequantizer can be updated anytime during the FL process in a concurrent manner because it does not require the latest client model weights for training. Thus, there is no training time bottleneck introduced by the training of our dequantizer.

Algorithm 8 ProWD framework for BHFL

input clients $\mathcal{C} \leftarrow \{\boldsymbol{w}_n, s_n\}_{n=1}^N$, $s_j \in \Pi = \{\pi_0, \ldots, \pi_k\}$, $\forall j$, weight dequantizer $\phi_\Theta^{\pi_0 \rightarrow \pi_k}$, global weights $\boldsymbol{w}_G$.

for each round $r = 1, 2, ..., R$ **do**
  Sample $\mathcal{C}^{(r)} \subseteq \mathcal{C}$ where $|\mathcal{C}^{(r)}| = M$
  Distribute federated weights $\boldsymbol{w}_G$ to clients $\mathcal{C}^{(r)}$
  for each client $(\boldsymbol{w}_n, s_n) \in \mathcal{C}^{(r)}$ **in parallel do**
    $\boldsymbol{w}_n \leftarrow Q_{s_n}(\boldsymbol{w}_G)$ ▷ send s_n-bit quantized weights
    $\boldsymbol{w}_n \leftarrow \text{LocalUpdate}(\boldsymbol{w}_n, s_n)$
    Broadcast $(\boldsymbol{w}_n, s_n)$ to the central server
  end for
  $\boldsymbol{w}_n \leftarrow \phi_\Theta(\boldsymbol{w}_n)$, **if** $s_n \neq \pi_k$ ▷ Dequantize weights
  Obtain binary masks $\boldsymbol{c}^*$ using Equation 3.19
  $\boldsymbol{w}_G \leftarrow \frac{1}{M} \sum_{n=1}^{M} \boldsymbol{c}_n \odot \boldsymbol{w}_n$ ▷ Equation 3.20
end for

3.2.4 Experimental Results

Datasets

We validate our method against the relevant FL methods under several BHFL scenarios, with varying bitwidth configurations of participating clients. We use the widely used benchmark dataset for federated learning methods, CIFAR-10 to validate our method following the IID experimental settings of the existing works [126, 49]. CIFAR-10 is an image classification dataset that consists of 10 object classes, each of which has 5,000 training instances and 1,000 test instances. For FL purposes, we uniformly split the training instances per class by the number of clients participating in the federated learning system. We use a modified version of VGG-7 network [144] for all our experiments.

Table 3.7: Average accuracy at each bitwidth and average accueacy across all clients on CIFAR-10 dataset. We set participating clients with 50% of Int8 and 50% of Float32 models (Left), and 50% of Int8 and 50% of Float32 models (Right). All of the results are measured by computing the 95% confidence interval over three independent runs.

Method	CIFAR-10, Int8 (50%) - Float32 (50%)				CIFAR-10, Int8 (80%) - Float32 (20%)			
	Int8 acc	Float32 acc	Gap	Average	Int8 acc	Float32 acc	Gap	Average
Local Training	69.59 (± 0.34)	75.82 (± 0.41)	+6.23	72.71 (± 0.26)	69.39 (± 0.34)	76.41 (± 0.52)	+7.02	70.79 (± 0.34)
FedAvg [105]	76.88 (± 0.49)	76.23 (± 0.36)	-0.65	76.56 (± 0.36)	77.43 (± 0.83)	74.64 (± 1.21)	-2.79	76.87 (± 0.87)
FedProx [94]	71.16 (± 0.35)	69.28 (± 0.90)	-1.88	70.22 (± 0.55)	69.60 (± 0.50)	66.28 (± 1.24)	-3.32	68.94 (± 0.57)
FedPAQ [126]	78.01 (± 0.55)	84.93 (± 0.30)	+6.93	81.47 (± 0.29)	76.61 (± 0.60)	82.27 (± 0.42)	+5.66	77.74 (± 0.49)
FedCOM [49]	75.37 (± 0.52)	77.69 (± 0.45)	+2.32	76.53 (± 0.38)	73.60 (± 1.08)	80.73 (± 0.73)	+7.12	75.03 (± 0.94)
FedCOMGATE [49]	76.74 (± 0.59)	77.36 (± 0.55)	+0.62	77.05 (± 0.36)	74.48 (± 0.63)	81.18 (± 0.56)	+6.70	75.82 (± 0.53)
FedGroupedAvg	61.85 (± 0.78)	85.08 (± 0.28)	+23.23	73.46 (± 0.29)	71.76 (± 0.68)	78.07 (± 0.58)	+6.31	73.02 (± 0.65)
FedGroupedAvg-Asymmetric	78.39 (± 0.54)	84.97 (± 0.27)	+6.57	81.68 (± 0.26)	70.14 (± 0.36)	78.70 (± 0.48)	+8.56	71.85 (± 0.36)
ProWD (Ours)	**82.87** (± 0.37)	**85.99** (± 0.20)	+3.12	**84.43** (± 0.22)	**79.23** (± 0.25)	81.26 (± 0.45)	+2.03	**79.63** (± 0.23)

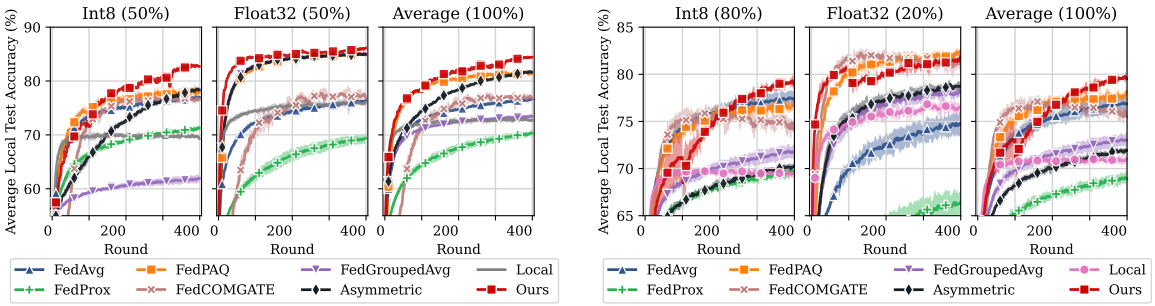


Figure 3.15: Visualization of average test accuracy on CIFAR-10 with 10 clients where the bitwidth of the clients is composed of **(a) 50% of Int8 and 50% of Float32** and **(b) 80% of Int8 and 20% of Float32** during BHFL. We average the results over three independent runs.

Baselines

We compare our method *BiTAT* against following FL baselines: **FedAvg** [105]: A popular federated learning method that performs simple averaging of the local models at the server at each round.
FedProx [94]: A federated learning method that aims to deal with device heterogeneity, with additional ℓ_2 distance regularization terms during the update of local models to prevent the model divergence.
FedPAQ [126]: A quantized parameter communication (QPC)-based FL method, which at each round quantizes the difference between the current local weights and the last aggregated weights at each client, then sends it to the server for aggregation.
FedCOM [49]: An extension of FedPAQ with a learnable global learning rate, which achieves faster convergence in data-homogeneous settings.
FedCOMGATE [49]: An extension of FedCOM to data-heterogeneous settings, which uses a correction vector that constrains the local models to evolve in a similar direction.
FedGroupedAvg: A variant of FedAvg in which the server separately aggregates weights only from the same bitwidth clients, and redistributes them to corresponding local clients.
FedGroupedAvg-Asymmetric: A modified version of FedGroupedAvg that sends clients only the aggregated weights of other clients that has the same bitwidth or higher.

Table 3.8: Average accuracy at each bitwidth, and across all clients on CIFAR-10. We set clients with 30% of Int6, 30% of Int8, 20% of Int12, and 20% of Int16 models. All the results are measured by computing and standard deviation over three independent runs.

Method	int6	int8	int12	int16	gap	AvgAcc
Local Training	69.51	69.25	68.96	70.40	+0.89	69.50
FedAvg	80.17	85.71	**86.49**	87.02	+6.85	84.47
FedPAQ	80.78	85.30	85.98	86.93	+6.15	84.40
FedCom	39.47	55.14	53.07	53.26	+13.79	49.65
FedComGate	56.74	68.43	67.63	68.14	+11.69	64.70
FedGroupedAvg	79.27	80.62	76.94	77.39	-1.88	78.83
FedGroup-Asym	80.25	79.74	79.18	77.73	-2.52	79.38
ProWD (Ours)	**82.89**	**86.11**	**86.47**	**87.51**	+4.62	**85.50**

Table 3.9: Ablation study for ProWD. Deq and SWA refer to the progressive weight dequantizer and selective weight aggregation, respectively. We report the results over three independent runs.

Method	**CIFAR-10**, int8 (50%) - Float32 (50%)				
	Deq	SWA	int8 acc	float32 acc	average
FedAvg	–	–	76.55 (± 0.44)	75.71 (± 0.32)	75.83 (± 0.41)
+Deq	✓	–	79.60 (± 0.39)	80.38 (± 0.44)	79.99 (± 0.26)
+SWA	–	✓	79.79 (± 0.98)	**85.94** (± 0.92)	82.84 (± 0.55)
Ours	✓	✓	**82.87** (± 0.37)	**85.99** (± 0.20)	**84.43** (± 0.22)
Method	**CIFAR-10**, int8 (80%) - Float32 (20%)				
	Deq	SWA	int8 acc	float32 acc	average
FedAvg	–	–	77.43 (± 0.83)	74.64 (± 1.21)	76.87 (± 0.87)
+Deq	✓	–	78.52 (± 0.50)	76.08 (± 0.44)	79.03 (± 0.31)
+SWA	–	✓	78.42 (± 0.47)	80.82 (± 0.31)	78.90 (± 0.37)
Ours	✓	✓	**79.23** (± 0.25)	**81.26** (± 0.45)	**79.63** (± 0.23)

Quantitative evaluation

We validate our methods under multiple BHFL scenarios with heterogeneous proportions of bitwidths among the clients. We first report the experimental results with 50% of Int8 and 50% Float32 clients (Left) and 80% of Int8 and 20% Float32 clients (Right) in Table 3.7. Int8 models in FedAvg obtain superior performance to local training, where each client trains independently on its local task due to positive knowledge transfer from the full-precision weights. This is also evident in the poor performance of Int8 clients in FedGroupedAvg which demonstrates that FL only with low-bitwidth clients is highly limited due to the lack of information the low-bit weights provide. QPC-based FL methods, FedPAQ, FedCOM, and FedCOMGATE, communicate the quantized form of accumulated gradients at each round, and the server adds the accumulated gradients to the global model at each round, before broadcasting it to the clients. Such gradient communication does not degrade the local task information during FL, which is helpful for Float32 models to mitigate the interference from the low-bit weights. However, the low-bit clients (e.g., Int8) cannot directly amalgamate the expressive knowledge from the high-bitwidth weights, easily falling into suboptimal local minima. FedGroupedAvg variants, while preserving the performance of the high-bit models, they often obtain inferior performance as they do not exploit the full knowledge provided by other models. On the other hand, our ProWD consistently outperforms all baselines with varying compositions of bitwidths, yielding small performance gap between low- and high-bitwidths, which demonstrates the effectiveness of the proposed progressive weight dequantization scheme with selective weight aggregation. The convergence plot in Figure 3.15 shows that our method rapidly converges to good performance while baselines converge to suboptimal local minima.

We further demonstrate the versatility of the ProWD framework under a BHFL scenario with multiple bitwidths in Table 3.8, using devices with diverse bitwidths. Note that we allow local clients to send the parameters with local bitwidth for this experiment, rather than ternarizing them for uplink communication. Int6 and Int8 are considered as low-bitwidths and Int12 and Int16 are considered as high-bitwidths. Our ProWD consistently outperforms baselines with any bitwidths while achieving a small accuracy gap between Int6 and Int16 clients. We expect that the reduced performance gap between ours and FedAvg is due the smaller disparity between weight distributions compared to those in Table 3.7 (Int8↔Float32), since all clients perform low-bit operations with slightly different bitwidths, in

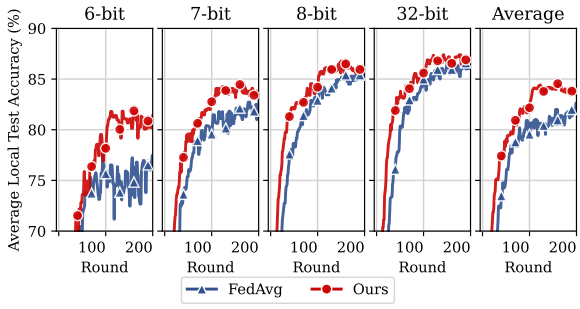


Figure 3.16: Performance comparison between FedAvg and Ours under the Bitwidth Heterogeneous Federated Learning setting with Int6, Int7, Int8, and Float32 clients.

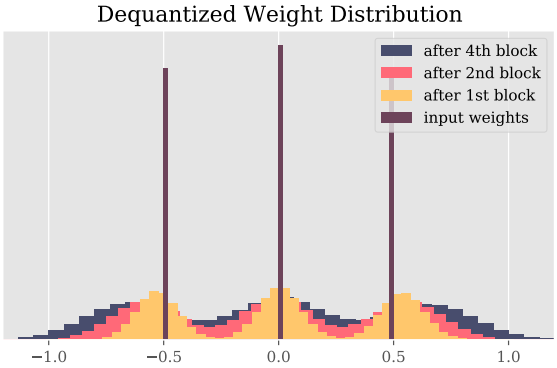


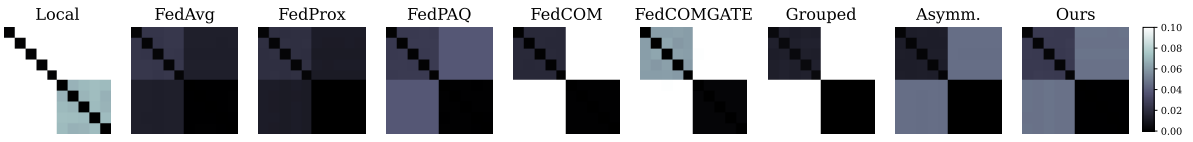
Figure 3.17: **The Weight Distribution after Progressive Dequantization.** Visualization of the distribution after the reconstruction of low-bitwidth model weights. We visualize the last Convolution layer weights in the neural network, trained on CIFAR-10.

which case FedAvg may suffer less from distributional shift at aggregation. To validate that, we further provide additional experiments with larger variance among the bitwidths of the participating devices in Figure 3.16, which shows a significant performance gain of our ProWD to FedAvg (Int6 clients acc: **5.9**%↑, average acc: **2.7**%↑, performance gap: **36**%↓).

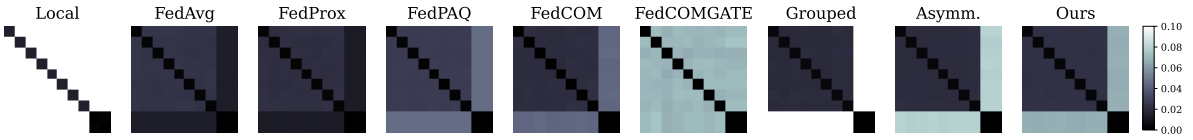
Ablation study Now we explicate the effect of each ingredient of our ProWD on the CIFAR-10 with an ablation study. We experiment on both the uniform (50% of Float32 and 50% of Int8 clients) and low-bitwidth dominant scenario (20% of Float32 and 80% of Int8 clients). As shown in Table 3.9, thanks to its ability to reconstruct full-precision weights from low-bitwidth weights, our weight dequantizer (Deq) improves the performance by 5.5% and 2.8% over that of FedAvg, respectively for each scenario. Also, our selective weight aggregation (SWA) minimizes the discrepancy across the model updates from different bitwidth clients, obtaining the significant performance gain of 9.2% and 2.6% for each scenario. This experimental result confirms the efficacy of both components, progressive weight decomposition, and selective weight aggregation.

Qualitative analysis

The effect of progressive weight dequantization To further analyze the role of the progressive weight dequantizer in our method, we visualize stepwise distributions of reconstructed weights using our dequantizer. For ease of interpretation, we use the initial input as the quantized weight from the last convolution layer in the Int8 client for training on CIFAR-10, where the result is illustrated in Figure 3.17. As low-bitwidth clients transmit ternarized model weights to the central server, the distribution of input weights to the dequantizer is visualized in three peaks. We also visualize the distribution from reconstructed weights after forwarding initial weights up to the first, second, and last block in our dequantizer, colored by *orange*, *pink*, and *navy*, respectively. As we expected, our dequantizer progressively recovers ternary weights towards the high-bitwidths by forwarding them through sequential dequantization blocks.



(a) INT8 (50%) - FLOAT32 (50%)



(b) INT8 (80%) - FLOAT32 (20%)

Figure 3.18: **Cosine distance matrix between the weights of each client.** Elements in a row and column describe the index of local clients. **(a)** the first five clients correspond to Int8 (Top left) and the other five to Float32 (Bottom right). **(b)** the first eight clients correspond to Int8, and the other two to Float32. Darker colors indicate higher similarities between clients.

Weight distance between local clients Next, we visualize the distance of model weights between clients in Figure 3.18 to dissect the difference of learned representations during BHFL. We use the cosine distance, computed by $1 - \cos\_\text{sim}(\boldsymbol{w}_n, \boldsymbol{w}_m)$ between n- and m-th local clients. Each client in the local training (Local) model does not share the knowledge with other clients, resulting in a large weight distance. Weights obtained using FedAvg have smaller distances between Int8 and Float32 models than the distances among the Int8 models, caused by the distributional shift of the high-bitwidth model weights towards the distribution of low-bitwidth model weights (Please see Figure 3.12).

Clients in FedPAQ communicate accumulated local gradients while keeping their local weights, alleviating the detrimental distributional shift of weights to some degree, resulting in a higher similarity among the same bitwidth clients than in bit-heterogeneous cases. Int8 client weights in FedGroupedAvg stay close to each other while straying far from the Float32 client weights, as it only allows communication among clients with the same bitwidth. FedProx shows a similar tendency with FedAvg. Interestingly, the Int8 clients in FedCOMGATE show different behaviors with FedCOM, with the weight distance among them farther than any other BHFL methods. This phenomenon is due to the "correction vectors" utilized in the FedCOMGATE algorithm designed to adapt to data heterogeneity and enable the weights to behave semi-independently, improving the model accuracy compared to FedCOM with a marginal amount. While FedGroupedAvg-Asymmetric (Asymm.) shows a similar distance matrix with FedGroupedAvg (Grouped), it allows the knowledge transfer from-high-to-low-bitwidths clients, performing higher weight similarity between bitwidth heterogeneous clients. Interestingly, ProWD keeps the clients' weights sufficiently different between each bitwidth and obtains sufficiently low proximity across bit-heterogeneous clients, successfully preventing the distributional shift in high-precision weights due to weight averaging. Our proposed method also allows high transferability of the learned knowledge across clients, showing better adaptation of the local tasks over QPC-based methods (Please see Table 3.7 and Figure 3.15).

3.2.5 Summary

We proposed a novel yet practical heterogeneous federated learning scenario where the participating devices have different bitwidth specifications, in which case the model aggregation could have a highly detrimental effect. We further identify the two main causes of the performance degeneration, which are

the suboptimal loss convergence due to distributional shift from aggregation, and the limited expressive power of the low-bitwidth weights. To tackle these challenges, we proposed a novel framework for bit-heterogeneous FL, based on progressive dequantization of the weights and selective weight aggregation. The progressive dequantizer at the server receives weights from low-bitwidth clients and recovers them into higher bitwidth weights by forwarding them through a sequence of dequantizer blocks. Further, selective weight aggregation determines which low-bit weight elements are compatible with higher-bit ones. Empirical evaluations of our framework across two BHFL scenarios with varying degrees of bitwidth-heterogeneity on the benchmark dataset demonstrate the effectiveness of our framework, which largely outperforms relevant FL baselines.

Chapter 4. Realistic Continual Learning with Incomplete and Uncurated Data Stream

4.1 Online Coreset Selection for Rehearsal-based Continual Learning [176]

4.1.1 Motivation

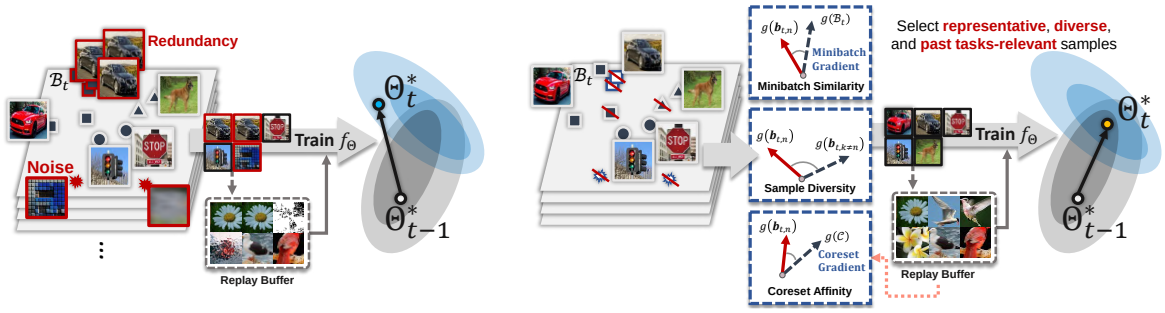
Humans possess the ability to learn a large number of tasks by accumulating knowledge and skills over time. Building a system resembling human learning abilities is a deep-rooted desire since sustainable learning over a long-term period is essential for general artificial intelligence. In light of this need, *continual learning* (CL) [152], or *lifelong learning*, tackles a learning scenario where a model continuously learns over a sequence of tasks [80, 97] within a broad research area, such as classification [74, 19], image generation [182], language learning [96, 9], clinical application [86, 91], speech recognition [132], and federated learning [174]. A well-known challenge for continual learning is *catastrophic forgetting* [104], where the continual learner loses the fidelity for past tasks after adapting the previously learned knowledge to future tasks.

Recent rehearsal-based continual learning methods adapt the continual model to the previous tasks by maintaining and revisiting a small replay buffer [153, 109]. However, the majority of these methods store random-sampled instances as a proxy set to mitigate catastrophic forgetting, limiting their practicality to real-world applications (see Figure 4.1 Left) when all the training instances are not equally useful, as some of them can be more representative or informative for the current task, and others can lead to performance degeneration for previous tasks. Furthermore, these unequal potentials could be more severe under practical scenarios containing *imbalanced, streaming, or noisy instances* (see Figure 4.2). This leads to an essential question in continual learning:

How can we obtain a coreset to promote task adaptation for the current task while minimizing catastrophic forgetting on previously seen tasks?

To address this question, we propose *Online Coreset Selection (OCS)*, a novel method for continual learning that selects representative training instances for the current and previous tasks from arriving streaming data in an online fashion based on our following three selection strategies: **(1)** *Minibatch similarity* selects samples that are representative to the current task $\mathcal{T}_t$. **(2)** *sample diversity* encourages minimal redundancy among the samples of current task $\mathcal{T}_t$. **(3)** *Coreset affinity* promotes minimum interference between the selected samples and knowledge of the previous tasks $\mathcal{T}_k$, $\forall k < t$. To this end, OCS minimizes the catastrophic forgetting of the previous tasks by utilizing the obtained coreset for future training and also encourages the current task adaptation by updating the model parameters on the top-κ selected data instances. The overall concept is illustrated in Figure 4.1 Right.

