Enhancing Online Continual Learning with Plug-and-Play State Space Model and Class-Conditional Mixture of Discretization

Sihao Liu[1] Yibo Yang[2] Xiaojie Li[3] David A. Clifton[4] Bernard Ghanem[2]
[1]Harbin Institute Of Technology [2]King Abdullah University of Science and Technology
[3]Harbin Institute Of Technology (Shenzhen) [4]University of Oxford
liusihaowo@gmail.com ibo@pku.edu.cn xiaojieli0903@gmail.com
david.clifton@eng.ox.ac.uk bernard.ghanem@kaust.edu.sa

Abstract

Online continual learning (OCL) seeks to learn new tasks from data streams that appear only once, while retaining knowledge of previously learned tasks. Most existing methods rely on replay, focusing on enhancing memory retention through regularization or distillation. However, they often overlook the adaptability of the model, limiting the ability to learn generalizable and discriminative features incrementally from online training data. To address this, we introduce a plug-and-play module, S6MOD, which can be integrated into most existing methods and directly improve adaptability. Specifically, S6MOD introduces an extra branch after the backbone, where a mixture of discretization selectively adjusts parameters in a selective state space model, enriching selective scan patterns such that the model can adaptively select the most sensitive discretization method for current dynamics. We further design a class-conditional routing algorithm for dynamic, uncertainty-based adjustment and implement a contrastive discretization loss to optimize it. Extensive experiments combining our module with various models demonstrate that S6MOD significantly enhances model adaptability, leading to substantial performance gains and achieving the state-of-the-art results.

1. Introduction

Continual learning (CL) [9, 28, 30, 42] is a task that requires models to continuously and efficiently learn new ability when receiving new data. It tests the model's ability to adapt to dynamically changing environments while retaining existing knowledge. For example, autonomous vehicles are expected to continuously learn new driving environments and traffic regulations, thereby improving detection accuracy under complex driving scenes. From the perspective of task format, CL can be divided into offline and online [9, 42]. Unlike offline CL [6, 32], online CL (OCL) [9, 16] aligns with real-world implementations as it requires data to arrive sequentially in mini-batches and allows only one epoch of training, posing greater challenges for efficient adaptation with data accessible for learning only once.

To tackle the challenging OCL, existing studies widely rely on the replay technique [17] that selectively stores a subset of old-class data to strengthen the memory ability of existing knowledge and mitigate catastrophic forgetting. Building on replay, various methods have been proposed with regularization, buffer allocation, and distillation strategies [17, 18, 33, 43, 45, 49]. For example, MIR [3] aims to mitigate mutual interference among tasks through a retrieval strategy, OCM [17] seeks to reduce forgetting by maximizing mutual information between different tasks, and CCLDC [43] adopts collaborative learning and distillation to improve plasticity. However, learning generalizable and discriminative features incrementally from online training data is still intractable, and imprecise features will impede and even mislead the replay strategies [44]. Therefore, inducing accurate and efficient adaptation with limited data access is critically important for OCL and there still remains significant potential for improvement.

Recently, selective state space models (SSMs), also known as S6 in Mamba [13], have shown promising results in modeling long-range dependencies with computational efficiency. Selective SSMs introduce a selective scan mechanism to make the interactions among sequential states aware of input context, and have been applied to vision tasks by integrating various scan directions [26, 56]. In addition to the success in sequence modeling [8, 25, 34, 48], selective SSMs also exhibit stronger adaptability than static parameters in few-shot class-incremental learning due to the dynamic operation weights, as reported in a recent pioneering work [23]. Despite the increased capacity of adaptation, directly applying selective SSMs to OCL will be infeasible because it is challenging for selective SSMs to capture a precise discretization pattern with limited context from online data that appears only once for training. Moreover, a single discretization mechanism may fail to capture all the essential details in some nonlinear dynamic systems or systems with multi-scale characteristics [1, 15].

To this end, we develop a plug-and-play branch based on selective state space model and class-conditional mixture of discretization. The branch with an equiangular tight frame classifier [28, 36, 47, 50, 51, 55] can be integrated on any existing OCL methods to supervise the original classification head and improve the adaptation ability in OCL. Considering multiple discretization methods can exhibit varying sensitivity to the dynamic characteristics of a system, we introduce a mixture of discretization into SSMs to enrich selective scan patterns such that the model can adaptively select the most sensitive discretization method for current dynamics. This flexibility allows our method to comprehensively capture the system's complex dynamic features, particularly in cases of rapid state changes and evolving systems as exemplified by OCL.

In order to guide the mixture of discretization for OCL, where a delicate trade-off between maintaining the stability of old knowledge and fostering the plasticity for new ability is needed, we further introduce a class-conditional routing to aggregate the discretization patterns. Concretely, we maintain a feature prototype for each class to calculate the class uncertainty based on the margin among different classes. For classes with less uncertainty, fewer discretization patterns will be aggregated to stabilize the abilities already acquired for these classes, while for classes with large uncertainty, more discretization patterns will be included to allocate additional capacity for adapting to these undeveloped abilities. After aggregation of the mixture of discretization, we employ a contrastive discretization loss that enforces within-class consistency and between-class diversity, contributing to the learning of generalizable and discriminative features after the selective scan. In experiments, we integrate our plug-and-play branch on numerous OCL methods and significant improvements can be consistently observed on multiple datasets.

Our contributions can be summarized as follows:

- To enhance the adaptability of existing methods for OCL, we propose S6MOD, a plug-and-play module based on selective state space model and class-conditional mixture of discretization, which can strengthen the base method.
- We further develop a class-conditional gating strategy that dynamically satisfy both objectives of maintaining the stability of knowledge already acquired and fostering the plasticity for undeveloped abilities. A contrastive discretization loss is employed to facilitate the learning of generalizable and discriminative features.
- In experiments, our method can be easily integrated on different OCL methods and is compatible with distillation techniques, to consistently achieve significant improvements on multiple datasets including CIFAR-10, CIFAR-100, and Tiny-ImageNet.

2. Related Work

Continual Learning(CL). CL can be categorized into three types based on the implementation methods [9, 28, 30, 42]: regularization-based, parameter isolation-based, and replay-based. Regularization-based methods [2, 20, 24, 41, 54] introduce regularization terms to restrict changes in model parameters, thereby protecting the already learned knowledge. Parameter isolation-based methods [12, 35, 37, 46, 52] typically assign different model parameters or subnetworks to different tasks to avoid interference between tasks. Replay-based methods [5, 7, 27, 32, 33] perform joint training by either storing a portion of old data or generating pseudo-data. This allows the model to revisit old data while learning new data, thereby reducing forgetting.

Online Continual Learning(OCL). OCL is a specialized form of CL designed to test the ability to continuously learn and update as data arrives in real-time streams [9, 16]. This means that OCL can only train for a single epoch and typically process only individual samples or mini batches at a time. Replay-based methods are widely used in OCL [17]. ER [33] introduces the combination of cross-entropy loss with a random buffer. OCM [17