Our method is simple, intuitive, and generally applicable to any rehearsal-based continual learning method. We evaluate the performance of OCS on various continual learning scenarios and show that it outperforms state-of-the-art rehearsal-based techniques on balanced, imbalanced, and noisy continual learning benchmarks of varying complexity. We also show that OCS is general and exhibits collaborative



(a) Existing rehearsal-based CL (b) Online Coreset Selection (OCS)

Figure 4.1: **Illustration of existing rehearsal-based CL and Online Coreset Selection (OCS)**: **(a)** Existing rehearsal-based methods train on all the arrived instances and memorize a fraction of them in the replay buffer, which results in a suboptimal performance due to the outliers (noisy or biased instances). **(b)** OCS obtains the coreset by leveraging our three selection strategies, which discard the outliers at each iteration. Consequently, the selected examples promote generalization and minimize interference with the previous tasks.

(a) Imbalanced continual learning (b) Noisy continual learning

Figure 4.2: **Realistic continual learning scenarios: (a)** Each task consists of class-imbalanced instances. **(b)** Each task has uninformative noise instances, which hamper training.

learning with the existing rehearsal-based methods, leading to increased task adaptation and inhibiting catastrophic forgetting. To summarize, our contributions are threefold:

- We address the problem of coreset selection for realistic and challenging continual learning scenarios, where the data continuum is composed of class-imbalanced or noisy instances that deteriorate the performance of the continual learner during training.
- We propose *Online Coreset Selection (OCS)*, a simple yet effective online coreset selection method to obtain a representative and diverse subset that has a high affinity to the previous tasks from each minibatch during continual learning. Specifically, we present three gradient-based selection criteria to select the coreset for current task adaptation while mitigating catastrophic forgetting.
- We demonstrate that OCS is applicable to any rehearsal-based continual learning method and experimentally validate it on multiple benchmark scenarios, where it largely improves the performance of the base algorithms across various performance metrics.

4.1.2 Related Work

Continual learning. In the past few years, there has been significant progress in continual learning to alleviate catastrophic forgetting [104]. The *regularization approaches* [74, 90, 137] modify the model parameters with additional regularization constraints to prevent catastrophic forgetting. The *architecture approaches* [130, 178, 170, 95, 175] utilize network isolation or expansion during continual

learning to improve network performance. Another line of research uses *rehearsal approaches*, which memorize or generate a small fraction of data points for previous tasks and utilize them to retain the task knowledge [102, 19, 4, 11]. For example, Gradient-based Sample Selection (GSS) [4] formulates the selection of the replay buffer as a constraint selection problem to maximize the variance of gradient direction. ER-MIR [3] iteratively constructs the replay buffer using a loss-based criterion, where the model selects the top-κ instances that increase the loss between the current and previous iteration. However, the existing rehearsal-based methods [125, 4, 3, 19, 20] do not select the coreset before the current task adaptation and update the model on all the arriving data streams, which makes them susceptible to real-world applications that include noisy and imbalanced data distributions. In contrast, OCS selects the instances before updating the model using our proposed selection criteria, which makes it robust to *past and current task training* across various CL scenarios.

Coreset selection. There exist various directions to obtain a coreset from a large dataset. Importance sampling [64, 70, 145] strengthens the loss/gradients of important samples based on influence functions. *Kool et al.* [75] connect stochastic Gumbel-top-k trick and beam search to hierarchically sample sequences without replacement. iCaRL [125] propose a herding-based strategy for coreset selection. VCL [113] formulate the coreset summarization in continual learning using online variational inference [134, 12]. GSS [4] select the replay buffer to maximize the variance in the gradient space. Contrary to these methods, OCS considers diversity, task informativity, and relevancy to past tasks. Recently, *Borsos et al.* [11] propose a bilevel optimization framework with cardinality constraints for coreset selection. However, their method is extremely limited in practice and inapplicable in large-scale settings due to the excessive computational cost incurred during training. In contrast, our method is simple and scalable since it can construct the coreset in the online streaming data continuum without additional optimization constraints.

4.1.3 Approach

Rehearsal-based Continual Learning

We consider learning a model over a sequence of tasks $\{\mathcal{T}_1, \dots, \mathcal{T}_T\} = \mathcal{T}$, where each task is composed of independently and identically distributed datapoints and their labels, such that task $\mathcal{T}_t$ includes $\mathcal{D}_t = \{\boldsymbol{x}_{t,n}, y_{t,n}\}_{n=1}^{N_t} \sim \mathcal{X}_t \times \mathcal{Y}_t$, where N_t is the total number of data instances, and $\mathcal{X}_t \times \mathcal{Y}_t$ is an unknown data generating distribution. We assume that an arbitrary set of labels for task $\mathcal{T}_t$, $\mathbf{y}_t = \{y_{t,n}\}_{n=1}^{N_t}$ has unique classes, $\mathbf{y}_t \cap \mathbf{y}_k = \emptyset$, $\forall t \neq k$. In a standard continual learning scenario, the model learns a corresponding task at each step and t-th task is accessible at step t only. Let neural network $f_\Theta : \mathcal{X}_{1:T} \rightarrow \mathcal{Y}_{1:T}$ be parameterized by a set of weights $\Theta = \{\boldsymbol{\theta}_l\}_{l=1}^{L}$, where L is the number of layers in the neural network. We define the training objective at step t as follows:

$$\underset{\Theta}{\text{minimize}} \sum_{n=1}^{N_t} \ell(f_\Theta(\boldsymbol{x}_{t,n}), y_{t,n}), \tag{4.1}$$

where $\ell(\cdot)$ is any standard loss function (e.g., cross-entropy loss). The naive CL design where a simple sequential training on multiple tasks without any means for tackling catastrophic forgetting cannot retain the knowledge of previous tasks and thus results in catastrophic forgetting. To tackle this problem, rehearsal-based methods [113, 19, 153] update the model on a randomly sampled replay buffer $\mathcal{C}_k$ constructed from the previously observed tasks, where $\mathcal{C}_k = \{\boldsymbol{x}_{k,j}, y_{k,j}\}_{j=1}^{J_k} \sim \mathcal{D}_k$, $\forall k < t$ and $J_k \ll N_k$. Consequently, the quality of selected instances is essential for rehearsal-based continual learning. For

example, some data instances can be more informative and representative than others to describe a task and improve model performance. In contrast, some data instances can degrade the model's memorization of past tasks' knowledge. Therefore, obtaining the most beneficial examples for the current task is crucial for the success of rehearsal-based CL methods.

To validate our hypothesis that samples have a different impact on current task adaptation and catastrophic forgetting of past tasks, we design a simple learning scenario with a sequence of two tasks, MNIST (T1) → CIFAR-10 (T2). 41To validate our hypothesis that samples have a different impact on current task adaptation and catastrophic forgetting of past tasks, we design a learning scenario with a sequence of two tasks, MNIST ($\mathcal{T}_1$) → CIFAR-10 ($\mathcal{T}_2$) using ResNet-18. After the standard single epoch training on $\mathcal{T}_1$, we independently update the model weights through a single backpropagation step using a single data point from $\mathcal{T}_2$, and measure the test accuracy of its corresponding class c and averaged forgetting of the entire dataset of a past task T1 $\mathcal{T}_1$. Results for individual impacts on 1000 data points from $\mathcal{T}_2$ are described in Figure 4.3. The influence of each data point from $\mathcal{T}_2$ has a large disparity not only on the corresponding class accuracy but also on past task's forgetting that results in a very high standard deviation. We emphasize that each data point has a different potential impact in terms of forgetting past tasks. Few data points are much more robust to catastrophic forgetting than others, and this can be severe when the influences are accumulated during training.

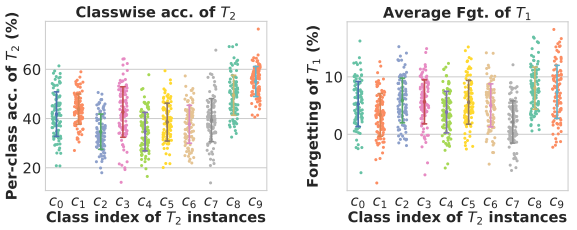


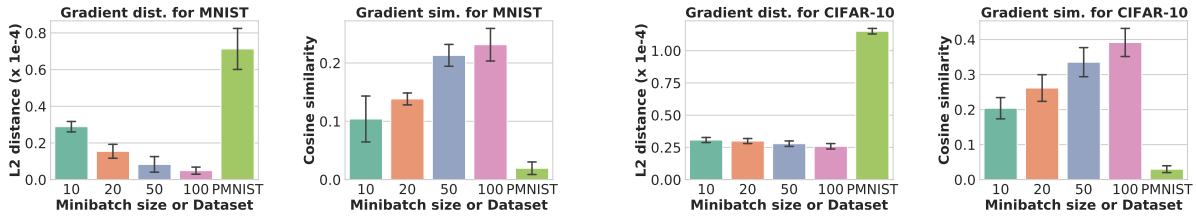
Figure 4.3: **Per-class accuracy and average forgetting** when a model trained on MNIST ($\mathcal{T}_1$) is updated on a single data point at class c on CIFAR-10 ($\mathcal{T}_2$).

Based on this motivation, our objective is to select the data instances that can promote current task adaptation while minimizing catastrophic forgetting of the previous tasks. We propose a selection criterion that selects the subset that maximizes the gradient similarity between the representative instances and the current task dataset. More formally:

$$\mathbf{u}^* = \underset{\mathbf{u}\in\mathbb{N}^\kappa}{\text{maximize}}\ \mathcal{S}\left(\frac{1}{N_t}\nabla f_\Theta\left(\mathcal{D}_t\right), \frac{1}{\kappa}\sum_{n\in\mathbf{u}}\nabla f_\Theta\left(\boldsymbol{x}_{t,n}, y_{t,n}\right)\right), \quad \text{where } \mathbf{u} = \{n : n \in \mathbb{N}_{<N_t}\}, \tag{4.2}$$

where $\mathcal{S}$ is any arbitrary similarity function and $\mathbf{u}^*$ is an index set that selects top-κ informative samples without replacement. However, obtaining a representative subset from the entire dataset is computationally expensive and intractable for online continual learning; therefore, we consider a minibatch as an approximation of the dataset and select a few representative data instances at each minibatch iteration. Consequently, the model iteratively updates the parameters to find the optimal local minima of the loss using informative data points, which obtain similar gradient directions with the averaged gradients of the dataset. In the next section, we propose *Online Coreset Selection* (OCS), which consists of a simple similarity criterion to achieve this objective. However, the similarity criterion is insufficient to select the representative coreset for online continual learning; hence, we propose diversity and coreset affinity criteria to mitigate catastrophic forgetting.

Distance between the whole dataset and its minibatch. To verify our conjecture that a minibatch can be an approximation of the whole dataset and select a few representative data instances at each minibatch iteration, we empirically validate that the whole dataset and its minibatch have significant semantic relevancy. More formally, for a given subset $\mathcal{B}_t$, dataset $\mathcal{D}_t$, and $\epsilon > 0$, we suppose that the



MNIST dataset using MLP

CIFAR-10 dataset using ResNet-18

Figure 4.4: **Empirical validation of ℓ_2 distance and cosine similarity between the gradient of the entire dataset and its minibatch gradient.** We report the mean and standard deviation of the metrics across five independent runs.

neural network satisfies following equation,

$$\mathcal{S}^*\left(\frac{1}{N_t}\nabla f_\Theta(\mathcal{D}_t), \frac{1}{|\mathcal{B}_t|}\nabla f_\Theta(\mathcal{B}_t)\right) \leq \epsilon, \tag{4.3}$$

where $\mathcal{S}^*$ is an arbitrary distance function. To this end, we conduct a simple experiment using two different datasets (MNIST and CIFAR-10) and network architectures (MLP and ResNet-18) to show that it generally holds on various datasets and network architectures. We use a 2-layered MLP for MNIST and ResNet-18 for CIFAR-10 in the following analysis. At each iteration of training, we measure the ℓ_2 distance ($\mathcal{S}^*(\cdot) = \ell_2(\cdot)$) and cosine similarity ($\mathcal{S}^*(\cdot) = 1/(\text{sim}(\cdot))$) between the training minibatch's averaged gradient and the entire training dataset. To show that the distance is sufficiently small, we also measure the gradient ℓ_2 distance between the entire and irrelevant datasets. Note that we recalculated the gradient for the whole dataset at each iteration for all results to correctly measure the distance at each step. As shown in Figure 4.4, the gradient of a larger minibatch shows a better approximation to the gradient of the entire dataset for ℓ_2 distance and cosine similarity. Further, note that even the gradients of an arbitrary subset with a small-sized minibatch are significantly similar to the whole dataset with a small ϵ, and it is more evident when compared to the irrelevant gradients from a different dataset, PMNIST (Permuted MNIST) [45].

Online Coreset Selection

We introduce our selection strategies and propose Online Coreset Selection (OCS) to strengthen current task adaptation and mitigate catastrophic forgetting. Thus far, the rehearsal-based continual learning methods [125, 4, 3, 19, 20] populate the replay buffer to preserve the knowledge of the previous tasks. However, we argue that some instances may be non-informative and inappropriate to construct the replay buffer under realistic setups (such as video streaming or imbalanced continual learning scenarios), leading to the degradation of the model's performance. Moreover, selecting valuable samples for current task training is critical since the model can easily overfit the biased and noisy data stream, negatively affecting the model generalization. To satisfy these desiderata, we propose *minibatch similarity* ($\mathcal{S}$) and *sample diversity* ($\mathcal{V}$) criteria based on our assumption, as mentioned earlier, to adaptively select the useful instances without the influence of outliers.

Definition 1 (Minibatch similarity). *Let $\boldsymbol{b}_{t,n} = \{\boldsymbol{x}_{t,n}, y_{t,n}\} \in \mathcal{B}_t$ denote n-th pair of data point with gradient $\nabla f_\Theta(\boldsymbol{b}_{t,n})$ and its corresponding label at task $\mathcal{T}_t$. Let $\bar{\nabla} f_\Theta(\mathcal{B}_t)$ denote the averaged gradient*

vector of $\mathcal{B}_t$. The minibatch similarity $\mathcal{S}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t\right)$ between $\boldsymbol{b}_{t,n}$ and $\mathcal{B}_t$ is given by

$$\mathcal{S}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t\right) = \frac{\nabla f_\Theta\left(\boldsymbol{b}_{t,n}\right) \bar{\nabla} f_\Theta\left(\mathcal{B}_t\right)^\top}{\|\nabla f_\Theta\left(\boldsymbol{b}_{t,n}\right)\| \cdot \|\bar{\nabla} f_\Theta\left(\mathcal{B}_t\right)\|}. \tag{4.4}$$

Definition 2 (Sample diversity). *Let $\boldsymbol{b}_{t,n} = \{\boldsymbol{x}_{t,n}, y_{t,n}\} \in \mathcal{B}_t$ denote n-th pair of a data point with gradient $\nabla f_\Theta\left(\boldsymbol{b}_{t,n}\right)$ and its corresponding label at task $\mathcal{T}_t$. The sample diversity $\mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t\setminus \boldsymbol{b}_{t,n}}\right)$ between $\boldsymbol{b}_{t,n}$ and all other instances in $\mathcal{B}_t$ $\left(\mathcal{B}_{t\setminus \boldsymbol{b}_{t,n}}\right)$ is given by*

$$\mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t\setminus \boldsymbol{b}_{t,n}}\right) = \frac{-1}{N_t - 1} \sum_{p \neq n}^{N_t - 1} \frac{\nabla f_\Theta\left(\boldsymbol{b}_{t,n}\right) \nabla f_\Theta\left(\boldsymbol{b}_{t,p}\right)^\top}{\|\nabla f_\Theta\left(\boldsymbol{b}_{t,n}\right)\| \cdot \|\nabla f_\Theta\left(\boldsymbol{b}_{t,p}\right)\|}. \tag{4.5}$$

In particular, minibatch similarity considers a minibatch as an approximation of the current task dataset and compares the minibatch-level similarity between the gradient vector of a data point $\boldsymbol{b}$ and its minibatch $\mathcal{B}$. It measures how well a given data instance describes the current task at each training step. Note that selecting examples with the largest minibatch similarity is reasonable when the variance of task instances is low; otherwise, it increases the redundancy among coreset items. In contrast, we formulate the sample diversity of each data point $b \in \mathcal{B}$ as an averaged dissimilarity (i.e., an average of negative similarities) between a data point itself and other samples in the same minibatch $\mathcal{B}$, and not as an average similarity. Thus, the measure of sample diversity in Equation (4) is negative, and the range is $[-1, 0]$.

Online Coreset Selection for Current Task Adaptation

The model receives a data continuum during training, including noisy or redundant data instances in real-world scenarios. Consequently, the arriving data instances can interrupt and hurt the performance of the model. To tackle this problem, we consider an amalgamation of minibatch similarity and sample diversity to select the most helpful instances for current task training. More formally, our online coreset selection for the current task adaptation can be defined as follows:

$$\mathbf{u}^* = \left\{ \underset{n}{\operatorname{argmax}^{(\kappa)}} \; \mathcal{S}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t\right) + \mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t\setminus \boldsymbol{b}_{t,n}}\right) \;\middle|\; n \in \{0, \ldots, |\mathcal{B}_t| - 1\} \right\}. \tag{4.6}$$

We emphasize that we can obtain the top-κ valuable instances for the current task by computing Equation 4.6 in an online manner. Once the representative coreset is selected, we optimize the following objective for the current task training at each iteration:

$$\underset{\Theta}{\text{minimize}} \; \frac{1}{\kappa} \sum_{(\hat{\boldsymbol{x}}, \hat{y}) \in \widehat{\mathcal{B}}_t}^{\kappa} \ell\left(f_\Theta\left(\hat{\boldsymbol{x}}\right), \hat{y}\right), \quad \text{where} \;\; \widehat{\mathcal{B}}_t = \mathcal{B}_t[\mathbf{u}^*]. \tag{4.7}$$

We consider a selected coreset at each iteration as a candidate for the replay buffer. After the completion of each task training, we choose a coreset $\mathcal{C}_t$ among the collected candidates, or we may also iteratively update $\mathcal{C}_t$ to maintain the bounded buffer size for continual learning.

Online Coreset Selection for Continual Learning

We now formulate OCS for online continual learning, where our objective is to obtain the coreset to retain the knowledge of the previous tasks using our proposed similarity and diversity selection criteria. However, continual learning is more challenging as the model suffers from catastrophic forgetting and the coreset size is smaller than the size of the arriving data streams. Thus, inspired by our observation

in Figure 4.3, we aim to train the continual learner on the selected instances that represent the current task and prevent the performance degeneration of previous tasks.

We achieve our goal by introducing our Coreset affinity criterion $\mathcal{A}$ to Equation 4.6. In particular, $\mathcal{A}$ computes the gradient vector similarity between a training sample and the coreset for previous tasks. More formally, $\mathcal{A}$ can be defined as follows:

Definition 3 (Coreset affinity). *Let $\boldsymbol{b}_{t,n} = \{\boldsymbol{x}_{t,n}, y_{t,n}\} \in \mathcal{B}_t$ denote the n-th pair of a data point with gradient $\nabla f_\Theta(\boldsymbol{b}_{t,n})$ and its corresponding label at task $\mathcal{T}_t$. Further let $\bar{\nabla} f_\Theta(\mathcal{B}_\mathcal{C})$ be the averaged gradient vector of $\mathcal{B}_\mathcal{C}$, which is randomly sampled from the coreset $\mathcal{C}$. The coreset affinity $\mathcal{A}(\boldsymbol{b}_{t,n} \mid \mathcal{B}_\mathcal{C} \sim \mathcal{C})$ between $\boldsymbol{b}_{t,n}$ and $\mathcal{B}_\mathcal{C}$ is given by*

$$\mathcal{A}(\boldsymbol{b}_{t,n} \mid \mathcal{B}_\mathcal{C} \sim \mathcal{C}) = \frac{\nabla f_\Theta(\boldsymbol{b}_{t,n})\, \bar{\nabla} f_\Theta(\mathcal{B}_\mathcal{C})^\top}{\|\nabla f_\Theta(\boldsymbol{b}_{t,n})\| \cdot \|\bar{\nabla} f_\Theta(\mathcal{B}_\mathcal{C})\|}. \tag{4.8}$$

While the past task is inaccessible after the completion of its training, our selectively populated replay buffer can be effectively used to describe the knowledge of the previous tasks. The key idea is to select the examples that minimize the angle between the gradient vector of the coreset containing previous task examples and the current task examples. Instead of randomly replacing the candidates in the coreset [102, 19, 4, 11], $\mathcal{A}$ promotes the selection of examples that do not degenerate the model performance on previous tasks. To this end, we select the most beneficial training instances which are representative and diverse for current task adaptation while maintaining the knowledge of past tasks. In summary, our OCS for training task $\mathcal{T}_t$ during CL can be formulated as:

$$\mathbf{u}^* = \Big\{ \underset{n}{\text{argmax}^{(\kappa)}}\ \mathcal{S}(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t) + \mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t\setminus \boldsymbol{b}_{t,n}}\right) + \tau \mathcal{A}(\boldsymbol{b}_{t,n} \mid \mathcal{B}_\mathcal{C}) \;\Big|\; n \in \{0, \ldots, |\mathcal{B}_t| - 1\} \Big\}. \tag{4.9}$$

τ is a hyperparameter that controls the degree of model plasticity and stability. Note that, during the first task training, we do not have interference from previous tasks, and we select the top-κ instances that maximize the minibatch similarity and sample diversity. Given the obtained coreset $\widehat{\mathcal{B}}_t = \mathcal{B}_t[\mathbf{u}^*]$, our optimization objective reflecting the coreset $\mathcal{C}$, is as follows:

$$\underset{\Theta}{\text{minimize}}\ \frac{1}{\kappa} \sum_{(\hat{\boldsymbol{x}}, \hat{y}) \in \widehat{\mathcal{B}}_t}^{\kappa} \ell(f_\Theta(\hat{\boldsymbol{x}}), \hat{y}) + \frac{\lambda}{|\mathcal{B}_\mathcal{C}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{B}_\mathcal{C}}^{|\mathcal{B}_\mathcal{C}|} \ell(f_\Theta(\boldsymbol{x}), y), \tag{4.10}$$

where $\mathcal{B}_\mathcal{C}$ is a randomly sampled minibatch from the coreset $\mathcal{C}$ and λ is a hyperparameter to balance the adaptation between the current task and past task coreset. The overall training procedure for *Online Coreset Selection (OCS)* is described in Algorithm 9. To the best of our knowledge, this is the first work that utilizes *selective online training* for the current task training and incorporates the relationship between the selected coreset and the current task instances to promote current task adaptation while minimizing interference with previous tasks.

Algorithm 9 Online Coreset Selection (OCS)

input Dataset $\{\mathcal{D}_t\}_{t=1}^{T}$, neural network f_Θ, hyperparameters λ, τ, replay buffer $\mathcal{C} \leftarrow \{\}$, buffer size bound J.

for task $\mathcal{T}_t = \mathcal{T}_1, \ldots, \mathcal{T}_T$ **do**
  $\mathcal{C}_t \leftarrow \{\}$ ▷ Initialize coreset for current task
  for batch $\mathcal{B}_t \sim \mathcal{D}_t$ **do**
    $\mathcal{B}_\mathcal{C} \leftarrow \textsc{Sample}(\mathcal{C})$ ▷ Randomly sample a batch from the replay buffer
    $\mathbf{u}^* = \underset{n \in \{0,\ldots,|\mathcal{B}_t|-1\}}{\operatorname{argmax}^{(\kappa)}} \mathcal{S}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_t\right) + \mathcal{V}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_{t \setminus b_{t,n}}\right) + \tau \mathcal{A}\left(\boldsymbol{b}_{t,n} \mid \mathcal{B}_\mathcal{C}\right)$ ▷ Coreset selection
    $\Theta \leftarrow \Theta - \eta \nabla f_\Theta(\mathcal{B}_t[\mathbf{u}^*] \cup \mathcal{B}_\mathcal{C})$ with Equation equation 4.10 ▷ Model update with selected instances
    $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \widehat{\mathcal{B}}_t$
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup \textsc{Select}(\mathcal{C}_t, \text{size} = J/T)$ with Equation (4.9) ▷ Memorize coreset in the replay buffer
end for

4.1.4 Experimental Results

Datasets

We validate OCS on domain-incremental CL for Balanced and Imbalanced Rotated MNIST using a single-head two-layer MLP with 256 ReLU units in each layer, task-incremental CL for Split CIFAR-100 and Multiple Datasets (a sequence of five datasets) with a multi-head structured ResNet-18 following prior works [19, 109, 108].

Baselines.

We compare OCS with regularization-based CL methods: EWC [74] and Stable SGD [109], rehearsal-based CL methods using random replay buffer: A-GEM [19] and ER-Reservior [20], coreset-based methods using CL algorithms: Uniform Sampling, k-means Features [113] and k-means Embeddings [136], and coreset-based CL methods: iCaRL [125], Grad Matching [16], GSS [4], ER-MIR [3], and Bilevel Optim [11]. We limit the buffer size for the rehearsal-based methods to one example per class per task. Additionally, we compare with Finetune, a naive CL method learned on a sequence of tasks, and Multitask, where the model is trained on the complete data.

Metrics.

We evaluate all the methods on two metrics following the CL literature [19, 108].

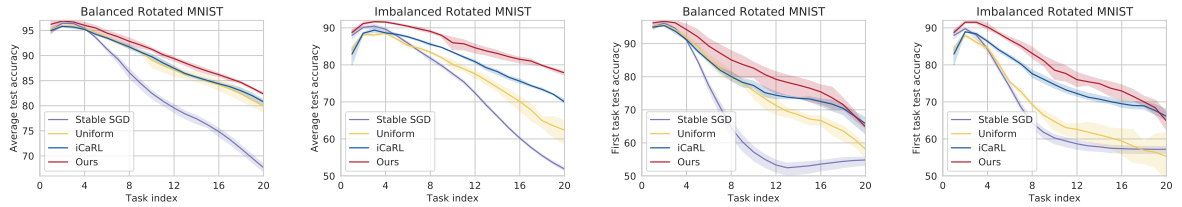
1. **Average Accuracy** (A_t) is the averaged test accuracy of all tasks after the completion of CL at task $\mathcal{T}_t$. That is, $A_t = \frac{1}{t}\sum_{i=1}^{t} a_{t,i}$, where $a_{t,i}$ is the test accuracy of task $\mathcal{T}_i$ after learning task $\mathcal{T}_t$.
2. **Average Forgetting** (F) is the averaged disparity between the peak and final task accuracy after the completion of continual learning. That is, $F = \frac{1}{T-1}\sum_{i=1}^{T-1} \max_{t \in \{1,\ldots,T-1\}}(a_{t,i} - a_{T,i})$.

Quantitative Analysis for Continual Learning

Balanced continual learning. Table 4.1 shows the results on the balanced CL benchmarks. First, observe that compared to the random replay-based methods (A-GEM and ER-Reservoir), OCS shows 19% relative gain in average accuracy, 62% and 79% reduction in forgetting over the strongest baseline on Rotated MNIST, and Split CIFAR-100, respectively. Second, OCS reduces the forgetting by 38% and 57% on Rotated MNIST and Multiple Datasets, respectively, over the coreset-based techniques,

Table 4.1: Performance comparison of OCS and other baselines on balanced and imbalanced continual learning. We report the mean and standard-deviation of the average accuracy (Accuracy) and average forgetting (Forgetting) across five independent runs. The best results are highlighted in **bold**.

	Method	Rotated MNIST		Split CIFAR-100		Multiple Datasets	
		Accuracy	Forgetting	Accuracy	Forgetting	Accuracy	Forgetting
Balanced CL	Finetune	46.3 (± 1.37)	0.52 (± 0.01)	40.4 (± 2.83)	0.31 (± 0.02)	49.8 (± 2.14)	0.23 (± 0.03)
	EWC [74]	70.7 (± 1.74)	0.23 (± 0.01)	48.5 (± 1.24)	0.48 (± 0.01)	42.7 (± 1.89)	0.28 (± 0.03)
	Stable SGD [109]	70.8 (± 0.78)	0.10 (± 0.02)	57.4 (± 0.91)	0.07 (± 0.01)	53.4 (± 2.66)	0.16 (± 0.03)
	A-GEM [19]	55.3 (± 1.47)	0.42 (± 0.01)	50.7 (± 2.32)	0.19 (± 0.04)	–	–
	ER-Reservoir [20]	69.2 (± 1.10)	0.21 (± 0.01)	46.9 (± 0.76)	0.21 (± 0.03)	–	–
	Uniform Sampling	79.9 (± 1.32)	0.14 (± 0.01)	58.8 (± 0.89)	0.05 (± 0.01)	56.0 (± 2.40)	0.11 (± 0.02)
	iCaRL [125]	80.7 (± 0.44)	0.13 (± 0.00)	60.3 (± 0.91)	0.04 (± 0.00)	59.4 (± 1.43)	0.07 (± 0.02)
	k-means Features [113]	79.1 (± 1.50)	0.14 (± 0.01)	59.3 (± 1.21)	0.06 (± 0.01)	53.6 (± 1.98)	0.14 (± 0.02)
	k-means Embedding [136]	80.6 (± 0.54)	0.13 (± 0.01)	55.5 (± 0.70)	0.06 (± 0.01)	55.4 (± 1.46)	0.11 (± 0.02)
	Grad Matching [16]	78.5 (± 0.86)	0.15 (± 0.01)	60.0 (± 1.24)	0.04 (± 0.01)	57.8 (± 1.35)	0.08 (± 0.02)
	GSS [4]	76.0 (± 0.58)	0.19 (± 0.01)	59.7 (± 1.22)	0.04 (± 0.01)	60.2 (± 1.00)	0.07 (± 0.01)
	ER-MIR [3]	80.7 (± 0.72)	0.14 (± 0.01)	60.2 (± 0.72)	0.04 (± 0.00)	56.9 (± 2,25)	0.11 (± 0.03)
	Bilevel Optim [11]	80.7 (± 0.44)	0.14 (± 0.00)	60.1 (± 1.07)	0.04 (± 0.01)	58.1 (± 2.26)	0.08 (± 0.02)
	OCS (Ours)	**82.5** (± 0.32)	**0.08** (± 0.00)	**60.5** (± 0.55)	**0.04** (± 0.01)	**61.5** (± 1.34)	**0.03** (± 0.01)
	Multitask	89.8 (± 0.37)	–	71.0 (± 0.21)	–	57.4 (± 0.84)	–
Imbalanced CL	Finetune	39.8 (± 1.06)	0.54 (± 0.01)	45.3 (± 1.38)	0.17 (± 0.01)	27.6 (± 3.66)	0.22 (± 0.04)
	Stable SGD [109]	52.0 (± 0.25)	0.19 (± 0.00)	48.7 (± 0.64)	0.03 (± 0.00)	29.5 (± 4.09)	0.20 (± 0.02)
	Uniform Sampling	61.6 (± 1.72)	0.15 (± 0.01)	51.0 (± 0.78)	0.03 (± 0.00)	35.0 (± 3.03)	0.11 (± 0.03)
	iCaRL [125]	71.7 (± 0.69)	0.09 (± 0.00)	51.2 (± 1.09)	0.02 (± 0.00)	43.6 (± 2.95)	0.05 (± 0.03)
	k-means Features [113]	52.3 (± 1.48)	0.24 (± 0.01)	50.6 (± 1.52)	0.04 (± 0.01)	36.1 (± 1.75)	0.09 (± 0.02)
	k-means Embedding [136]	63.2 (± 0.90)	0.13 (± 0.02)	50.4 (± 1.39)	0.03 (± 0.01)	35.6 (± 1.35)	0.11 (± 0.02)
	Grad Matching [16]	55.6 (± 1.86)	0.18 (± 0.02)	51.1 (± 1.14)	0.02 (± 0.00)	34.6 (± 0.50)	0.12 (± 0.01)
	GSS [4]	68.7 (± 0.98)	0.18 (± 0.01)	44.5 (± 1.35)	0.04 (± 0.01)	32.9 (± 0.90)	0.13 (± 0.01)
	ER-MIR [3]	69.3 (± 1.01)	0.16 (± 0.01)	44.8 (± 1.42)	0.03 (± 0.01)	32.3 (± 3.49)	0.15 (± 0.03)
	Bilevel Optim [11]	63.2 (± 1.04)	0.22 (± 0.01)	44.0 (± 0.86)	0.03 (± 0.01)	35.1 (± 2.78)	0.12 (± 0.02)
	OCS (Ours)	**76.5** (± 0.84)	**0.08** (± 0.01)	**51.4** (± 1.11)	**0.02** (± 0.00)	**47.5** (± 1.66)	**0.03** (± 0.02)
	Multitask	81.0 (± 0.95)	–	48.2 (± 0.72)	–	41.4 (± 0.97)	–



(a) Average test accuracy (b) First task test accuracy

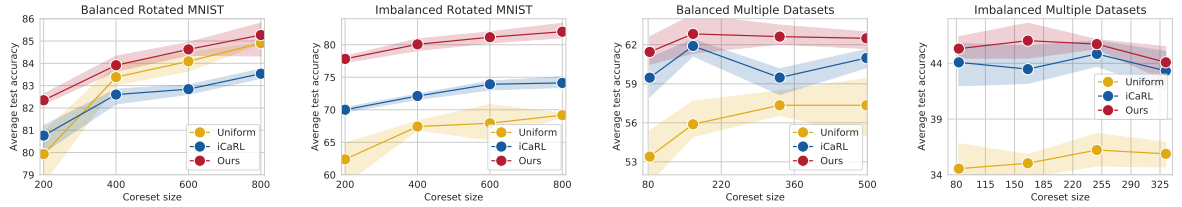
Figure 4.5: **(a)** Average accuracy **(b)** First task accuracy for balanced/imbalanced Rotated MNIST during CL.

demonstrating that it selects valuable samples from the previous tasks. We further illustrate this in Figure 4.5, where OCS consistently exhibits superior average accuracy and first-task accuracy. Third, we show the scalability of OCS with larger episodic memory in Figure 4.6. Interestingly, iCaRL shows lower performance than uniform sampling with a larger memory buffer for Rotated MNIST, while OCS outperforms across all memory sizes on both datasets. Furthermore, we note that ER-MIR, GSS, and Bilevel Optim require 0.9×, 3.9×, and 4.2× training time than OCS (see Table 4.5) on TITAN Xp, showing a clear advantage of OCS for the online streaming scenarios.

Imbalanced continual learning. To demonstrate the effectiveness of OCS in challenging scenarios, we evaluate imbalanced CL in Table 4.1. We emphasize that compared to balanced CL, OCS shows significant gains over all the baselines for Rotated MNIST and Multiple Datasets. Notably, it leads to a

Table 4.2: Performance comparison of OCS and other baselines on varying proportions of noise instances during noisy continual learning. We report the mean and standard-deviation of the average accuracy (Accuracy) and average forgetting (Forgetting) across five independent runs. The best results are highlighted in **bold**.

Method	0%		40%		60%	
	Accuracy	Forgetting	Accuracy	Forgetting	Accuracy	Forgetting
Stable SGD [109]	70.8 (± 0.78)	0.10 (± 0.02)	56.2 (± 0.95)	0.40 (± 0.01)	56.1 (± 0.62)	0.40 (± 0.01)
Uniform sampling	79.9 (± 1.32)	0.14 (± 0.01)	74.9 (± 2.45)	0.20 (± 0.03)	68.3 (± 3.68)	0.26 (± 0.03)
iCaRL [125]	80.7 (± 0.44)	0.13 (± 0.00)	77.4 (± 0.60)	0.18 (± 0.01)	71.4 (± 2.63)	0.23 (± 0.03)
k-means embedding [136]	80.6 (± 0.54)	0.13 (± 0.01)	78.5 (± 0.86)	0.17 (± 0.00)	77.5 (± 1.67)	0.26 (± 0.03)
GSS [4]	76.0 (± 0.58)	0.19 (± 0.01)	71.7 (± 0.95)	0.19 (± 0.01)	68.8 (± 1.02)	0.17 (± 0.02)
ER-MIR [3]	80.7 (± 0.72)	0.14 (± 0.01)	76.0 (± 1.34)	0.17 (± 0.01)	73.5 (± 0.94)	0.18 (± 0.01)
OCS (Ours)	**82.5** (± 0.32)	**0.08** (± 0.00)	**80.4** (± 0.20)	**0.14** (± 0.00)	**80.3** (± 0.75)	**0.10** (± 0.01)



(a) Rotated MNIST (b) Multiple Datasets

Figure 4.6: Performance comparison on various coreset sizes for balanced/imbalanced continual learning.

relative improvement of $\sim 7\%$ and $\sim 9\%$ on the accuracy, $\sim 11\%$ and 40% reduction on the forgetting compared to the best baseline for each dataset, respectively. The poor performance of the baselines in this setup is largely attributed to their lack of current task coreset selection, which results in a biased estimate degenerating model performance (see Figure 4.9). Moreover, we observe that OCS outperforms Multitask for complex imbalanced datasets, perhaps due to the bias from the dominant classes and the absence of selection criteria in Multitask. Similar to balanced CL, OCS leads to superior performance for larger episodic memory in imbalanced CL (see Figure 4.6).

Noisy continual learning. Next, we evaluate on noisy Rotated MNIST dataset, which is constructed by perturbing a proportion of instances of the original dataset with Gaussian noise $\mathcal{N}(0, 1)$. Table 4.2 shows that the addition of noise significantly degrades the performance on all the baselines. In contrast, OCS leads to a relative gain of 43% on accuracy, 20% and 35% reduction in forgetting on 40% and 60% proportion of noisy data. Note that the performance gap is more significant for the higher distribution of noisy examples, supporting our claim that the similarity and diversity across the training examples in the coreset play an essential role in task adaptation in continual learning.

Ablation Studies

Effect of gradients. In Table 4.3, we empirically justify the utilization of gradients (Grad-OCS) compared to the raw inputs (Input-OCS) and feature-representations (Feat-OCS). We observe that Grad-OCS significantly outperforms Input-OCS and Feat-OCS on balanced and imbalanced CL, demonstrating that the gradients are a better metric to approximate the dataset.

Effect of individual components. We further dissect Minibatch similarity ($\mathcal{S}$), Sample diversity ($\mathcal{V}$) and Coreset affinity ($\mathcal{A}$) in Table 4.4. Note that selection using $\mathcal{S}$ shows reasonable performance as the model can select valuable data points; however, it may select redundant samples, which degrades

Table 4.3: Ablation study for analyzing the effect of gradients selection for OCS.

Method	Balanced Rotated MNIST		Imbalanced Rotated MNIST	
	Accuracy	Forgetting	Accuracy	Forgetting
Input-OCS	72.7 (± 0.47)	0.13 (± 0.01)	50.6 (± 1.74)	0.04 (± 0.00)
Feat-OCS	71.7 (± 0.62)	0.17 (± 0.01)	30.6 (± 0.40)	0.03 (± 0.01)
Grad-OCS	**82.5** (± 0.32)	**0.08** (± 0.00)	**76.5** (± 0.84)	**0.08** (± 0.01)

Table 4.4: Ablation study to investigate the impact of selection criteria $\mathcal{S}, \mathcal{V}$, and $\mathcal{A}$ on OCS.

Method			Noisy Rot-MNIST (60%)		Multiple Datasets	
$\mathcal{S}$	$\mathcal{V}$	$\mathcal{A}$	Accuracy	Forgetting	Accuracy	Forgetting
✓	–	–	64.0 (± 1.18)	0.33 (± 0.01)	56.3 (± 0.97)	0.10 (± 0.02)
–	✓	–	40.1 (± 1.32)	0.06 (± 0.02)	49.8 (± 1.30)	0.12 (± 0.01)
–	–	✓	79.6 (± 0.87)	0.10 (± 0.01)	58.1 (± 0.96)	0.05 (± 0.01)
✓	✓	–	66.8 (± 1.39)	0.30 (± 0.01)	58.6 (± 1.91)	0.09 (± 0.02)
✓	✓	✓	**80.3** (± 0.75)	**0.10** (± 0.01)	**61.5** (± 1.34)	**0.03** (± 0.01)

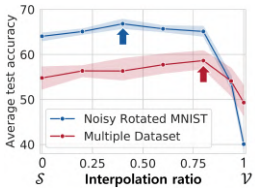


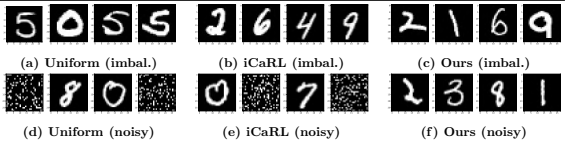
Figure 4.7: Interpolation between $\mathcal{S}$ and $\mathcal{V}$.

Table 4.5: Running time on Balanced Rot-MNIST.

Method	Training Time
ER-MIR	0.38 h (×0.87)
GSS	1.71 h (×3.89)
Bilevel	1.83 h (×4.17)
OCS (Ours)	0.44 h (×1.00)

Table 4.6: Collaborative learning with rehearsal-based CL on various datasets with 20 tasks each.

	MC-SGD [108]		MC-SGD + OCS	
Dataset	Accuracy	Forgetting	Accuracy	Forgetting
Per-MNIST	84.6 (± 0.54)	0.06 (± 0.01)	**86.6** (± 0.42)	**0.02** (± 0.00)
Rot-MNIST	82.3 (± 0.68)	0.08 (± 0.01)	**85.1** (± 0.27)	**0.04** (± 0.00)
Split CIFAR	58.4 (± 0.95)	0.02 (± 0.00)	**59.1** (± 0.55)	**0.00** (± 0.00)



(a) Uniform (imbal.) (b) iCaRL (imbal.) (c) Ours (imbal.)

(d) Uniform (noisy) (e) iCaRL (noisy) (f) Ours (noisy)

Figure 4.8: Randomly picked coreset examples. Top: Imbalanced Rotated MNIST. Bottom: Noisy Rotated MNIST with 60% of noisy instances.

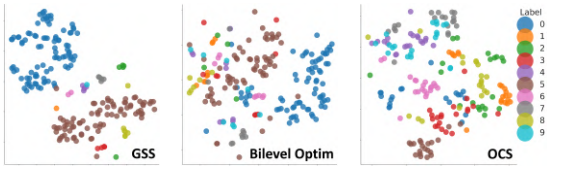


Figure 4.9: T-SNE visualization of the selected samples on Imbalanced Rotated MNIST.

its performance. In addition, $\mathcal{V}$ in isolation is insufficient since it can select non-redundant and non-representative instances. The combination of $\mathcal{S}$ and $\mathcal{V}$ improves the average accuracy, but it shows a marginal improvement in forgetting. To further gain insight into $\mathcal{S}$ and $\mathcal{V}$, we interpolate between $\mathcal{S}$ and $\mathcal{V}$ in Figure 4.7, where we can observe that an optimal balance of $\mathcal{S}$ and $\mathcal{V}$ (indicated by the arrows) can further improve the performance of our proposed selection strategy.

Furthermore, $\mathcal{A}$ improves the forgetting since the selected candidates have similar gradient direction to the coreset of the previous tasks maximizing their performance. However, $\mathcal{A}$ does not consider the current task distribution explicitly and depends on the quality of the memorized replay buffer. We observe that using $\mathcal{A}$ in isolation obtains reasonably high performance on simple digit-based domain-incremental CL problems (e.g., Rotated MNIST) due to its inherent high resemblance among same class instances. Consequently, suppressing catastrophic forgetting by selective training based on $\mathcal{A}$ shows a relatively high impact rather than selective training based on distinguishing more informative or diverse samples. In contrast, $\mathcal{A}$ in isolation for selection is insufficient for complicated and realistic CL problems such as imbalanced CL and multiple datasets, and all the three components ($\mathcal{S}$, $\mathcal{V}$, and $\mathcal{A}$) contribute to the performance of OCS. For Multiple Datasets, selection using only $\mathcal{A}$ (58.1%) obtained worse performance in comparison to $\mathcal{S}+\mathcal{V}+\mathcal{A}$ (61.5%) and $\mathcal{S}+\mathcal{V}$ (58.6%). Further, selection using $\mathcal{S}+\mathcal{A}$ ($59.4 \pm 2.0\%$) and $\mathcal{V}+\mathcal{A}$ ($56.4\% \pm 1.3$) also obtained 2.1%p and 5.1%p lower average accuracy than full OCS, respectively.

Further Analysis

Coreset visualization. Next, we visualize the coreset selected by different methods for imbalanced and noisy rotated MNIST in Figure 4.8. We observe that uniform sampling selects highly biased samples representing the dominant classes for imbalanced CL and noisy instances for noisy CL. In contrast, iCaRL

selects the representative samples per class for imbalanced CL; however, it selects noisy instances during noisy CL. In comparison, OCS selects the beneficial examples for each class during imbalanced CL and discards uninformative noisy instances in the noisy CL training regime.

T-SNE visualization. We further compare the T-SNE visualization of the selected coreset by Bilevel Optim, GSS, and OCS in Figure 4.9. We observe that the samples chosen by OCS are diverse, whereas Bilevel Optim and GSS select the majority of the samples from the dominant classes. We attribute the representative clusters and diversity in the samples selected by OCS to our proposed $\mathcal{S}$ (selects the valuable samples) and $\mathcal{V}$ (minimizes the redundancy among the selected samples) criteria.

Collaborative learning with MC-SGD. We remark that OCS can be applied to any rehearsal-based CL method with a replay buffer during training. We empirically demonstrate the effect of collaborative learning with other CL methods in Table 4.6. In particular, we use Mode Connectivity SGD (MC-SGD) [108], which encourages the mode connectivity between model parameters for continual and multitask loss and approximates the multitask loss through a randomly selected replay buffer. Note that OCS leads to a relative gain of 1.2% to 3.4% on accuracy over MC-SGD on Permuted MNIST, Rotated MNIST, and Split CIFAR-100 datasets. Furthermore, MC-SGD + OCS shows considerably lower forgetting, illustrating that OCS prevents the loss of prior task knowledge.

OCS with partial gradients. While we consider ResNet-18 as sufficiently deep neural networks, our OCS also can be utilized in extremely deep networks (e.g., ResNet-101) through OCS selection by computing partial gradients of the neural network to reduce the computational cost during online training. This simple modification is straightforward, and to validate its potential, we have performed an additional ablation study on Split TinyImageNet in Table 4.7. ResNet-18 contains a convolutional layer with a first residual block (we name it 'block 1') and three consecutive residual blocks (we name them blocks 2, 3, and 4, respectively). We have evaluated the variants of OCS which select the coreset based on weights gradients of partial blocks. The first column of the table describes gradients used in OCS selection. That is, the original OCS uses all gradients of all network components ([1,2,3,4]).

Table 4.7: Ablation study for analyzing the selection with partial gradients in OCS.

Used Blocks for OCS	Average Accuracy	Average Forgetting	Gradients Usage Ratio
[1,]	35.72 (± 0.57)	5.45 (± 0.55)	1.6
[1, 2,]	**37.42** (± 0.93)	4.70 (± 0.94)	**6.3**
[1, 2, 3,]	36.24 (± 1.09)	4.97 (± 0.88)	**25.0**
[1, 2, 3, 4]	**37.06** (± 0.69)	**4.32** (± 0.71)	100.0
[2, 3, 4]	35.10 (± 0.33)	4.83 (± 0.95)	98.4
[3, 4]	35.47 (± 0.65)	4.85 (± 0.54)	93.7
[4]	34.75 (± 1.16)	5.20 (± 0.43)	75.0

Surprisingly, we observe OCS using the earlier two blocks (i.e., [1, 2]) show 0.36%p higher performance while using only the 6.3% of gradients compared to the original OCS. We expect that this specific benefit of earlier blocks is due to the different roles of blocks in neural networks. It is well known that earlier layers relatively focus on capturing generic representations while the latter capture class-discriminative information. Thus, obtaining the coreset that represents the task-general information and preserves shared knowledge with past tasks is specifically important since generic knowledge is easy to drift and much more susceptible to catastrophic forgetting.

We believe that further investigation of this observation would definitely provide more useful insights for future work and enhance the performance of OCS while greatly reducing computational costs.

4.1.5 Summary

We propose Online Coreset Selection (OCS), a novel approach for coreset selection during online continual learning. Our approach is modeled as a gradient-based selection strategy that selects representative and diverse instances, which helps preserve the knowledge of the previous tasks at each iteration. This paper takes the first step to utilize the coreset for improving the current task adaptation while mitigating the catastrophic forgetting of previous tasks. Our experimental evaluation of the standard balanced continual learning datasets against state-of-the-art rehearsal-based techniques demonstrates the efficiency of our approach. OCS achieves impressive results on various challenging continual learning scenarios containing class-imbalanced and noisy instances. We further demonstrate the positive effect of collaborative learning of our selection strategy with existing rehearsal-based continual learning methods. Our future work will focus on improving the selection strategies and exploring ways to utilize unlabelled data streams during training.

4.2 Representational Continuity for Unsupervised Continual Learning [103]

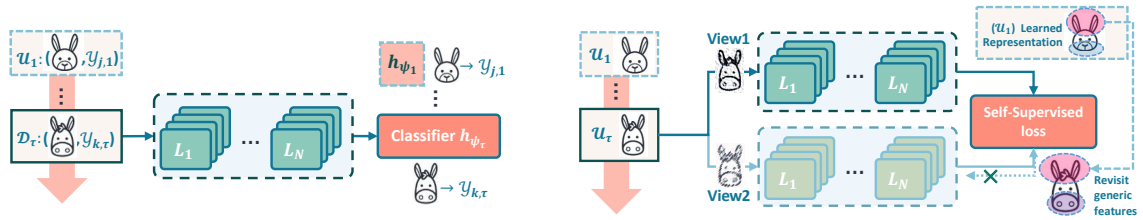
4.2.1 Motivation

Recently continual learning [152, 181, 137, 20, 3, 67, 173] has gained a lot of attention in the deep learning community due to its ability to continually learn on a sequence of non-stationary tasks [80, 97] and close proximity to the human learning process [42]. However, the inability of the learner to prevent forgetting of the knowledge learned from the previous tasks has been a long-standing problem [104, 45]. To address this problem, a large body of methods [130, 178, 95, 4, 15] have been proposed; however, all these methods focus on the supervised learning paradigm, but obtaining high-quality labels is expensive and often impractical in real-world scenarios. In contrast, CL for unsupervised representation learning has received limited attention in the community. Although [123] instantiated a continual unsupervised representation learning framework (CURL), it is not scalable for high-resolution tasks, as it is composed of MLP encoders/decoders and a simple MoG generative replay. This is evident in their limited empirical evaluation using digit-based gray-scale datasets.

Meanwhile, a set of directions have shown huge potential to tackle the representation learning problem without labels [54, 22, 47, 23, 25, 180] by aligning contrastive pairs of training instances or maximizing the similarity between two augmented views of each image. However, a common assumption for existing methods is the availability of a large amount of unbiased and unlabelled datasets to learn the feature representations. We argue that this assumption is not realistic for most of the real-time applications, including self-driving cars [10], medical applications [72] and conversational agents [96]. The collected datasets are often limited in size during the initial training phase [41], and datasets/tasks change continuously with time.

To accommodate such continuous shifts in data distributions, representation learning models need to increment the knowledge without losing the representations learned in the past. With this motivation, we attempt to bridge the gap between unsupervised representation learning and continual learning to address the challenge of learning the representations on a sequence of tasks. Specifically, we focus on *unsupervised continual learning (UCL)*, where the goal of the continual learner is to learn the representations from a stream of unlabelled data instances without forgetting (see Figure 4.10). To this end, we extend various existing SCL strategies to the unsupervised continual learning framework and analyze the performance of current state-of-the-art representation learning techniques: *SimSiam* [25] and *BarlowTwins* [180] for UCL. Surprisingly, we observe that the unsupervised representations are comparatively more robust to catastrophic forgetting across all datasets and simply fine-tuning the sequence of tasks can outperform various state-of-the-art continual learning alternatives. Furthermore, we show that UCL generalizes better to various out-of-distribution tasks and outperforms SCL for few-shot training scenarios.

We demystify the robustness of unsupervised representations by investigating the feature similarity, measured by centered kernel alignment (CKA) [76] between two independent UCL and SCL models and between UCL and SCL models. We notice that two unsupervised model representations have a relatively high feature similarity compared to two supervised representations. Furthermore, in all cases, the two models have high similarity in lower layers indicating that they learn similar low-level features. Further, we measure the ℓ_2 distance between model parameters [111] and visually compare the feature representations learned by different SCL and UCL strategies. We observe that UCL obtains human perceptual feature patterns for previous tasks, demonstrating their effectiveness in alleviating catastrophic forgetting. We



Supervised Continual Learning (SCL) Unsupervised Continual Learning (UCL)

Figure 4.10: **Illustration of supervised and unsupervised continual learning.** The objective of SCL is to learn the ability to classify labeled images in the current task while preserving the past tasks' knowledge, where the tasks are non-iid to each other. On the other hand, UCL aims to learn the representation of images without the presence of labels and the model learns general-purpose representations during sequential training.

conjecture that this is due to their characteristic ability to learn general-purpose features [33], which makes them transfer better and comparatively more robust to catastrophic forgetting.

To gain further insights, we visualize the loss landscape [93] of the UCL and SCL models and observe that UCL obtains a flatter and smoother loss landscape than SCL. Additionally, we propose a simple yet effective technique coined **L**ifelong **U**nsupervised **M**ixu**p** (Lump), which utilizes mixup [183] for unlabelled training instances. In particular, LUMP interpolates between the current task examples and examples from previous instances to minimize catastrophic forgetting. We emphasize that LUMP is easy to implement, does not require additional hyperparameters, and simply trains on the interpolated instances. To this end, LUMP requires little, or no modification to existing rehearsal-based methods effectively minimizes catastrophic forgetting even with uniformly selecting the examples from the replay buffer. We show that LUMP with UCL outperforms the state-of-the-art supervised continual learning methods across multiple experimental settings with significantly lower catastrophic forgetting. In summary, our contributions are as follows:

- We attempt to bridge the gap between continual learning and representation learning and tackle the two crucial problems of continual learning with unlabelled data and representation learning on a sequence of tasks.
- Systematic quantitative analysis shows that UCL achieves better performance over SCL with significantly lower catastrophic forgetting on Sequential CIFAR-10, CIFAR-100, and Tiny-ImageNet. Additionally, we evaluate out-of-distribution tasks and few-shot training demonstrating the expressive power of unsupervised representations.
- We provide visualization of the representations and loss landscapes, which show that UCL learns discriminative, human perceptual patterns and achieves a flatter and smoother loss landscape. Furthermore, we propose **L**ifelong **U**nsupervised **M**ixu**p** (Lump) for UCL, which effectively alleviates catastrophic forgetting and provides better qualitative interpretations.

4.2.2 Related Work

Continual learning. We can partition the existing continual learning methods into three categories. The *regularization* approaches [97, 181, 135, 2] impose a regularization constraint to the objective that mitigates catastrophic forgetting. The *architectural* approaches [130, 178, 95, 68] avoid this problem by including task-specific parameters and allowing the expansion of the network during continual learning.

The *rehearsal* approaches [125, 128, 4] allocate a small memory buffer to store and replay the examples from the previous task. However, all these methods are confined to supervised learning, which limits their application in real-life problems. [123, 146] tackled the problem of continual unsupervised representation learning; however, their methods are restricted to simple low-resolution tasks and not scalable to large-scale continual learning datasets.

Representational learning. A large number of works have addressed the unsupervised learning problem in the standard machine learning framework. Specifically, contrastive learning frameworks [54, 22, 47, 23, 24] that learn the representations by measuring the similarities of positive and negative pairs have gained a lot of attention in the community. However, all these methods require large batches and negative sample pairs, which restrict the scalability of these networks. [25] tackled these limitations and proposed *SimSiam*, that use standard Siamese networks [13] with the stop-gradient operation to prevent the collapsing of Siamese networks to a constant. Recently, BarlowTwins [180] formulated an objective that pushes the cross-correlation matrix between the embeddings of distorted versions of a sample closer to the identity matrix. However, all these methods assume the presence of large datasets for pre-training, which is impractical in real-world applications. In contrast, we tackle the problem of incremental representational learning and learn the representations sequentially while maximizing task adaptation and minimizing catastrophic forgetting.

4.2.3 Approach

Preliminaries

Problem Setup We consider the continual learning setting, where we learn on a continuum of data consisting of T tasks $\mathcal{T}_{1:T} = (\mathcal{T}_1 \dots \mathcal{T}_T)$. In supervised continual learning, each task consists of a task descriptor $\tau \in \{1 \dots T\}$ and its corresponding dataset $\mathcal{D}_\tau = \{(\boldsymbol{x}_{i,\tau}, y_{i,\tau})_{i=1}^{n_\tau}\}$ with n_τ examples. Each input-pair $(\boldsymbol{x}_{i,\tau}, y_{i,\tau}) \in \mathcal{X}_\tau \times \mathcal{Y}_\tau$, where $(\mathcal{X}_\tau, \mathcal{Y}_\tau)$ is an unkown data distribution. Let us consider a network $f_\Theta : \mathcal{X}_\tau \to \mathbb{R}^D$ parametrized by $\Theta = \{\boldsymbol{w}_l\}_{l=1}^{l=L}$, where $\mathbb{R}^D$ and L denote D-dimensional embedding space and number of layers respectively. The classifier is denoted by $h_\psi : \mathbb{R}^D \to \mathcal{Y}_\tau$. The network error using cross-entropy loss (CE) for SCL with finetuning can be formally defined as:

$$\mathcal{L}_{\text{SCL}}^{\text{FINETUNE}} = \text{CE}\left(h_\psi\left(f_\Theta\left(\boldsymbol{x}_{i,\tau}\right), \tau\right), y_{i,\tau}\right). \tag{4.11}$$

In this work, we assume the absence of label supervision during training and focus on unsupervised continual learning. In particular, each task consists of $\mathcal{U}_\tau = \{(\boldsymbol{x}_{i,\tau})_{i=1}^{n_\tau}\}$, $\boldsymbol{x}_{i,\tau} \in \mathcal{X}_\tau$ with n_τ examples. Our aim is to learn the representations $f_\Theta : \mathcal{X}_\tau \to \mathbb{R}^D$ on a sequence of tasks while preserving the knowledge of the previous tasks.

Learning Protocol and Evaluation Metrics Currently, the traditional continual learning strategies follow the standard training protocol, where we learn the network representations $f_\Theta : \mathcal{X}_\tau \to \mathcal{Y}_\tau$ on a sequence of tasks. In contrast, our goal is to learn the feature representations $f_\Theta : \mathcal{X}_\tau \to \mathbb{R}^D$, so we follow a two-step learning protocol to obtain the model predictions. First, we pre-train the representations on a sequence of tasks $T_{1:T} = (\mathcal{T} \dots \mathcal{T}_T)$ to obtain the representations. Next, we evaluate the quality of our pre-trained representations using a K-nearest neighbor (KNN) classifier [164] following the setup in [22, 25, 180]. To validate the knowledge transfer of the learned representations, we adopt the metrics from the SCL literature [20, 109]. Let $a_{\tau,i}$ denote the test accuracy of task i after learning task $\mathcal{T}_\tau$ using

a KNN on frozen pre-trained representations on task $\mathcal{T}_\tau$. More formally, we can define the metrics to evaluate the continually learned representations as follow:

1. **Average accuracy** is the average test accuracy of all the tasks completed until the continual learning of task τ: $A_\tau = \frac{1}{\tau}\sum_{i=1}^{\tau} a_{\tau,i}$
2. **Average Forgetting** is the average performance decrease of each task between its maximum accuracy and accuracy at the completion of training: $F = \frac{1}{T-1}\sum_{i=1}^{T-1} \max_{\tau\in\{1,\dots,T\}} (a_{\tau,i} - a_{T,i})$

Unsupervised Continual Learning

Continuous representation learning with sequential tasks To learn feature representations, contrastive learning [22, 23, 54] maximizes the similarity of representations between the images of the same views (positive pairs) and minimizes the similarity between images of different views (negative pairs). However, these methods require large batches, negative sample pairs [22, 23], or architectural modifications [54, 24], or non-differentiable operators [17], which makes their application difficult for continual learning scenarios. In this work, we focus on SimSiam [25] and BarlowTwins [180], which tackle these limitations and achieve state-of-the-art performance on standard representation learning benchmarks.

SimSiam [25] uses a variant of Siamese networks [13] for learning input data representations. It consists of an encoder network f_Θ, which is composed of a backbone network and is shared across a projection MLP and prediction MLP head $h(\cdot)$. Specifically, SimSiam minimizes the cosine similarity between the output vectors of the projector and the predictor MLP across two different augmentations for an image. Initially, we consider FINETUNE, which is a naive CL baseline that minimizes the cosine-similarity between the projector output ($z^1_{i,\tau} = f_\Theta(x^1_{i,\tau})$) and the predictor output ($p^2_{i,\tau} = h(f_\Theta(x^2_{i,\tau}))$ on a sequence of tasks as follows:

$$\mathcal{L}_{\text{UCL}}^{\text{FINETUNE}} = \frac{1}{2}D(p^1_{i,\tau}, \texttt{stopgrad}(z^2_{i,\tau})) + \frac{1}{2}D(p^2_{i,\tau}, \texttt{stopgrad}(z^1_{i,\tau})), \tag{4.12}$$
$$\text{where} \quad D(p^1_{i,\tau}, z^2_{i,\tau}) = -\frac{p^1_{i,\tau}}{\|p^2_{i,\tau}\|_2} \cdot \frac{z^2_{i,\tau}}{\|z^2_{i,\tau}\|_2},$$

$x^1_{i,\tau}$ and $x^2_{i,\tau}$ are two randomly augmented views of an input example $x_{i,\tau} \in \mathcal{T}_\tau$ and $\|\cdot\|_2$ denotes the ℓ_2-norm. Note that the `stopgrad` is a crucial component in SimSiam to prevent the trivial solutions obtained by Siamese networks. Due to its simplicity and effectiveness, we chose Simsiam to analyze the performance of unsupervised representations for continual learning.

BarlowTwins [180] minimizes the redundancy between the embedding vector components of the distorted versions of an instance while conserving the maximum information inspired from [7]. In particular, the objective function eliminates the SimSiam `stopgrad` component and instead makes the cross-correlation matrix computed between the outputs of two identical networks closer to the identity matrix. Let $\mathcal{C}$ be the cross-correlation matrix between the outputs of two Siamese branches along the batch dimension and Z_1 and Z_2 denote the batch embeddings of the distorted views for all images of a batch from the current task ($x_\tau \in \mathcal{U}_\tau$). The objective function for UCL with finetuning and BarlowTwins can then be defined as:

$$\mathcal{L}_{\text{UCL}}^{\text{FINETUNE}} = \sum_i (1 - \mathcal{C}_{ii})^2 + \quad \lambda \cdot \sum_i \sum_{j\neq i} {\mathcal{C}_{ij}}^2, \quad \text{where} \quad \mathcal{C}_{ij} = \frac{\sum_{\mathcal{B}} z^1_{\mathcal{B},i} z^2_{\mathcal{B},j}}{\sqrt{\sum_{\mathcal{B}} \left(z^1_{\mathcal{B},i}\right)^2}\sqrt{\sum_{\mathcal{B}} \left(z^2_{\mathcal{B},j}\right)^2}}. \tag{4.13}$$

λ is a positive constant trading off the importance of the invariance and redundancy reduction terms of the loss, i and j denote the network's output vector dimensions. Similar to SimSiam, BarlowTwins is simple, easy to implement, and can be applied to existing continual learning strategies with little or no modification.

Preserving representational continuity: A view of existing SCL methods Learning feature representations from labeled instances on a sequence of tasks has been long studied in continual learning. However, the majority of these learning strategies are not directly applicable to UCL. To compare with the regularization-based strategies, we extend *Synaptic Intelligence (SI)* [181] to UCL and consider the online per-synapse consolidation during the entire training trajectory of the unsupervised representations. For architectural-based strategies, we investigate *Progressive Neural Networks (PNN)* [130] and learn the feature representations progressively using the representations learning frameworks.

We also formulate *Dark Experience Replay (DER)* [15] for UCL. DER for SCL alleviates catastrophic forgetting by matching the network logits across a sequence of tasks during the optimization trajectory. Notably, the loss for SCL-DER can be defined as follow:

$$\mathcal{L}_{\text{SCL}}^{\text{DER}} = \mathcal{L}_{\text{SCL}}^{\text{FINETUNE}} + \ \alpha \cdot \mathbb{E}_{(x,p)\sim\mathcal{M}}\big[\, \|\text{softmax}(p) - \text{softmax}(h_\psi(x_{i,\tau}))\|_2^2 \,\big], \tag{4.14}$$

where $p = h_{\psi_\tau(x)}$, $\mathcal{L}_{\text{SCL}}^{\text{FINETUNE}}$ denotes the cross-entropy loss on the current task (see Equation 4.11) and random examples are selected using reservoir sampling from the replay-buffer $\mathcal{M}$. Since we do not have access to the labels for UCL, we cannot minimize the aforementioned objective.

Instead, we utilize the output of the projected output by the backbone network to preserve the knowledge of the past tasks over the entire training trajectory. In particular, DER for UCL consists of a combination of two terms. The first term learns the representations using SimSiam from Equation 4.12 or BarlowTwins from Equation 4.13, and the second term minimizes the Euclidean distance between the projected outputs to minimize catastrophic forgetting. More formally, UCL-DER minimizes the following loss:

$$\mathcal{L}_{\text{UCL}}^{\text{DER}} = \mathcal{L}_{\text{UCL}}^{\text{FINETUNE}} + \ \alpha \cdot \mathbb{E}_{(x)\sim\mathcal{M}}\big[\, \|f_{\Theta_\tau}(x) - f_\Theta(x_{i,\tau})\|_2^2 \,\big] \tag{4.15}$$

However, the performance of the rehearsal-based methods is sensitive to the choice of α and often requires supervised training setup, task identities, and boundaries. To tackle this issue, we propose Lifelong Unsupervised Mixup in the subsequent subsection, which interpolates between the current and past task instances to mitigate catastrophic forgetting effectively.

Lifelong unsupervised mixup The standard Mixup [183] training constructs virtual training examples based on the principle of Vicinal Risk Minimization []. In particular, let (x_i, y_i) and (x_j, y_j) denote two random feature-target pairs sampled from the training data distribution and let $(\tilde{x}, \tilde{y})$ denote the interpolated feature-target pair in the vicinity of these examples; mixup then minimizes the following objective:

$$\begin{aligned} \mathcal{L}^{\text{MIXUP}}(\tilde{x}, \tilde{y}) &= \text{CE}\left(h_\psi\left(f_\Theta\left(\tilde{x}\right)\right), \tilde{y}\right), \\ \text{where} \quad \tilde{x} &= \lambda \cdot x_i + (1-\lambda)\cdot x_j \quad \text{and} \quad \tilde{y} = \lambda \cdot y_i + (1-\lambda)\cdot y_j. \end{aligned} \tag{4.16}$$

$\lambda \sim \text{Beta}(\alpha, \alpha)$, for $\alpha \in (0, \infty)$. In this work, we focus on lifelong self-supervised learning and propose Lifelong Unsupervised Mixup (LUMP) that utilizes mixup for UCL by incorporating the instances stored in the replay buffer from the previous tasks into the vicinal distribution. In particular, LUMP

interpolates between the examples of the current task $(x_{i,\tau}) \in \mathcal{U}_\tau$ and random examples selected using uniform sampling from the replay buffer, which encourages the model to behave linearly across a sequence of tasks. More formally, LUMP minimizes the objective in Equation 4.12 and Equation 4.13 on the following interpolated instances $\tilde{x}_{i,\tau}$ for the current task τ:

$$\tilde{x}_{i,\tau} = \lambda \cdot x_{i,\tau} + (1 - \lambda) \cdot x_{j,\mathcal{M}}, \tag{4.17}$$

where $x_{j,\mathcal{M}} \sim \mathcal{M}$ denotes the example selected using uniform sampling from replay buffer $\mathcal{M}$. The interpolated examples not only augment the past tasks' instances in the replay buffer but also approximate a regularized loss minimization [184]. During UCL, LUMP enhances the robustness of learned representation by revisiting the attributes of the past task that are similar to the current task. Recently, [73, 89, 155, 140] also employed mixup for contrastive learning. Our work is different from these existing works in that our objective is different, and we focus on unsupervised continual learning. To this end, LUMP successively mitigates catastrophic forgetting and learns discriminative & human-perceptual features over the current state-of-the-art SCL strategies (see Table 4.8 and Figure 4.13).

4.2.4 Experimental Results

Experimental setup

Baselines. We compare with multiple supervised and unsupervised continual learning baselines across different categories of continual learning methods.

1. **Supervised continual learning.** FINETUNE is a vanilla supervised learning method trained on a sequence of tasks without regularization or episodic memory, and MULTITASK optimizes the model on complete data. For regularization-based CL methods, we compare against SI [181] and AGEM [19]. We include PNN [130] for architecture-based methods. Lastly, we consider GSS [4] that populates the replay buffer using solid-angle minimization and DER [15] matches the network logits sampled through the optimization trajectory for rehearsal during continual learning.

2. **Unsupervised continual learning.** We consider the unsupervised variants of various SCL baselines to show the utility of the unsupervised representations for sequential learning. Specifically, we use SIMSIAM [25] and BARLOWTWINS [180], which are the state-of-the-art representational learning techniques for learning the unsupervised continual representations. We compare with FINETUNE and MULTITASK following the supervised learning baselines, and SI [181], PNN [130] for unsupervised regularization and architecture CL methods respectively. For rehearsal-based methods, we compare with the UCL variant of DER [15].

Datasets. We compare the performance of SCL and UCL on various continual learning benchmarks using single-head ResNet-18 [55] architecture. **Split CIFAR-10** [77] consists of two random classes out of the ten classes for each task. **Split CIFAR-100** [77] consists of five random classes out of the 100 classes for each task. **Split Tiny-ImageNet** is a variant of the ImageNet dataset [28] containing five random classes out of the 100 classes for each task with the images sized 64×64 pixels.

Training and evaluation setup. We follow the hyperparameter setup of [15] for all the SCL strategies and tune them for the UCL representation learning strategies. All the learned representations are evaluated with KNN classifier [164] across three independent runs. Further, we use the hyper-parameters

Table 4.8: **Accuracy and forgetting** of the learnt representations on Split CIFAR-10, Split CIFAR-100 and Split Tiny-ImageNet on Resnet-18 architecture with KNN classifier [164]. All the values are measured by computing mean and standard deviation across three trials. The best and second-best results are highlighted in **bold** and underline, respectively.

	Method	Split CIFAR-10		Split CIFAR-100		Split Tiny-ImageNet	
		Accuracy	Forgetting	Accuracy	Forgetting	Accuracy	Forgetting
	Supervised Continual Learning						
	Finetune	82.87 (± 0.47)	14.26 (± 0.52)	61.08 (± 0.04)	31.23 (± 0.41)	53.10 (± 1.37)	33.15 (± 1.22)
	PNN [130]	82.74 (± 2.12)	–	66.05 (± 0.86)	–	64.38 (± 0.92)	–
	Si [181]	85.18 (± 0.65)	11.39 (± 0.77)	63.58 (± 0.37)	27.98 (± 0.34)	44.96 (± 2.41)	26.29 (± 1.40)
	A-gem [19]	82.41 (± 1.24)	13.82 (± 1.27)	59.81 (± 1.07)	30.08 (± 0.91)	60.45 (± 0.24)	24.94 (± 1.24)
	Gss [4]	89.49 (± 1.75)	7.50 (± 1.52)	70.78 (± 1.67)	21.28 (± 1.52)	70.96 (± 0.72)	14.76 (± 1.22)
	Der [15]	91.35 (± 0.46)	5.65 (± 0.35)	79.52 (± 1.88)	12.80 (± 1.47)	68.03 (± 0.85)	17.74 (± 0.65)
	Multitask	97.77 (± 0.15)	–	93.89 (± 0.78)	–	91.79 (± 0.46)	–
	Unsupervised Continual Learning						
SimSiam	Finetune	90.11 (± 0.12)	5.42 (± 0.08)	75.42 (± 0.78)	10.19 (± 0.37)	71.07 (± 0.20)	9.48 (± 0.56)
SimSiam	PNN [130]	90.93 (± 0.22)	–	66.58 (± 1.00)	–	62.15 (± 1.35)	–
SimSiam	Si [181]	**92.75 (± 0.06)**	1.81 (± 0.21)	80.08 (± 1.30)	5.54 (± 1.30)	72.34 (± 0.42)	8.26 (± 0.64)
SimSiam	Der [15]	91.22 (± 0.30)	4.63 (± 0.26)	77.27 (± 0.30)	9.31 (± 0.09)	71.90 (± 1.44)	8.36 (± 2.06)
SimSiam	Lump	91.00 (± 0.40)	2.92 (± 0.53)	**82.30 (± 1.35)**	4.71 (± 1.52)	**76.66 (± 2.39)**	3.54 (± 1.04)
SimSiam	Multitask	95.76 (± 0.08)	–	86.31 (± 0.38)	–	82.89 (± 0.49)	–
BarlowTwins	Finetune	87.72 (± 0.32)	4.08 (± 0.56)	71.97 (± 0.54)	9.45 (± 1.01)	66.28 (± 1.23)	8.89 (± 0.66)
BarlowTwins	PNN [130]	87.52 (± 0.33)	–	57.93 (± 2.98)	–	48.70 (± 2.59)	–
BarlowTwins	Si [181]	90.21 (± 0.08)	2.03 (± 0.22)	75.04 (± 0.63)	7.43 (± 0.67)	56.96 (± 1.48)	17.04 (± 0.89)
BarlowTwins	Der [15]	88.67 (± 0.24)	2.41 (± 0.26)	73.48 (± 0.53)	7.98 (± 0.29)	68.56 (± 1.47)	7.87 (± 0.44)
BarlowTwins	Lump	90.31 (± 0.30)	**1.13 (± 0.18)**	80.24 (± 1.04)	**3.53 (± 0.83)**	72.17 (± 0.89)	**2.43 (± 1.00)**
BarlowTwins	Multitask	95.48 (± 0.14)	–	87.16 (± 0.52)	–	82.42 (± 0.74)	–

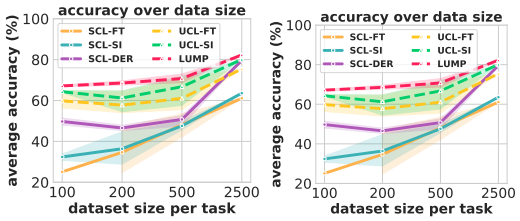
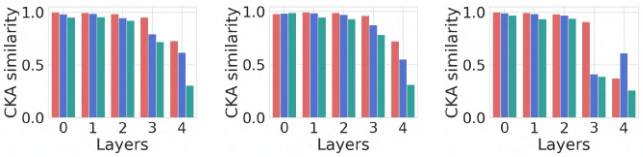


Figure 4.11: **Evaluation on Few-shot training** for Split CIFAR-100 across different number of training instances per task. The results are measured across three independent trials.



(a) Finetune (b) Si (c) Der

Figure 4.12: **CKA Feature similarity** between two independent UCL models (**red**), two independent SCL models (**blue**), and UCL and SCL model (**green**) for different strategies on Split CIFAR-100 test distribution.

obtained by SimSiam for training UCL strategies with BarlowTwins to analyze the sensitivity of UCL to hyper-parameters and for a fair comparison between different methods. We train all the UCL methods for 200 epochs and evaluate with the KNN classifier [164].

Table 4.9: **Comparison of accuracy** on out of distribution datasets using a KNN classifier [164] on pre-trained SCL and UCL representations. We consider MNIST [84], Fashion-MNIST (FMNIST) [165], SVHN [110] as out of distribution for Split CIFAR-100 and Split CIFAR-10. All the values are measured by computing mean and standard deviation across three trials. The best and second-best results are highlighted in **bold** and underline respectively.

	In-class	Split CIFAR-10				Split CIFAR-100			
	Out-of-class	MNIST	FMNIST	SVHN	CIFAR-100	MNIST	FMNIST	SVHN	CIFAR-10
	Supervised Continual Learning								
	Finetune	86.42 (± 1.11)	74.47 (± 0.84)	41.00 (± 0.85)	17.42 (± 0.96)	75.02 (± 3.97)	62.37 (± 3.20)	38.05 (± 0.73)	39.18 (± 0.83)
	Si [181]	87.08 (± 0.79)	76.41 (± 0.81)	42.62 (± 1.31)	19.14 (± 0.91)	79.96 (± 2.63)	63.71 (± 1.36)	40.92 (± 1.64)	40.41 (± 1.71)
	A-gem [19]	86.07 (± 1.94)	74.74 (± 3.21)	37.77 (± 3.49)	16.11 (± 0.38)	77.56 (± 3.21)	64.16 (± 2.29)	37.48 (± 1.73)	37.91 (± 1.33)
	Gss [4]	70.36 (± 3.54)	69.20 (± 2.51)	33.11 (± 2.26)	18.21 (± 0.39)	76.54 (± 0.46)	65.31 (± 1.72)	35.72 (± 2.37)	49.41 (± 1.81)
	Der [15]	80.32 (± 1.91)	70.49 (± 1.54)	41.48 (± 2.76)	17.72 (± 0.25)	87.71 (± 2.23)	75.97 (± 1.29)	50.26 (± 0.95)	59.07 (± 1.06)
	multitask	88.79 (± 1.13)	79.50 (± 0.52)	41.26 (± 1.95)	27.68 (± 0.66)	92.29 (± 3.37)	86.12 (± 1.87)	54.94 (± 1.77)	54.04 (± 3.68)
	Unsupervised Continual Learning								
SimSiam	Finetune	89.23 (± 0.99)	80.05 (± 0.34)	49.66 (± 0.81)	34.52 (± 0.12)	85.99 (± 0.86)	76.90 (± 0.11)	50.09 (± **1.41**)	57.15 (± 0.96)
SimSiam	Si [181]	**93.72** (± **0.58**)	**82.50** (± **0.51**)	**57.88** (± **0.16**)	**36.21** (± **0.69**)	91.50 (± 1.26)	80.57 (± 0.93)	**54.07** (± **2.73**)	60.55 (± **2.54**)
SimSiam	Der [15]	88.35 (± 0.82)	79.33 (± 0.62)	48.83 (± 0.55))	30.68 (± 0.36)	87.96 (± 2.04)	76.21 (± 0.63)	47.70 (± 0.94)	56.26 (± 0.16)
SimSiam	Lump	91.03 (± 0.22)	80.78 (± 0.88)	45.18 (± 1.57)	31.17 (± 1.83)	**91.76** (± **1.17**)	**81.61** (± **0.45**)	50.13 (± 0.71)	**63.00** (± **0.53**)
	multitask	90.69 (± 0.13)	80.65 (± 0.42)	47.67 (± 0.45)	39.55 (± 0.18)	90.35 (± 0.24)	81.11 (± 1.86)	52.20 (± 0.61)	70.19 (± 0.15)
BarlowTwins	Finetune	86.86 (± 1.62)	78.37 (± 0.74)	44.64 (± 2.39)	28.03 (± 0.52)	76.08 (± 2.86)	76.82 (± 0.83)	42.95 (± 0.90)	53.12 (± 0.13)
BarlowTwins	Si [181]	90.31 (± 0.69)	80.58 (± 0.68)	49.18 (± 0.51)	31.80 (± 0.4)	85.24 (± 0.99)	78.82 (± 0.67)	45.18 (± 1.37)	53.99 (± 0.56)
BarlowTwins	Der [15]	85.15 (± 2.19)	77.96 (± 0.59)	45.68 (± 0.93)	27.83 (± 0.86)	78.08 (± 1.95)	76.67 (± 0.68)	44.58 (± 1.01)	53.24 (± 0.82)
BarlowTwins	Lump	88.73 (± 0.54)	81.69 (± 0.45)	51.53 (± 0.41)	31.53 (± 0.36)	90.22 (± 1.39)	81.28 (± 0.91)	50.24 (± 0.95)	60.76 (± 0.87)
	multitask	88.63 (± 1.38)	79.49 (± 0.29)	49.24 (± 2.44)	36.33 (± 0.29)	86.98 (± 1.70)	79.40 (± 1.10)	50.19 (± 0.81)	49.50 (± 0.38)

Table 4.10: ℓ_2 distance between UCL parameters after completion of training.

Model	Finetune	Si	Der	MultiTask
Finetune	60.00 (± 1.70)			
Si	76.46 (± 0.48)	92.35 (± 0.61)		
der	55.60 (± 1.42)	75.54 (± 0.97)	48.76 (± 1.54)	
Multitask	61.32 (± 0.59)	79.95 (± 0.40)	57.90 (± 0.86)	61.42 (± 0.78)

Table 4.11: ℓ_2 distance between SCL paraneters after completion of training.

Model	Finetune	Si	Der	MultiTask
Finetune	183.31 (± 0.10)			
Si	206.16 (± 0.28)	226.05 (± 0.13)		
der	202.61 (± 0.46)	224.78 (± 0.75)	219.06 (± 0.27)	
Multitask	258.12 (± 0.26)	277.30 (± 0.69)	271.48 (± 0.45)	314.84 (± 0.92)

Quantitative results

Evaluation on SimSiam. Table 4.8 shows the evaluation results for supervised and unsupervised representations learned by SimSiam [25] across various continual learning strategies. In all cases, continual learning with unsupervised representations achieves significantly better performance than supervised representations with substantially lower forgetting. For instance, Si with UCL obtains better performance and 68%, 54%, and 44% lower forgetting relative to the best-performing SCL strategy on Split CIFAR-10, Split CIFAR-100, and Split Tiny-ImageNet, respectively. Surprisingly, Finetune with UCL achieves higher performance and significantly lower forgetting in comparison to all SCL strategies except Der. Furthermore, Lump improves upon the UCL strategies: 2.8% and 5.9% relative increase in accuracy and 15% and 57.1% relative decrease in forgetting on Split CIFAR-100 and Split Tiny-ImageNet, respectively.

Evaluation on BarlowTwins. To verify that unsupervised representations are indeed more robust to catastrophic forgetting, we train BarlowTwins [180] on a sequence of tasks. We notice that the representations learned with BarlowTwins substantially improve the accuracy and forgetting over SCL:

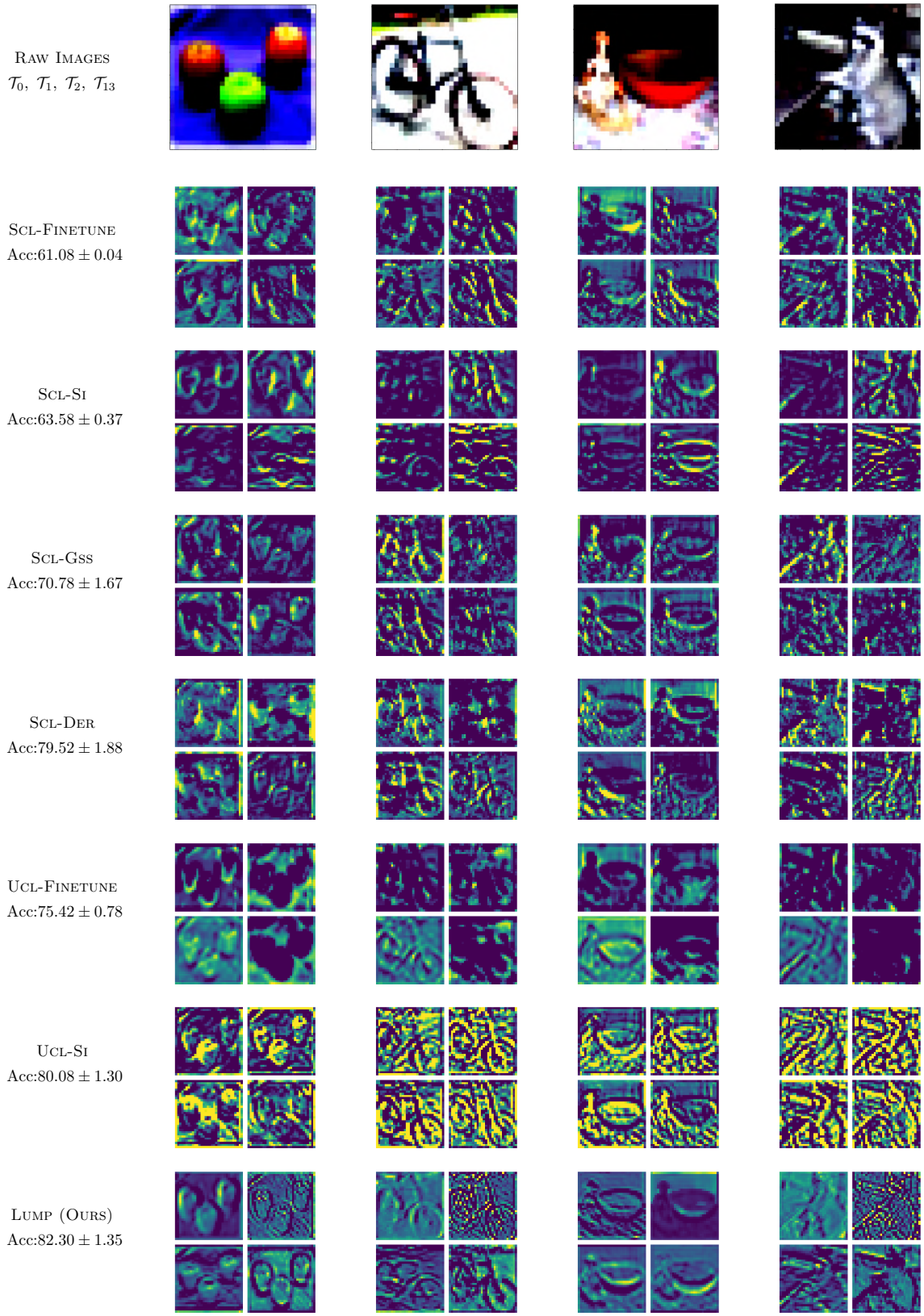


Figure 4.13: **Visualization of feature maps** for the second block representations learnt by SCL and UCL strategies (with Simsiam) for Resnet-18 architecture after the completion of continual learning for Split CIFAR-100 dataset ($n = 20$). The accuracy is the mean across three runs for the corresponding task.

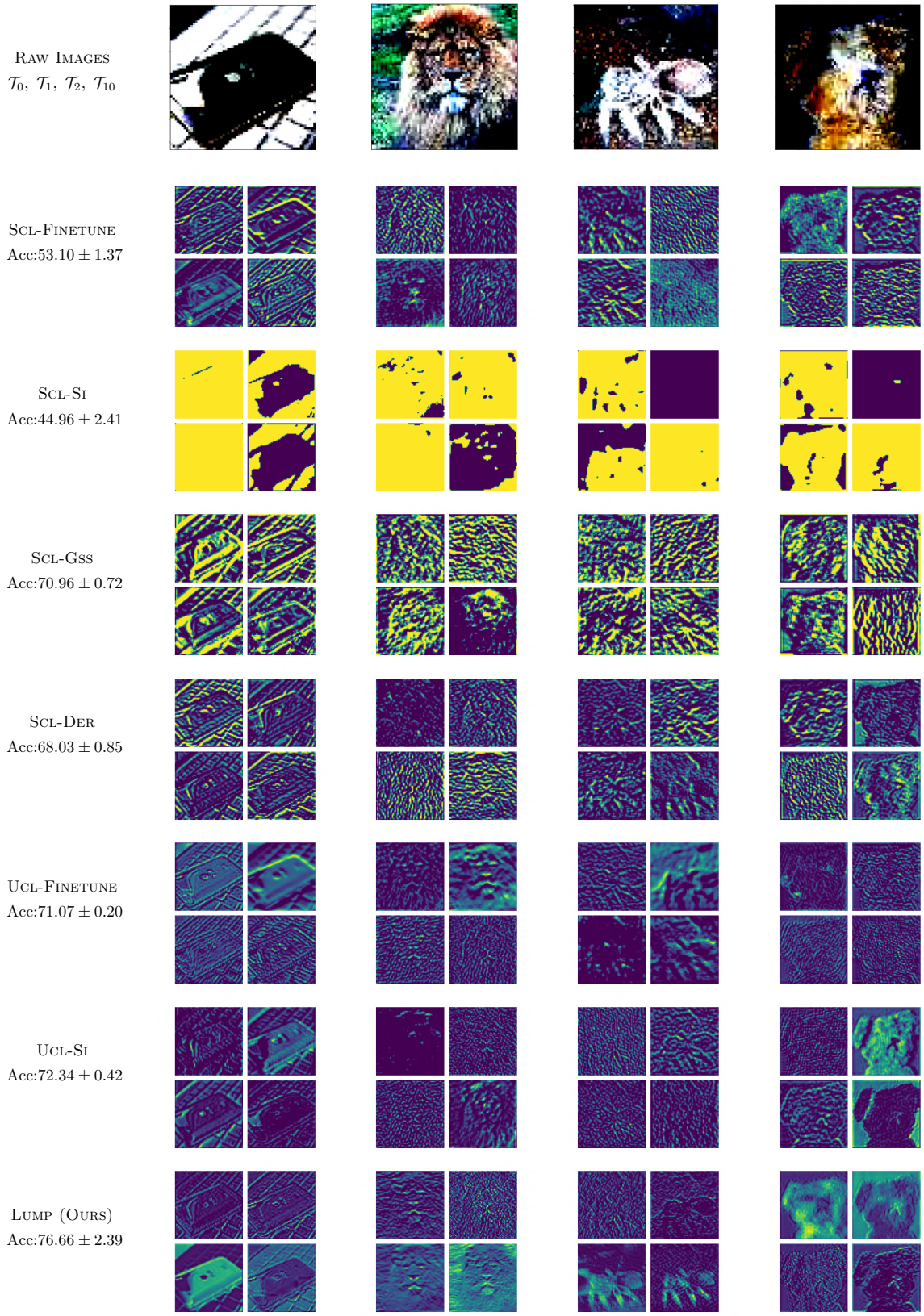


Figure 4.14: **Visualization of feature maps** for the second block representations learnt by SCL and UCL strategies (with Simsiam) for Resnet-18 architecture after the completion of continual learning for Split Tiny-ImageNet dataset ($n = 20$). The accuracy is the mean across three runs for the corresponding task.

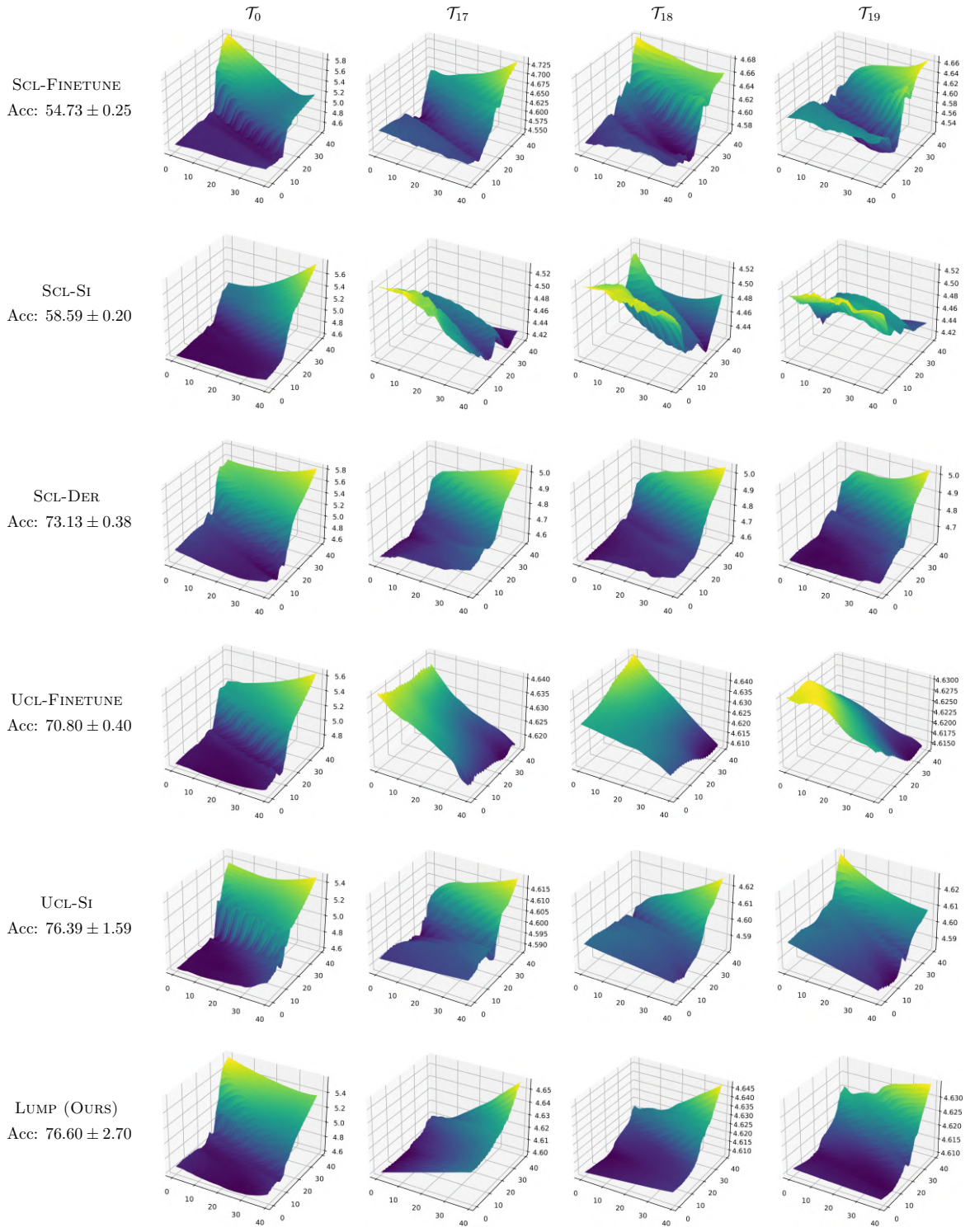


Figure 4.15: **Loss landscape visualization** of $\mathcal{T}_0$ after the completion of training on task $\mathcal{T}_0$, $\mathcal{T}_{17}, \mathcal{T}_{18}$, and $\mathcal{T}_{19}$ for Split CIFAR-100 dataset on ResNet-18 architecture. We use Simsiam for UCL methods.

71.4%, 69.7% and 73.2% decrease in forgetting with FINETUNE on Split CIFAR-10, Split CIFAR-100 and Split Tiny-ImageNet respectively. Similarly, we observe that SI and DER are more robust to catastrophic forgetting; however, PNN underperforms on complicated tasks since feature accumulation using adaptor

modules is insufficient to construct useful representations for current task adaptation. Interestingly, representations learned with BarlowTwins achieve lower forgetting for FINETUNE, DER, and LUMP than SimSiam with comparable accuracy across all the datasets.

Evaluation on Few-shot training. Figure 4.11 compares the effect of few-shot training on UCL and SCL, where each task has a limited number of training instances. Specifically, we conduct the experimental evaluation using 100, 200, 500, and 2500 training instances for each task in split CIFAR-100 dataset. Surprisingly, we observe that the gap in average accuracy between SCL and UCL methods widens with a decrease in the number of training instances. Note that UCL decreases the accuracy by 15.78%p on average with lower forgetting when the number of training instances decreases from 2500 to 100; whereas SCL obtains a severe 32.21%p deterioration in accuracy. We conjecture that this is an outcome of the discriminative feature embeddings learned by UCL, which discriminates all the images in the dataset and captures more than class-specific information, as also observed in [33]. Furthermore, LUMP improves the performance over all the baselines with a significant margin across all few-shot experiments.

Evaluation on OOD datasets. We evaluate the learnt representations on various out-of-distribution (OOD) datasets in Table 4.9 to measure their generalization to unseen data distributions. In particular, we conduct the OOD evaluation on MNIST [84], Fashion-MNIST (FMNIST) [165], SVHN [110], CIFAR-10 and CIFAR-100 [77] using a KNN classifier [164]. We observe that unsupervised representations outperform the supervised representations in all cases across all the datasets. In particular, the UCL representations learned with Simsiam, and SI on Split-CIFAR-10 improve the absolute performance over the best-performing SCL strategy by 4.58%, 6.09%, 15.26%, and 17.07% on MNIST, FMNIST, SVHN, and CIFAR-100 respectively. Further, LUMP trained on Split-CIFAR-100 outperforms SI across all datasets and obtains comparable performance with Split CIFAR-10 dataset.

Qualitative analysis

Similarity in feature and parameter space. We analyze the similarity between the representations learned between (i) Two independent UCL models, (ii) Two independent SCL models (iii) SCL and UCL models using centered kernel alignment (CKA) [76] in Figure 4.12, which provides a score between 0 and 1 measuring the similarity between a pair of hidden representations. For two representations $\Theta_1 : \mathcal{X} \rightarrow \mathbb{R}^{d_1}$ and $\Theta_2 : \mathcal{X} \rightarrow \mathbb{R}^{d_1}$, $\text{CKA}(\Theta_1, \Theta_2) = \frac{||\text{Cov}(\Theta_1(x), \Theta_2(x))||_F^2}{||\text{Cov}(\Theta_1(x))||_F \cdot ||\text{Cov}(\Theta_2(x))||_F}$, where covariances are with respect to the test distribution. Additionally, we measure the ℓ_2 distance [111] between the parameters of two independent UCL models (see Table 4.10) and two independent SCL models (see Table 4.11). First, we observe that the representations learned by two independent UCL methods have a high feature similarity and lower ℓ_2 distance compared to the two independent SCL methods, demonstrating UCL representations' robustness. Second, we note that the representations between any two independent models are highly similar in the lower layers indicating that they learn similar high-level features, including edges and shapes; however, the features are dissimilar for the higher modules. Lastly, we see that the representations between a UCL and SCL model are similar in the lower layers but diverge in the higher layers across all CL strategies.

Visualization of feature space. Next, we visualize the learned features to dissect further the representations learned by UCL and SCL strategies. Figure 4.13 shows the visualization of the latent feature maps for tasks $\mathcal{T}_0$ and $\mathcal{T}_{13}$ after the completion of continual learning. For $\mathcal{T}_0$, we observe that the

SCL methods are prone to catastrophic forgetting, as the features appear noisy and do not have coherent patterns. In contrast, the features learned by UCL strategies are perceptually relevant and robust to catastrophic forgetting, with LUMP learning the most distinctive features. Similar to $\mathcal{T}_0$, we observe that the UCL features are more relevant and distinguishable than SCL for $\mathcal{T}_{13}$. Note that we randomly selected the examples and feature maps for all visualizations.

Loss landscape visualization. To gain further insights, we visualize the loss landscape of task $\mathcal{T}_0$ after the completion of training on task $\mathcal{T}_0$ and $\mathcal{T}_{19}$ for various UCL and SCL strategies in Figure 4.15. We measure the cross-entropy loss for all methods with a randomly initialized linear classifier for a fair evaluation of two different directions. We use the visualization tool from [93] that searches the task loss surface by repeatedly adding random perturbations to model weights. We observe that the loss landscape after $\mathcal{T}_0$ looks quite similar across all the strategies since the forgetting does not exist yet. However, after training $\mathcal{T}_{19}$, there is a clear difference with the UCL strategies obtaining a flatter and smoother loss landscape because UCL methods are more stable and robust to the forgetting, which hurts the loss landscapes of past tasks for SCL. It is important to observe that LUMP obtains a smoother landscape than other UCL strategies, demonstrating its effectiveness.

4.2.5 Summary

This work attempts to bridge the gap between unsupervised representation learning and continual learning. In particular, we establish the following findings for unsupervised continual learning.

Surpassing supervised continual learning. Our empirical evaluation across various CL strategies and datasets shows that UCL representations are more robust to catastrophic forgetting than SCL representations. Furthermore, we notice that UCL generalizes better to OOD tasks and achieves stronger performance on few-shot learning tasks. We propose *Lifelong unsupervised mixup (*LUMP*)*, which interpolates the unsupervised instances between the current task and past task and obtains higher performance with lower catastrophic forgetting across a wide range of tasks.

Dissecting the learned representations. We conduct a systematic analysis to understand the differences between the representations learned by UCL and SCL strategies. By investigating the similarity between the representations, we observe that UCL and SCL strategies have high similarities in the lower layers but are dissimilar in the higher layers. We also show that UCL representations learn coherent and discriminative patterns and smoother loss landscape than SCL.

Limitations and future work. In this work, we do not consider the high-resolution tasks for CL. We intend to evaluate the forgetting of the learnt representations on ImageNet [28] in future work, since UCL shows lower catastrophic forgetting and representation learning has made significant progress on ImageNet over the past years. In follow-up work, we intend to conduct further analysis to understand the behavior of UCL and develop sophisticated methods to continually learn unsupervised representations under various setups, such as class-incremental or task-agnostic CL.

4.3 Efficient Video Representation Learning via Masked Video Modeling with Motion-centric Token Selection [58]

4.3.1 Motivation

Massive amounts of video data flood the web and media every single day with the rapid growth of portable devices equipped with cameras, such as google glasses, smartphones, UAVs, and robots. However, direct utilization of user-generated video data for solving target task problems is nontrivial as the annotation process is time-consuming and expensive. One potentially reasonable approach to tackle this problem is to learn generic representations from unlabeled video data streams that can transfer to downstream video-based tasks. Unsupervised Video Representation Learning [40, 121, 51, 122, 116] methods allow learning spatial and temporal features from input video frames in a self-supervised manner without any human-annotated clues. A caveat to such pre-training for video tasks is that, unlike representation learning on image-based tasks capturing static information of objects in instances, video-based tasks involve temporal causality, where successive frames are strongly correlated in their semantics.

Recently, a nascent self-supervised learning paradigm, Masked Image Modeling (MIM) [53, 167], significantly outperforms the previous representation learning methods on various downstream tasks. MIM aims to predict masked regions in input images by solving a pixel regression problem, in which the model splits each image into small patches and generates masks to zero out of a subset of patches. Most of these approaches use Vision transformer backbones [34, 100] due to their compatibility with patchwise operations.

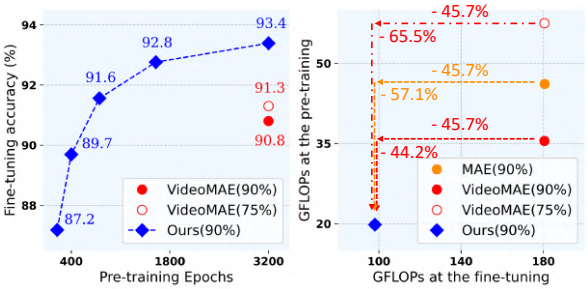
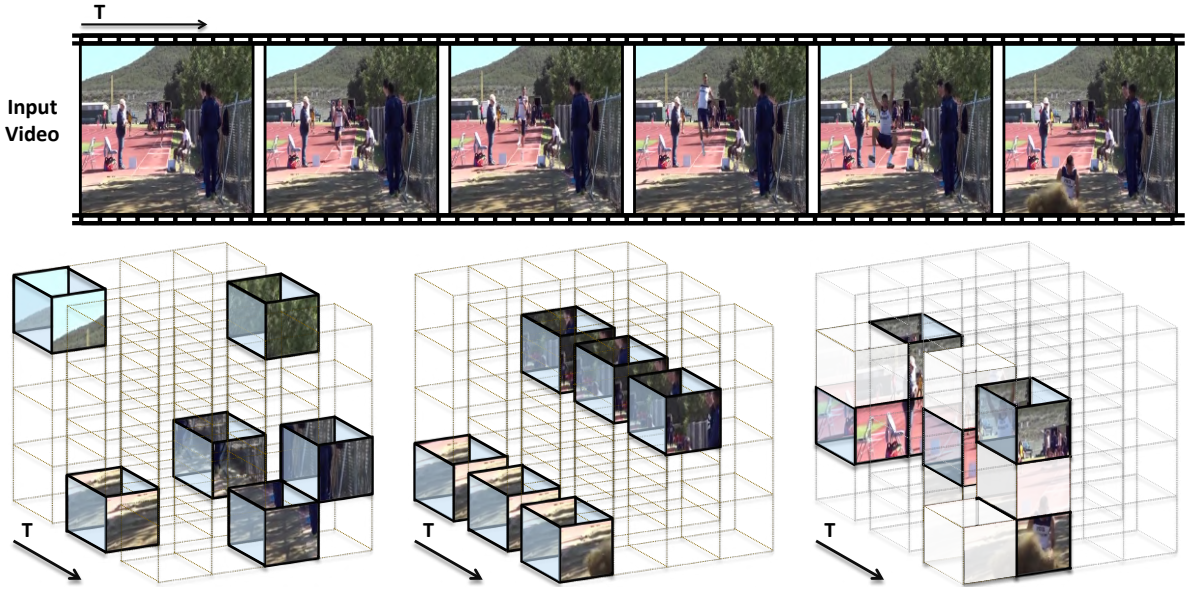


Figure 4.16: **Performance and efficiency of our VideoMS**. **(Left)** VideoMS significantly outperforms the SoTA model (VideoMAE [154]) over UCF101 [149], even at significantly fewer training epochs. We additionally report the results of VideoMAE [154] with a lower masking ratio (75%) during pre-training. **(Right)** Our VideoMS drastically reduces the computational cost during pre-training and fine-tuning compared to the SoTA Masked Video Modeling methods, VideoMAE [154] and MAE [39].

Such a simple and intuitive strategy of MIMs was also exploited in video representation learning. Masked Video Modeling (MVM) [154, 39], which learns to reconstruct randomly masked spatiotemporal areas in video clips at each minibatch, has recently shown impressive performance on a variety of downstream tasks. Yet, critical challenges still remain in efficiently utilizing spatiotemporal information from real-world videos: (1) Tokens (a pair of two temporally successive patches in the same space) from videos are not equally valuable to reconstruct, as their informativeness depends on adjacent frames, unlike in image representation learning. (2) Training on the entire video is almost infeasible without access to a large amount of computes. Prior MVM approaches that reconstruct the whole video heavily rely on numerous GPUs without considering each object's changes and redundancy. For example, VideoMAE [154] takes about 27 hours to pre-train for 800 epochs with a ViT-B backbone using 64 NVIDIA V100 (32GB) GPUs, and MAE [39] takes about 35.8 hours to pre-train for 800 epochs with a ViT-L backbone using 128 NVIDIA A100 (80GB) GPUs.



(a) Random [39] **(b) Space-only** [154] **(c) Motion-centric (Ours)**

Figure 4.17: **Illustration of Motion-centric Token Selection. (a-b)** MVM methods forward a few tokens in arriving video frames into the encoder based on fully random and time-only random selection, and then the decoder *reconstructs entire tokens*. **(c)** Unlike prior works, our proposed motion-centric masking strategy only recovers key tokens related to moving objects (visible). To do this, we use encoder features extracted from randomly selected tokens among them (visible and colored).

Effective Batch Size	128	256	512	1028
VideoMAE [154]	258.4 GB	486.4 GB	*Out-of-Memory*	*Out-of-Memory*
VideoMS (Ours)	**60.0 GB**	**91.2 GB**	**151.2 GB**	**273.6 GB**

Table 4.12: Memory allocation comparison using ViT-B in the pre-training phase (one node, A100×8 GPUs). VideoMAE suffers from *Out-Of-Memory* issue when the batch size is larger than 256.

To tackle these inefficiencies for efficient video representation learning, we propose a token-selective training algorithm, which we refer to as *Motion-centric Token Selection (VideoMS)*. We aim to selectively find helpful visual tokens based on the *distance* across adjacent tokens in the embedding space to detect significant changes in objects or status while discarding tokens for redundant frames and meaningless backgrounds as shown in Figure 4.17(c). Thus, *VideoMS* drastically reduces memory usage by back-propagating only to a few selected tokens of high importance as shown in Figure 4.16. Our method uses **5.3 times smaller memory** (**81**% ↓) with the effective batch size of 256 and allows to use up to **4 times larger batch size** using NVIDIA A100 GPUs compared to VideoMAE [154] (Please see Table 4.12). Further, most existing VRL methods load video frames within uniform distances in each minibatch clip, neglecting high diversity over frames and noise in uncurated videos. To enhance frame selection, we extend our tokenwise masking strategy to the frame level, constructing informative minibatch video clips by inducing to gather frames that contain various changes over time.

- We propose an *efficient* video self-supervised learning method that promptly selects the most informative tokens based on the objects' motion, to avoid wasteful training on uninformative spatiotemporal regions in the given videos.
- We demonstrate that our proposed method outperforms existing SoTA VRL methods, achieving remarkable performance improvements while significantly reducing the computational costs and

memory requirement.

- We further suggest an online video frame selection to instantly and adaptively discard noisy and redundant frames from incoming videos, leading the model to help learn objects' dynamic movements.

4.3.2 Related Work

A video stream consists of a massive amount of correlated frames exhibiting spatial and temporal locality. Annotating video data requires significant human efforts, and thus it is impractical to fully label all the frames due to extremely high cost from both time and financial standpoints. To capture spatiotemporal representation from video datasets without any given labels, many recent works have focused on the video representation learning problem from diverse perspectives.

Contrastive Video Representation Learning One of the main research directions in video representation learning is contrastive learning with various constraints [40, 51, 122, 157, 50]. Odd-one-out learning [40] learns to distinguish odd video subsequences which are composed of permuted order of frames against normal video streams in a self-supervised fashion. *Han et al.* [50] introduce the Dense Predictive Coding (DFC) framework for video self-supervised learning that predicts future frame representation from current spatiotemporal features from 3D-Resnet. They adopt a modified contrastive loss, considering pairs with semantic, spatial, and temporal negatives. Motivated by an observation that the human visual system can rapidly and precisely distinguish different motions, *Wang et al.* [157] propose a new video representation learning method that model learns to predict video pace given a few clustered speed candidates. CoCLR [51] and CVRL [122] tackle video representation learning by proposing modified InfoNCE [115] losses. On the other hand, CoCLR collaboratively trains RGB and optical flow models with shared positives, and CVRL carefully utilizes various augmentation technologies which are crucial for capturing spatiotemporal representation from video data.

Masked Video Modeling Self-supervised learning with Masked Image Modeling (MIM) [53, 167, 66] has recently become a popular alternative to contrastive learning for its ability to learn rich representations without having to define negative examples. MIM aims to learn image representations by solving pixel regression problems in regions of an image that are zeroed out through random [53, 167] or attention-based [66] masking strategies. Inspired by MIM, several recent works on video representation learning [189, 159, 154, 39] present spatiotemporal masking strategies for representation learning given video streams. To capture spatial representation and temporal dynamics for unsupervised video streams, MAE [39] and VideoMAE [154] extend a masked image autoencoder to mask partial regions in arriving video clips via random and space-only random sampling, respectively. They find that spatiotemporal inductive bias in video clips helps a decoder predict input pixels in masked regions, allowing a higher masking ratio ($\sim$ 90%) than MIM ($\sim$ 60% [167] and $\sim$ 75% [53]) on image self-supervised learning. BEVT [159] proposes to jointly train an image-level masked autoencoder and a masked video modeling method by sharing weights of the encoder, formulated with Video Swin [101]. They resort to random sampling given spatiotemporal inductive bias, which can be a good approximator with stochasticity during data-driven training. Nevertheless, selecting random spatiotemporal tokens to reconstruct for masked video modeling is inefficient since embedded tokens in video frames are not equally important, especially since the informativeness of each token is affected by the adjacent frames.

Input Selective Training As training benchmark datasets often have massive scales and contain a lot of redundant and less meaningful instances, various approaches [176, 38, 11, 117] have been developed to utilize only the important instances among entire datasets. A few works focus on instance-level selection for training. *Borsos et al.* [11] presents a bilevel optimization method to select coreset from arriving stream data. OCS [176] proposes a gradient-based coreset selection criterion that selects only the beneficial instances for a task among incoming inputs, dropping redundant and noisy instances in an online fashion. Several recent works have discussed patch selection techniques for efficient image recognition with Vision Transformers (ViT). AdaVit [107] introduces parametric decision networks for patch/head/block selection for training ViT. ATS [38] samples meaningful tokens based on the significant score, computed by multiplication of attentions and value matrices. AttMask [66] and A-Vit [171] adopt parameter-free token masking strategies utilizing the average of multi-head attentions. However, those methods have no means to capture the temporal relatedness across frames in video tasks. Very recently, *Wang et al.* [158] propose an efficient fine-tuning framework for video tasks that sequentially perform temporal and spatial token selection given pre-trained ViT backbones. They introduce a lightweight scorer network to learn the importance score at each token and select the most informative regions in incoming videos. *Park et al.* [117] suggest the greedy K-center search that iteratively finds the most distant patches in the geometrical space from video clips. These approaches limit their selection processes to fine-tuning and are not suitable for pre-training. On the other hand, our proposed method, VideoMS, drastically reduces the required computational cost and memory from both stages through motion-centric token selection.

4.3.3 Approach

We first introduce Masked Image Modeling (MIM) for image representation learning and then extend it to our video representation learning scheme, *Masked Video Modeling (MVM)*. Then we discuss several crucial challenges in recently proposed masked video modeling methods.

Masked Image Modeling

Learning to reconstruct intentionally corrupted data with masking is broadly utilized as means of representation learning in Natural Language Processing [30, 147, 48, 148] and has demonstrated its efficacy and power in broad research problems. In vision tasks, Masked Image Modeling [53, 167] aims to learn representations of the input images by solving the regression problem in which a model predicts RGB pixel values in randomly zeroed patch regions in images. The model first splits a high-resolution input image into equally-sized square patches and fills the values of a few patches to be zeros according to a pre-defined masking ratio. A transformer encoder extracts visual features from unmasked patches with positional information, and a decoder tries to reconstruct the original input images by predicting the RGB values of missing image patches from encoded features. Let a self-supervised model f be formulated into an encoder-decoder framework $f(\cdot) = D(E(\cdot))$. Given the dataset $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^{N}, \boldsymbol{x}_i \in \mathbb{R}^{C\times H\times W}$ with the number of instance N, the channel C, the height H, and the width W, the objective function is formulated as follows:

$$
\begin{aligned}
\ell\left(\boldsymbol{x}\right) &= \frac{1}{N}\sum_{i=1}^{N}\left\|f(\boldsymbol{m}_i \otimes \boldsymbol{k}_i; \mathcal{W}) - \boldsymbol{m}_i \otimes \boldsymbol{x}_i\right\|_p \\
&= \frac{1}{N}\sum_{i=1}^{N}\left\|D\left(E\left(\boldsymbol{m}_i \otimes \boldsymbol{k}_i; \mathcal{W}_E\right); \mathcal{W}_D\right) - \boldsymbol{m}_i \otimes \boldsymbol{x}_i\right\|_p, \\
&\text{where} \quad \boldsymbol{k}_i = \texttt{Conv2d}(\boldsymbol{x}_i; \boldsymbol{w}, s) \in \mathbb{R}^{J \times d_k}.
\end{aligned} \tag{4.18}
$$

where $\boldsymbol{k}_i$ indicates a d_k-dimensional patch embedding vector for $\boldsymbol{x}_i$, obtained by a convolutional layer parameterized by $\boldsymbol{w}$ with a stride s. Further, $J = \left\lfloor \frac{H}{s} \right\rfloor \cdot \left\lfloor \frac{W}{s} \right\rfloor$ denotes the number of patches per image and $\|\cdot\|_p$ denotes p-norm. $\mathcal{W}_E$ and $\mathcal{W}_D$ is a set of weights in the encoder and decoder, respectively. $\otimes$ is a dimensionality-preserving vector-matrix multiplication operation. A mask $\boldsymbol{m}_i \in \{0,1\}^J$ is drawn by the binary distribution B with a probability of ρ without replacement, that is $|\boldsymbol{m}_i| = [J \cdot \rho]$. After self-supervised pre-training to minimize Equation 4.18, the encoder transfers the learned representations to various downstream tasks.

Masked Video Modeling

The dominant approach to video representation learning approach has been contrastive learning, but the recent success of MIM has led to a breakthrough in effectively capturing spatiotemporal information from the arriving video stream, which we call Masked Video Modeling [154, 167]. Let $\boldsymbol{v} \in \mathbb{R}^{2\tau \times C \times H \times W}$ be a short clip consisting of 2τ successive frames from the entire video. The model aims to reconstruct partially masked frames in $\boldsymbol{v}$, guided by spatial and temporal relationships between tokens in adjacent frames. The encoder takes tokenized embedding vectors from a pair of successive frames using a 3D convolutional operation. Let $\boldsymbol{k}_i$ be an i-th spatiotemporal embedding vector for a pair of two frames $\boldsymbol{v}[2i : 2i+1]$. Given $\boldsymbol{v}$, we reformulate the loss function of Masked Video Modeling as follows:

$$
\begin{aligned}
\ell\left(\boldsymbol{v}\right) &= \sum_{i=0}^{\tau-1}\left\|f_{\mathcal{W}}(\boldsymbol{m}_i \otimes \boldsymbol{k}_i) - \boldsymbol{m}_i \otimes \boldsymbol{v}[2i : 2i+1]\right\|_p, \\
&\text{where} \quad \boldsymbol{m} = G\left(J,\ \rho,\ \tau\right) \in \{0,1\}^{\tau \times J},
\end{aligned} \tag{4.19}
$$

where $G(\cdot)$ is an arbitrary random masking function depending on the specific policy, for example, random (or agnostic) and space-only (or tube) masking techniques, illustrated in Figure 4.17 (a) and (b).

Unlike the samples from the image dataset, which are permutation-invariant as they are independent of each other, consecutive frames from the video stream inherently have a strong correlation and redundancy. Thus, masked video modeling can enjoy more spatiotemporal inductive bias from other adjacent frames in the input clip, achieving good reconstruction quality even with lessened hints (i.e., a proportion of unmasked tokens). Indeed, MVM allows a much higher masking ratio $1-\rho$ per video against MIM methods. This property is advantageous because masked modeling with a higher masking ratio significantly reduces computations when training the encoder-decoder framework.

Challenges in Masked Video Modeling

Recently proposed masked video modeling methods capture meaningful representations from pre-training videos via randomly masking spatiotemporal tokens from input clips. Random sampling-based

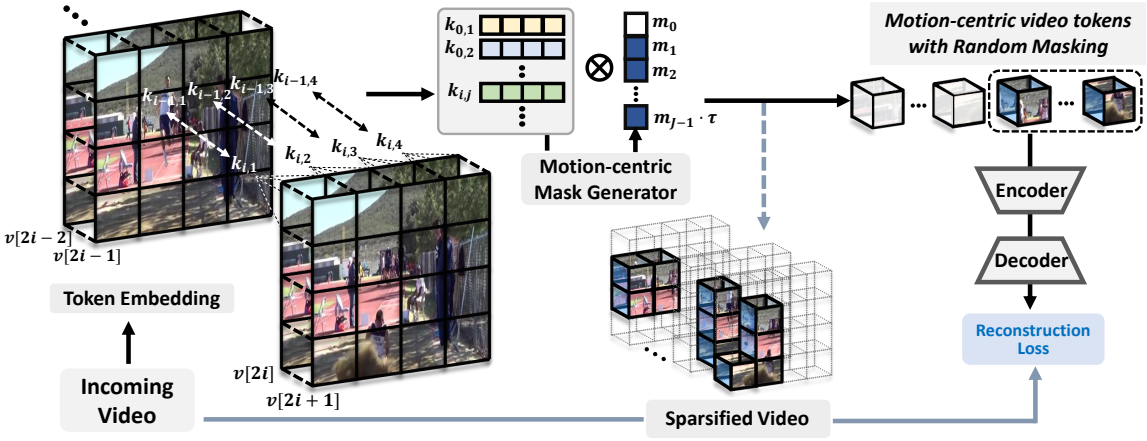


Figure 4.18: **Overview of proposed Motion-centric Token selection (VideoMS)**. We generate the mask by computing the embedded feature distance between adjacent tokens in the time dimension. Our motion-centric masking generator selects tokens with a large disparity with the paired ones in the previous time dimension, which indicates that they include rich motion features. And the model focuses more on learning spatiotemporal representation and drastically saves computational costs by reconstructing only sparsified motion-centric videos.

masking strategies are reasonable for curated and distributionally-stable video datasets. Yet, there is plenty of room for further development to make the model much more robust and computation-efficient.

Here, we summarize two major limitations in the random sampling-based Masked Video Modeling scenario: *(1) Tokenized image patches from a video clip are not equally important.* At each iteration, MVM methods determine which tokens to mask according to specific random selection strategies (e.g., random, space-only, time-only, ...). Yet, the relative amount of information in each token highly depends on the position of the informative objects and the correlation across patches within adjacent frames, which renders most of the tokens highly uninformative or redundant. *(2) MVMs uniformly collect discretized frames from video streams to construct minibatch clips.* Real-world videos may include noisy and highly redundant frame sequences, often uninformative or even detrimental in representing temporal causal relationships and features of moving objects. However, the uniform sampling of video frames results in a dissipation of computational and memory resources in video representation learning. They consume massive training budgets in memory occupancy and suffer from slow convergence speed (e.g., training $3,200$ and $4,800$ epochs on UCF101 [149] and HMDB51 [79], respectively, in VideoMAE [154]). These challenges further exacerbate the problem when applying masked video modeling on uncurated real-world videos such as first-person-view videos[46]. To this end, determining which frames and spatiotemporal tokens to recover is crucial for practical and efficient video representation learning.

Motion-centric Token Selection for Efficient Masked Video Modeling

Inspired by the challenges in recently proposed masked video modeling methods [154, 39], we propose a novel and efficient video representation learning scheme based on Masked Video Modeling with Motion-centric Token Selection, dubbed *VideoMS*. We first explain our motion-centric token selection strategy for MVM. We assume that informative tokens include significant dynamics in the states or movements of objects over time and exclude the most uninformative tokens, which are hardly related to the user's attention or objectives in videos, from the training procedures. We further tackle the training over uncurated and challenging video streams by adaptively selecting input frames based on the amount of

information on objects' movements.

Mask Generation based on Spatiotemporal Token Embedding Distance

Valuable spatiotemporal information in the video streams mainly comes from actively moving objects rather than from static backgrounds. Thus, we aim to learn self-supervision on unsupervised video datasets from only a few crucial regions based on the objects' motions. Let $\boldsymbol{k}_i$ be a token embedding of a pair of $2i^{th}$ and $(2i+1)^{th}$ frames from an input video clip $\boldsymbol{v} \in \mathbb{R}^{2\tau \times C \times H \times W}$, where an index $0 \leq i < \tau$. When $\boldsymbol{k}_{i+1,j}$ indicates the j^{th} token embedding of $\boldsymbol{k}_{i+1}$, we compute the importance $I_{i+1,j}$ of the token $\boldsymbol{k}_{i+1,j}$ by computing the *distance* from the token embedding of the same spatial position in the previous time step, $\boldsymbol{k}_{i,j}$:

$$\begin{aligned} I_{i+1,j} &= S\left(\boldsymbol{k}_{i+1,j}, \boldsymbol{k}_{i,j}\right), \\ \text{where} \quad \boldsymbol{k}_i &= \texttt{Conv3d}(\boldsymbol{v}[2i:2i+1]; \boldsymbol{w}), \end{aligned} \tag{4.20}$$

$S(\cdot,\cdot)$ indicates a distance function, such as Euclidian distance, negative Cosine Similarity, and negative Centered Kernel Alignment (CKA) [27]. Throughout this paper, we use the ℓ_2 norm for the distance function, which is simple yet empirically performs well. After computing the change in spatial information, which means the informative tokens as the video progresses, we perform a two-step mask sampling. Given $\boldsymbol{v}$, the model determines the token embedding vectors with the highest importance ratio of ρ_{pre} based on Equation 4.20. This process is called *motion-centric masking generation* (Depicted in Figure 4.18) and we only use 30% ($\rho_{pre} = 0.3$) of the total of $\tau \times J$ token embeddings for pre-training on UCF101 [149] and HMDB51 [79]. Since only the selected embedding vectors $\widetilde{\boldsymbol{k}}$ go through the encoder-decoder frameworks, this will drastically reduce the computational cost during training (Please see Figure 4.17 (c) and 'Sparsified Video' in Figure 4.18). Second, we generate binary masks via random sampling for $\widetilde{\boldsymbol{k}}$ and the model forwards drawn embedding vectors with the ratio of ρ_{post} into the encoder. A corresponding decoder predicts RGB pixels on all embedding vectors $\widetilde{\boldsymbol{k}}_i \in \widetilde{\boldsymbol{k}}$ from the encoder output features. We set a high masking ratio in general, $\rho_{pre} \cdot \rho_{post} = 0.1$, to follow the settings of our MVM baselines. To this end, the objective function of our VideoMS is formulated as follows:

$$\begin{aligned} \arg\min_{\mathcal{W},\boldsymbol{w}} \sum_{n=1}^{N} \sum_{i=0}^{\tau-1} \left\| f_{\mathcal{W}}(\boldsymbol{m}_i^n \otimes \widetilde{\boldsymbol{k}}_i^n) - \boldsymbol{m}_i^n \otimes \widetilde{\boldsymbol{v}}_n[2i:2i+1] \right\|_F \\ \text{where} \quad \boldsymbol{m}^n = G\left([J \cdot \rho_{pre}], \rho_{post}, \tau\right) \in \{0,1\}^{\tau \times [J \cdot \rho_{pre}]}. \end{aligned} \tag{4.21}$$

We omit the positional embedding term for notational simplicity. Whereas earlier MVM works used a fixed masking ratio per video cues for self-supervised training, our approach allows a dynamic masking rate for each frame by design based on the occupancy of essential tokens. As shown in Figure 4.20, each video frame contains a different amount of valuable spatiotemporal information, and our dynamic masking strategy enables the model to focus on learning the essential representation in a holistic view from arriving video clips. We remark that our proposed masking strategy can be applied to video representation learning and the pre-trained model can be utilized to solve diverse downstream video tasks with a negligible additional computational cost. Furthermore, we can adapt our masking strategy during fine-tuning that selects motion-centric tokens from incoming videos and only processes them for solving downstream task problems. We find that removing unimportant tokens is crucial for both phases, obtaining more meaningful representations and thus achieving superior video downstream task performances. (Please see Table 4.14).

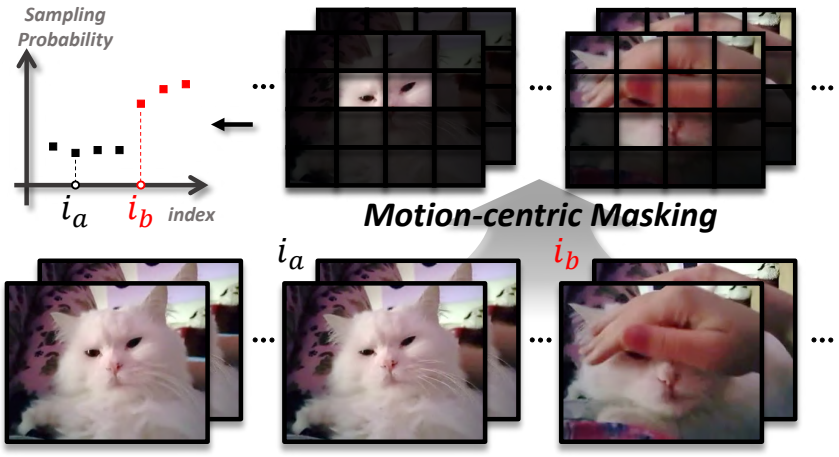


Figure 4.19: Illustration of our proposed **adaptive frame selection.** We dynamically sample the given video by utilizing our **Motion-centric Masking** and learn fluent spatiotemporal representation in non-static sampled frames.

Adaptive Frame Selection Encouraging Model to Temporal Clip Causality

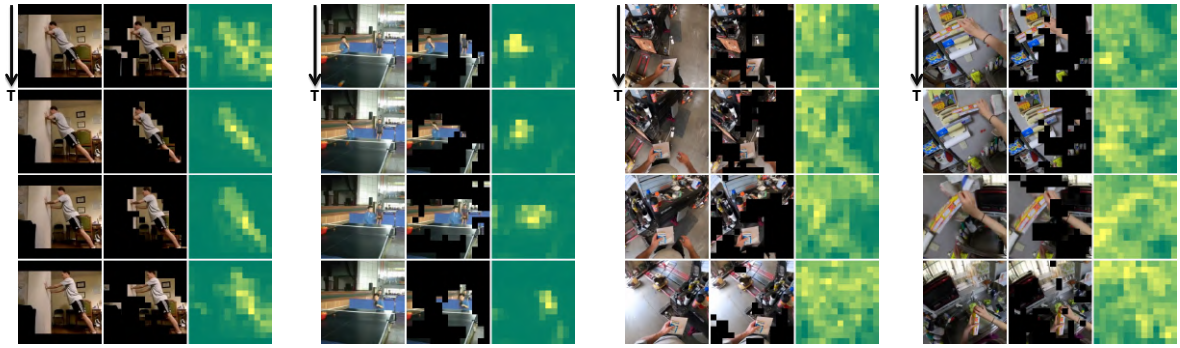
Real-world video streams, especially egocentric first-person videos, may contain many redundant frames and also often include non-useful intermediate clips, such as temporally glancing at uninspiring walls, grounds, or skies, that interfere with estimating a causality of the user's or cameraman's attention. However, most video-based learning methods construct minibatch clips by sampling evenly spaced frames from arriving videos. At each training iteration, a model selects a random frame in each video and collects τ successive frames in total with the same stride. This approach is reasonable for well-curated video benchmark datasets [71, 149] containing only information-dense frames with fixed viewpoints; however, it is unsuitable for uncurated real-world videos, which are more likely to have redundant and overlapped frames that the masked-video model can easily reconstruct.

To overcome the limitation, we propose to adaptively discard uninformative frames in the arrival video and build causal clips that represent the most crucial behaviors of the objects. We illustrate a simple overview of our adaptive frame selection in Figure 4.19 and visualize the sampled results in Figure 4.21. We first select evenly spaced $[\alpha \cdot \tau]$ frames as candidates, α times larger than the input clip size τ, where $\alpha > 1$. We count the number of chosen tokens c_i for the $2i^{th}$ and $(2i+1)^{th}$ frames based on our motion-centric selection. Then, the model iteratively trains on the input clip by drawing τ frames from $[\alpha \cdot \tau]$ candidates without replacement, with a probability of $\frac{c_i}{\sum_i c_i}$ that the paired $2i^{th}$ and $(2i+1)^{th}$ frames are drawn. The model trains video clips with a limited length at each iteration since longer clips require massive computations and memory. Therefore, we remark that adaptive frame selection is crucial to better capture causality in the arrival video, as the model can observe longer video fragments while avoiding redundant frames.

4.3.4 Experimental Results

Experimental Settings

We validate our VideoMS method on multiple video datasets: *UCF101* [149], *HMDB51* [79], and *Ego4D* [46]. UCF101 and HMDB51 are mainly used datasets for video action recognition, which consist of 9.5k/3.5k and 3.5k/1.5k train/val videos, respectively. We evaluate our adaptive frame selection strategy



(a) fixed and static **(b) fixed and dynamic** **(c) changing and messy**

Figure 4.20: **Examples of Motion-centric Token selection.** We show the original video frames (left), Motion-centric masking results (middle), and obtained importance heatmaps (right) on UCF101 [149] (**(a)** and **(b)**, $\rho_{pre} = 0.3$) and Ego4D [46] (**(c)**, $\rho_{pre} = 0.5$) datasets. Curated fixed-view (third-person view) videos, **(a)** and **(b)**, show narrow and concentrated motion information as shown in their heatmaps, whereas motion cues in egocentric videos **(c)** are distributed and contain multi objects.

on the Ego4D dataset, which consists of egocentric videos containing raw and uncurated people's daily lives, by applying it during pre-training. We validate our methods through the *Object State Change Classification (OSCC)*[1] task from Ego4D. Given 8-second videos, OSCC classifies whether the object's state has changed due to interaction with a camera wearer.

Implementation details

We follow the implementation details and experimental settings from prior works [39, 154] for fair comparisons. We set the size of an input clip $\tau = 16$, tokenization size $2 \times 16 \times 16$, and the fixed sampling stride of 4 for UCF101, and 2 for HMDB51, respectively. For our proposed method, *VideoMS*, ρ_{pre} is set to 0.3 and 0.6 for pre-training and fine-tuning, respectively, and we simply adopt Joint Space-Time Attention [8].

Quantitative Evaluation

We first extensively compare our proposed method against strong VRL baselines. We pre-train the VRL methods over benchmark datasets without labels and report the fine-tuning performance on the same datasets in Table 4.13. Our masked video modeling with motion-centric token selection achieves superior fine-tuning performance against baselines across all benchmark tasks. Specifically, it outperforms the best baseline by 2.1%$p \uparrow$ on UCF101 and 3.2%$p \uparrow$ on HMDB51, which demonstrates the effectiveness of focusing on the objects' motions for video representation learning. For the UCF101 dataset, we visualize the convergence plot of VideoMS in Figure 4.16 Left, which shows that it reaches a higher fine-tuning performance than VideoMAE [154] by pre-training at only 800 epochs. We provide masked input examples with their importance heatmaps through our proposed motion-centric masking in Figure 4.20. Our masking strategy enables the model to capture the most critical tokens containing static and dynamic moving of objects (e.g., *doing wall push up* (Figure 4.20 (a)) and *playing ping pong* (Figure 4.20 (b)). Furthermore, our masking strategy captures informative objects even with the rapid change of the view in the first-person videos, as it masks out objects which are crucial for understanding

[1] https://github.com/EGO4D/hands-and-objects/tree/main/state-change-localization-classification

Method	Backbone	Extra data	Frames	Top-1 Acc.(%)	
				UCF101	HMDB51
OPN [88]	VGG	UCF101	N/A	59.6	23.8
VCOP [169]	R(2+1)D	UCF101	N/A	72.4	30.9
CoCLR[51]	S3D-G	UCF101	32	81.4	52.1
Vi^{2}CLR[32]	S3D	UCF101	32	82.8	52.9
CoCLR[51]	S3D-G	K400	32	87.9	54.6
Vi^{2}CLR[32]	S3D	K400	32	89.1	55.7
VideoMAE [154]	ViT-B	✗	16	91.3	62.6
VideoMS(Ours)	ViT-B	✗	16	**93.4**	**65.8**

Table 4.13: **Comparison to state-of-the-art methods on UCF101 and HMDB51.** We outperform the previous method, including the masking-based video model, without using the pre-training step on a large-scale dataset. These results are drawn from [32] and [154].

Method	ρ_{pre}	top-1	GFLOPs
MAE [39]	-	-	46.10
VideoMAE† [154]	-	91.25*	57.50
	-	90.80**	35.48
Ours	0.5	90.91	23.43
	0.4	91.51	21.54
	0.3	**91.56**	19.81
	0.2	90.80	**18.22**
Ours†	0.3	**93.39**	19.81

Method	ρ_{pre}	top-1	GFLOPs
MAE [39]	-	-	180.6
VideoMAE†[154]	-	91.25	180.5
Ours	1.0	89.71	180.5
	0.8	90.17	137.5
	0.7	91.25	117.3
	0.6	**91.56**	98.1
	0.5	91.48	**79.8**
Ours†	0.6	**93.39**	98.1

Table 4.14: **The rate of Motion-centric Masking at pre-training (Left) and fine-tuning (Right) on UCF101.** We highlight the default motion-centric masking rate (ρ_{pre}) as red texts. We basically pre-train our model 800 epochs, but also report the results with 3,200 pre-training epochs following [154], denoted by †. * and ** denotes the results with 75% and 90% masking, respectively.

the camera wearers' attention (Figure 4.20 (c)), while not attending to backgrounds such as walls and floors.

The Impact of Motion-centric Token Selection One of our major contributions is the significantly enhanced computational efficiency during VRL, as we process a few latent vectors in the decoder to reconstruct only the motion-activated tokens in the given videos according to the motion-centric masking ratio ρ_{pre}. We set ρ_{pre} to 0.3 so that our VideoMS reconstructs the 30% of the essential spatiotemporal regions focusing on objects' movements and behaviors, from a sparsified video clip, which reduces 65.5% of GFLOPs at the pre-training phase (Please see Figure 4.16 Right and Table 4.14 Left). Additionally, as reported in Table 4.14 Right, our approach can reduce the computational overhead at the fine-tuning phase, which is practically useful when transferring the learned representation learning model to downstream video tasks.

Next, we emphasize that our motion-centric masking is also applicable when solving video-based downstream tasks. We process only 60% of tokens ($\rho_{pre} = 0.6$) of the given video during fine-tuning. Surprisingly, our motion-centric masking for supervised downstream tasks gains increased performance than our variant without masking on the fine-tuning tasks ($\rho_{pre} = 1.0$) using only about 55% of GFLOPs, as shown in Table 4.14 Right. The results support our hypothesis that video data often contain redundant information and our selective video learning using the proposed masking strategy successfully captures essential space-time regions in video inputs, allowing us to focus more on learning spatiotemporally meaningful features. The fine-tuning masking ratio is higher than the ratio at the pre-training stage, showing that the model uses more information to fully exploit the task-relevant cues from the given videos compared to pre-training, which aims to obtain general information.

Also, we examine the impact of the informativeness of the tokens in Table 4.15. We selectively learn informative tokens by calculating the distance between tokens in adjacent frames in the embedding space, as the tokens farther away from adjacent frames contain less redundant information. When we reversely select near-distance tokens, it drastically decreases the accuracy in pre-training and fine-tuning, obtaining the worst performance when reversely selecting informative tokens in both phases.

Figure 4.21: **Qualitative Motion-centric Sampling results.** We dynamically sample the given video (24 frames of the first row) by probability sampling the frames that have fluent spatiotemporal features (light blue-boxed frames). Without adaptive sampling, the sampled frames (gray-boxed frames) would be highly similar to each other.

Adaptive Frame Selection for Uncurated Video Streams VRL aims to capture spatiotemporal features in incoming videos. But when real-world videos include many redundant scenes over a long time, the model may waste time and computations by learning on meaningless frames, which also can lead to poor local optima due to bias and catastrophic forgetting. To overcome the limitation, we adopt the ratio of motion-centric sampling α to focus on the frames with larger motions across frames. Figure 4.21 provides examples that our sampling method takes various frames and ignores redundant frames in the given video during pre-training. As reported in Table 4.16 Left, downstream task performance with adaptive frame selection results in superior performance to baselines. By constructing the given videos to have fluent motion information, our model focuses more on learning the core part of the video selected by our motion-centric masking ratio ρ_{pre}. Furthermore, we compare our method to recent SoTA methods over OSCC dataset in Table 4.16 Right. Our model pre-trains OSCC without labels and fine-tunes it to classify whether the object's state has changed. With only visual information, we outperform the previous SoTA method Egocentric VLP [98], which uses visual and text information, by 2.3%$p\uparrow$.

Method	Motion-centric Masking		FT Acc.
	Pre-train	Fine-tune	top-1
VideoMS (Ours)	ascending	ascending	60.93
	ascending	descending	73.49
	descending	ascending	75.60
	descending	descending	**91.56**

Table 4.15: **The impact of informative-token selection.** Fine-tuning (FT) performance when the model prioritizes selection with far-distance (descending) or near-distance (ascending) embedded tokens during pre-training and fine-tuning.

α	Acc. (%)
-	73.17
1.3	73.54
1.5	**73.85**
1.8	73.79
2.0	73.80

Method	Modality	Acc. (%)
Egocentric VLP [98]	V+T	73.9
SViT [35]	V	69.8
TarHeels [60]	V	70.8
VideoMS (Ours)	V	**76.2** (+5.4%p)

Table 4.16: **Effect of Motion-centric Sampling ratios (Left) and Comparison to public SoTA methods for OSCC on val set (Right).** We set the default motion-centric sampling rate ($\alpha = 1.5$) and report fine-tuning accuracy. Pre-trained data modalities from the OSCC dataset 'V' and 'T' refers to visual and text.

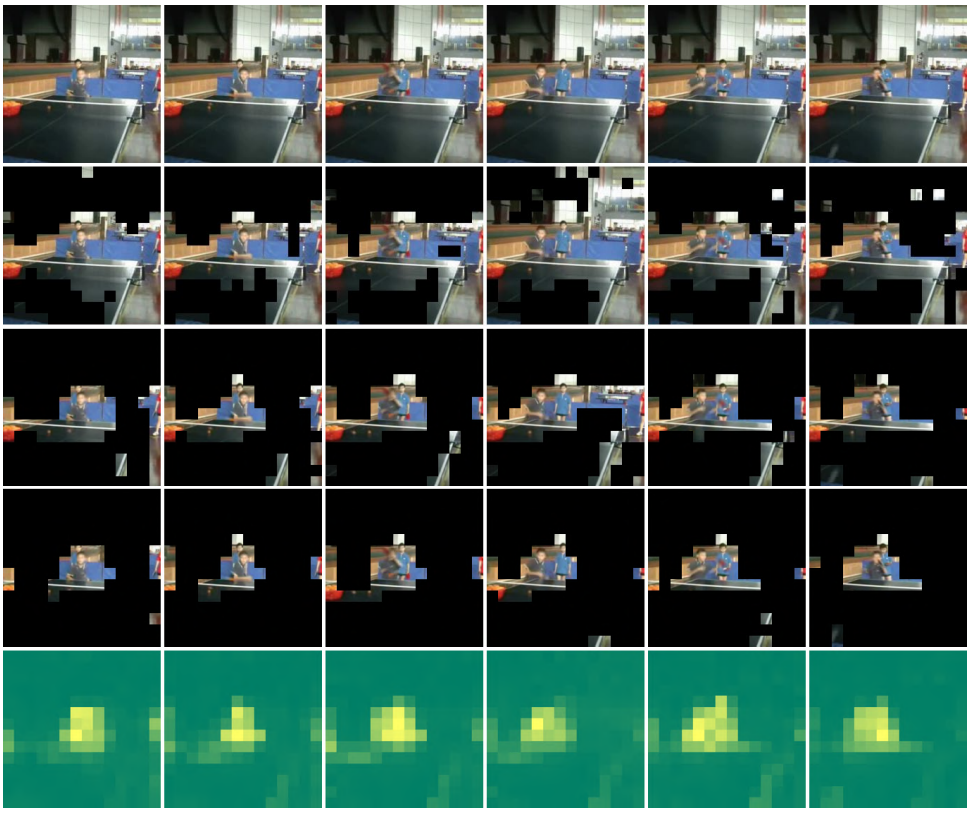


Figure 4.22: **More Examples of Motion-centric Token selection.** We show the original video frames in the first row, Motion-centric masking results in the second, third, and fourth row by varying the ρ_{pre} to 0.5, 0.25, and 0.15, respectively, and obtained importance heatmaps in the last row on UCF101 [149] dataset.

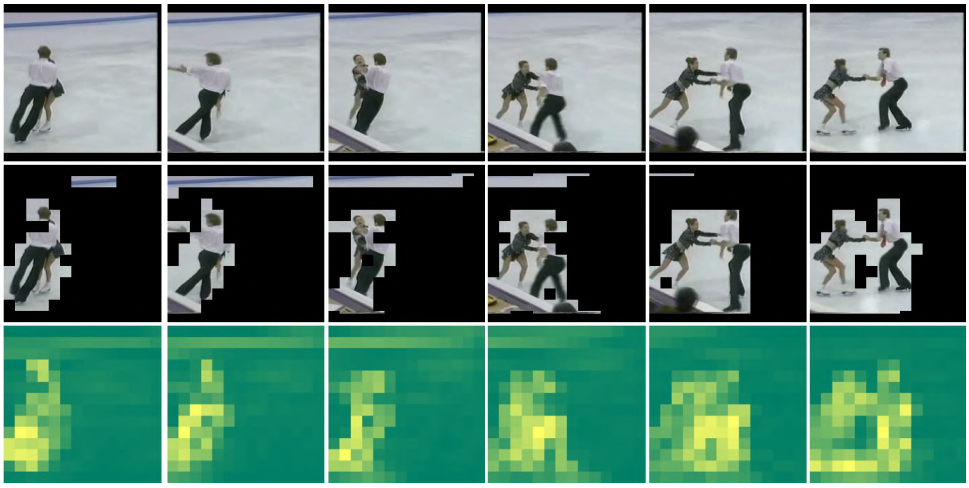


Figure 4.23: **More Examples of Motion-centric Token selection.** We show the original video frames in the first row, Motion-centric masking results in the second row, and obtained importance heatmaps in the last row on UCF101 [149] dataset. ρ_{pre} is set to 0.3 in this example.

Further Visualizations of motion-centric masking We further visualize our masking results in Figure 4.22 for deeper understanding. The first sample is similar in Figure 4.20 (b) but changes the ρ_{pre} from 0.5 to 0.15. The masked result shows that our masking strategy could be better and save more computational costs if the view is fixed and the moving object is few. In Figure 4.23, the masked results in frames of the first to fourth show that our masking strategy wrongly captured the blue line of each frame's upper regions as informative tokens in the given video. As the heatmaps show that those tokens are relatively low informative than the two skaters in the video, it still makes sense that our masking strategy can successfully capture the informative tokens in the given video.

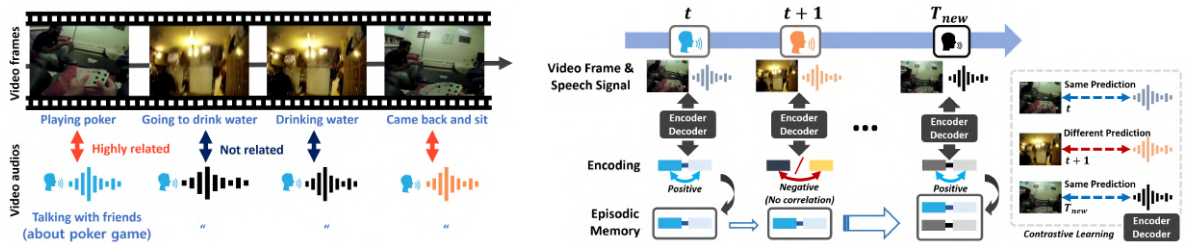
4.3.5 Summary

In this paper, we propose a simple yet efficient parameter-free token/frame selection method for masked video representation learning. Our method is based on the intuition that not all tokens in the given video are equally informative. We adaptively select the most useful spatiotemporal informative tokens based on objects' moving and train only crucial motion-centric tokens with the sparsified video clips, drastically reducing memory allocation and computational cost. In addition, we propose a frame selection technique to construct input video data by sampling incoming frames with a probability proportional to the degree of the occupancy of motion-centric tokens. This strategy can be useful for video representation learning with uncurated videos such as egocentric videos from first-person cameras. The experimental results show that our method is significantly more efficient in computations and memory, reaching the target performance with a much smaller number of training epochs than the baselines and achieving substantially higher performance when trained for the same number of epochs.

Chapter 5. Conclusion and Future Work

So far, I have addressed critical challenges of deploying the On-device, Online Continual Learning framework for the real world, particularly focused on sequential training of a course of discrete image-based tasks. However, the world is continuous, and multiple sensory information persistently arrives over the lifetime. Similarly, the incoming data stream in portable devices is a form of egocentric videos containing dynamically-changing perspectives and domains depending on user-centered attention and causality. Beyond deploying incremental models for image classification tasks, I aim to understand the present circumstances from incoming input streams and anticipate future user actions or environmental changes by actively utilizing the episodic memory learned from the past. Ultimately, machines can evolve to understand semantics and estimate the physical world/wearer's intentions. I briefly discuss my future research directions in the following sections.

5.1 Audio-video-text Multimodal Video Understanding



(a) Sparse Multimodal Correlation between Audio and Video Frames

(b) Causality-based Mixed-modal Buffer Selection

Figure 5.1: **(a) Multimodal Video Understanding is challenging** since the data includes sparse multimodal connectivity, and the model should store only related multimodal pairs in the replay buffer for future training. **(b) the model captures the entanglements of multimodal representations** using contrastive learning, only memorizing highly correlated pairs guided by speaker predictions from different domains.

Though most continual learning literature is restricted to solving single-modal problems in a vision or language domain, real-world egocentric videos obtained by on-device AIs often contain audio-video multimodality and occasional text information (i.e., captions). However, this challenging scenario raises several crucial problems. Let's take a simple example. A man calls a friend while cleaning his room. During the call, **1) The view of a man and calling sounds are only sometimes meaningful.** Many intermediate video/audio clips are composed of meaningless noise. Training all of the arriving data streams is therefore not helpful. **2) Incoming visual and acoustic information are not always correlated.** Temporal video clips or frames with corresponding audio can be highly correlated, and in some others, they are still important in some sense yet irrelevant to each other (Please see Figure 5.1 (a). **3) The on-device agent should preserve and improve the prior knowledge** while training new input streams every second.

I aim to tackle the problems of multimodal video understanding for semantic/episodic memory and future forecasting [46], to understand challenging multimodal environments around the embedded

machines, and construct incrementally improving perception from multiple-modal senses like humans. I suggest a new direction with causality-based input selective training and mixed-modal experience replay, described in Figure 5.1 (b). Real-world videos include highly sparse correlations among multiple modes, and simple training of all current incoming modalities is unsuitable for the objective. Transferring only correlated multimodal features across a long period is crucial to prevent the model from training unfactual multimodal relationships and adapting long-term memory for sustainable AGI. Motivated by my prior works, Online Coreset Selection [176] and Motion-centric Token Selection [58], the model learns to project the audio/video representations in the feature embedding space and captures accurate relationships between features across different mode inputs. It performs multimodal self-supervised contrastive learning, allowing selective multimodal knowledge transfer and weight sharing if correlated. Also, the model finds strongly correlated input pairs from different modes to store them in a replay buffer for long-term memory. Thus, it can prevent learning unintended and unfactual multimodal causality and encourage positive transfer across different data sources.

5.2 Self-improving Single-life/Autonomous Reinforcement Learning

Beyond solving static tasks like classification, for deploying sustainable AI, I aim to build continuously improving autonomous Reinforcement Learning (RL) agents on a real-world platform throughout a single life. Given environments, multiple irreversible states can occur (e.g., objects falling under the table, weather changes, robot arms breaking, unexpected obstacles like birds or bugs entering the state, etc.), requesting different proactive interventions from humans, or other external supports.

While earlier work [138, 139, 166] focuses on avoiding an irreversible state during autonomous RL, I believe the agent needs to estimate multiple irreversible states, where they occasionally occur during training, and occurrence probability can vary over single-life RL [21] (i.e., continuous distributional shift), similar to the temporally changing input distribution in continual learning. Further, some irreversible states are much more frequently occurring than others (imbalanced occurrence of irreversible states).

I aim to build an RL agent that incrementally learns to avoid multiple irreversible states sequentially as the environment changes (ideally throughout a single life) without forgetting previously learned skills. Thus, the agent can request proactive human intervention appropriately, or resolves the problem with simple remedies w/o resetting the environment through human intervention (if a bug/bird comes and hinders the operation of the reinforcement agent, the machine can scare them away with sounds or winds).

To prevent skill forgetting and deploy a continuously self-improving agent for autonomous/single-life RL, I aim to explore adopting long-term episodic memory and conditional generative replay/imitation learning. The agent stores latent states/representations of irreversible events in the long-term episodic memory and repeats learned skills (and types of irreversible states) via generative replay conditioned to the current environment (since the environment might be changed during single-life RL without resetting). To this end, the RL agent incrementally acquires/adapts new skills throughout their single-lifespan while minimizing human intervention.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467*, 2016.

[2] Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[3] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[4] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[5] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[6] Jinheon Baek, Wonyong Jeong, Jiongdao Jin, Jaehong Yoon, and Sung Ju Hwang. Personalized subgraph federated learning. *arXiv preprint arXiv:2209.07529*, 2022.

[7] Horace Barlow. Possible principles underlying the transformations of sensory messages. 1961.

[8] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[9] Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-jussà. Continual lifelong learning in natural language processing: A survey. *arXiv preprint arXiv:2012.09823*, 2020.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[11] Zalán Borsos, Mojmír Mutnỳ, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[12] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. Streaming variational bayes. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[13] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. 1994.

[14] Yaroslav Bulatov. Not-mnist dataset. 2011.

[15] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[16] Trevor Campbell and Tamara Broderick. Automated scalable bayesian inference via hilbert coresets. *Journal of Machine Learning Research*, 2019.

[17] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[18] Arslan Chaudhry, Naeemullah Khan, Puneet K Dokania, and Philip HS Torr. Continual learning in low-rank orthogonal subspaces. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[19] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[20] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019.

[21] Annie S Chen, Archit Sharma, Sergey Levine, and Chelsea Finn. You only live once: Single-life reinforcement learning. *arXiv preprint arXiv:2210.08863*, 2022.

[22] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[23] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[24] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[25] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[26] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 2019.

[27] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 2012.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[29] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*, 2020.

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

[31] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[32] Ali Diba, Vivek Sharma, Reza Safdari, Dariush Lotfi, Saquib Sarfraz, Rainer Stiefelhagen, and Luc Van Gool. Vi2clr: Video and image for visual contrastive learning of representation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[33] Carl Doersch, Ankush Gupta, and Andrew Zisserman. Crosstransformers: spatially-aware few-shot transfer. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[34] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[35] Maria Escobar, Laura Daza, Cristina González, Jordi Pont-Tuset, and Pablo Arbeláez. Video swin transformers for egocentric video understanding@ ego4d challenges 2022. *arXiv preprint arXiv:2207.11329*, 2022.

[36] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. 2020.

[37] Yuchun Fang, Zhengyan Ma, Zhaoxiang Zhang, Xu-Yao Zhang, and Xiang Bai. Dynamic multi-task learning with convolutional neural network. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[38] Mohsen Fayyaz, Soroush Abbasi Kouhpayegani, Farnoush Rezaei Jafari, Eric Sommerlade, Hamid Reza Vaezi Joze, Hamed Pirsiavash, and Juergen Gall. Ats: Adaptive token sampling for efficient vision transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[39] Christoph Feichtenhofer, Haoqi Fan, Yanghao Li, and Kaiming He. Masked autoencoders as spatiotemporal learners. 2022.

[40] Basura Fernando, Hakan Bilen, Efstratios Gavves, and Stephen Gould. Self-supervised video representation learning with odd-one-out networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[42] Timo Flesch, Jan Balaguer, Ronald Dekker, Hamed Nili, and Christopher Summerfield. Comparing continual task learning in minds and machines. *Proceedings of the National Academy of Sciences*, 2018.

[43] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[44] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[45] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[46] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[47] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent - a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[48] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[49] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[50] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[51] Tengda Han, Weidi Xie, and Andrew Zisserman. Self-supervised co-training for video representation learning. 2020.

[52] Chaoyang He, Murali Annavaram, and Salman Avestimehr. Group knowledge transfer: Federated learning of large cnns at the edge. 2020.

[53] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[54] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[56] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[57] Ching-Yi 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 *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[58] Sunil Hwang, Jaehong Yoon, Youngwan Lee, and Sung Ju Hwang. Efficient video representation learning via masked video modeling with motion-centric token selection. *arXiv preprint arXiv:2211.10636*, 2022.

[59] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[60] Md Mohaiminul Islam and Gedas Bertasius. Object state change classification in egocentric videos using the divided space-time attention mechanism. *arXiv preprint arXiv:2207.11814*, 2022.

[61] Wonyong Jeong, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. Federated semi-supervised learning with inter-client consistency and disjoint learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[62] Donglin Jiang, Chen Shan, and Zhihui Zhang. Federated learning algorithm based on knowledge distillation. In *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. IEEE, 2020.

[63] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

[64] Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[65] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[66] Ioannis Kakogeorgiou, Spyros Gidaris, Bill Psomas, Yannis Avrithis, Andrei Bursuc, Konstantinos Karantzalos, and Nikos Komodakis. What to hide from your students: Attention-guided masked image modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[67] Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D Yoo. Forget-free continual learning with winning subnetworks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[68] Haeyong Kang, Jaehong Yoon, Sultan Rizky Hikmawan Madjid, Sung Ju Hwang, and Chang D Yoo. On the soft-subnetwork for few-shot class incremental learning. *arXiv preprint arXiv:2209.07529*, 2022.

[69] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[70] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[71] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[72] Christopher J Kelly, Alan Karthikesalingam, Mustafa Suleyman, Greg Corrado, and Dominic King. Key challenges for delivering clinical impact with artificial intelligence. *BMC medicine*, 2019.

[73] Sungnyun Kim, Gihun Lee, Sangmin Bae, and Se-Young Yun. Mixco: Mix-up contrastive learning for visual representation. *arXiv preprint arXiv:2010.06300*, 2020.

[74] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

[75] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[76] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2019.

[77] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[79] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

[80] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

[81] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

[82] Christoph Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to Detect Unseen Object Classes by Between-Class Attribute Transfer. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[83] Matthias De Lange, Xu Jia, Sarah Parisot, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Unsupervised model personalization while preserving privacy and scalability: An open problem. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[84] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[85] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[86] Cecilia S Lee and Aaron Y Lee. Clinical applications of continual learning machine learning. *The Lancet Digital Health*, 2020.

[87] Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, Yongkweon Jeon, Baeseong Park, and Jeongin Yun. FleXOR: Trainable fractional quantization. 2020.

[88] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[89] Kibok Lee, Yian Zhu, Kihyuk Sohn, Chun-Liang Li, Jinwoo Shin, and Honglak Lee. i-mix: A domain-agnostic strategy for contrastive representation learning. In *ICLR*, 2021.

[90] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[91] Matthias Lenga, Heinrich Schulz, and Axel Saalbach. Continual learning for domain adaptation in chest x-ray classification. In *Medical Imaging with Deep Learning*, 2020.

[92] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[93] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[94] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[95] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[96] Yuanpeng Li, Liang Zhao, Kenneth Church, and Mohamed Elhoseiny. Compositional language continual learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[97] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[98] Kevin Qinghong Lin, Alex Jinpeng Wang, Mattia Soldan, Michael Wray, Rui Yan, Eric Zhongcong Xu, Difei Gao, Rongcheng Tu, Wenzhe Zhao, Weijie Kong, et al. Egocentric video-language pretraining. *arXiv preprint arXiv:2206.01670*, 2022.

[99] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. 2020.

[100] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[101] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[102] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[103] Divyam Madaan, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang. Rethinking the representational continuity: Towards unsupervised continual learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[104] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. 1989.

[105] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[106] Guangxu Mei, Ziyu Guo, Shijun Liu, and Li Pan. Sgnn: A graph neural network based federated learning approach by hiding structure. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

[107] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[108] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[109] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[110] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[111] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems (NIPS)*, 2020.

[112] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE international conference on image processing (ICIP)*, pages 343–347. IEEE, 2014.

[113] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[114] Didrik Nielsen and Ole Winther. Closing the dequantization gap: PixelCNN as a single-layer flow. 2020.

[115] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[116] Tian Pan, Yibing Song, Tianyu Yang, Wenhao Jiang, and Wei Liu. Videomoco: Contrastive video representation learning with temporally adversarial examples. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[117] Seong Hyeon Park, Jihoon Tack, Byeongho Heo, Jung-Woo Ha, and Jinwoo Shin. K-centered patch sampling for efficient video recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[118] Anastasia Pentina and Christoph H. Lampert. A pac-bayesian bound for lifelong learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

[119] Anastasia Pentina and Christoph H. Lampert. Lifelong learning with non-i.i.d. tasks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[120] Anastasia Pentina and Ruth Urner. Lifelong learning with weighted majority votes. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[121] AJ Piergiovanni, Anelia Angelova, and Michael S Ryoo. Evolving losses for unsupervised video representation learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[122] Rui Qian, Tianjian Meng, Boqing Gong, Ming-Hsuan Yang, Huisheng Wang, Serge Belongie, and Yin Cui. Spatiotemporal contrastive video representation learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[123] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, 2019.

[124] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[125] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[126] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[127] 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 *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[128] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[129] Mohammad Rostami, Soheil Kolouri, Kyungnam Kim, and Eric Eaton. Multi-agent distributed lifelong learning for collective knowledge acquisition. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[130] Andrei Rusu, Neil Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[131] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

[132] Samik Sadhu and Hynek Hermansky. Continual learning in automatic speech recognition. In *Interspeech*, 2020.

[133] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. 2016.

[134] Masa-Aki Sato. Online model selection based on the variational bayes. *Neural computation*, 2001.

[135] Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[136] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[137] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[138] Archit Sharma, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Autonomous reinforcement learning via subgoal curricula. 2021.

[139] Archit Sharma, Kelvin Xu, Nikhil Sardana, Abhishek Gupta, Karol Hausman, Sergey Levine, and Chelsea Finn. Autonomous reinforcement learning: Formalism and benchmarking. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[140] Zhiqiang Shen, Zechun Liu, Zhuang Liu, Marios Savvides, Trevor Darrell, and Eric Xing. Un-mix: Rethinking image mixtures for unsupervised visual representation learning. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI)*, 2022.

[141] Hanul Shin, Jung Kwon Lee, Jaehon Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[142] Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef, and Itai Zeitak. Overcoming forgetting in federated learning on non-iid data. *arXiv preprint arXiv:1910.07796*, 2019.

[143] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[144] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[145] Samarth Sinha, Jiaming Song, Animesh Garg, and Stefano Ermon. Experience replay with likelihood-free importance weights. *arXiv preprint arXiv:2006.13169*, 2020.

[146] James Smith, Cameron Taylor, Seth Baer, and Constantine Dovrolis. Unsupervised progressive learning and the stam architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[147] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019.

[148] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. 2020.

[149] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[150] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, 2011.

[151] Xiao Sun, Naigang Wang, Chia-yu Chen, and Jia-min Ni. Ultra-low precision 4-bit training of deep neural networks. 2020.

[152] Sebastian Thrun. *A Lifelong Learning Perspective for Mobile Robot Control*. Elsevier, 1995.

[153] Michalis K Titsias, Jonathan Schwarz, Alexander G de G Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with gaussian processes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[154] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. 2022.

[155] Vikas Verma, Thang Luong, Kenji Kawaguchi, Hieu Pham, and Quoc Le. Towards domain-agnostic contrastive learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[156] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[157] Jiangliu Wang, Jianbo Jiao, and Yun-Hui Liu. Self-supervised video representation learning by pace prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[158] Junke Wang, Xitong Yang, Hengduo Li, Liu Li, Zuxuan Wu, and Yu-Gang Jiang. Efficient video transformers with spatial-temporal token selection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[159] Rui Wang, Dongdong Chen, Zuxuan Wu, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Yu-Gang Jiang, Luowei Zhou, and Lu Yuan. Bevt: Bert pretraining of video transformers. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[160] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[161] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[162] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. Fedgnn: Federated graph neural network for privacy-preserving recommendation. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2021.

[163] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and Inference with Integers in Deep Neural Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[164] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[165] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[166] Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. When to ask for help: Proactive interventions in autonomous reinforcement learning. 2022.

[167] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. *arXiv preprint arXiv:2111.09886*, 2021.

[168] Yazhou Xing, Zian Qian, and Qifeng Chen. Invertible image signal processing. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[169] Dejing Xu, Jun Xiao, Zhou Zhao, Jian Shao, Di Xie, and Yueting Zhuang. Self-supervised spatiotemporal learning via video clip order prediction. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[170] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[171] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[172] Jaehong Yoon and Sung Ju Hwang. Combined group and exclusive sparsity for deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[173] Jaehong Yoon, Sung Ju Hwang, and Yue Cao. Continual learners are incremental model generalizers. *preprint*, 2022.

[174] Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[175] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[176] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[177] Jaehong Yoon, Geon Park, Wonyong Jeong, and Sung Ju Hwang. Bitwidth heterogeneous federated learning with progressive weight dequantization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[178] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[179] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[180] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[181] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[182] Mengyao Zhai, Lei Chen, Frederick Tung, Jiawei He, Megha Nawhal, and Greg Mori. Lifelong gan: Continual learning for conditional image generation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019.

[183] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[184] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[185] Xishan Zhang, Shaoli Liu, Rui Zhang, Chang Liu, Di Huang, Shiyi Zhou, Jiaming Guo, Qi Guo, Zidong Du, Tian Zhi, and Yunji Chen. Fixed-Point Back-Propagation Training. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[186] Kang Zhao, Sida Huang, Pan Pan, Yinghan Li, Yingya Zhang, Zhenyu Gu, and Yinghui Xu. Distribution Adaptive INT8 Quantization for Training CNNs. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[187] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[188] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoders. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[189] Luowei Zhou, Yingbo Zhou, Jason J Corso, Richard Socher, and Caiming Xiong. End-to-end dense video captioning with masked transformer. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[190] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv:1606.06160 [cs]*, 2018.

[191] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. Towards Unified INT8 Training for Convolutional Neural Network. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.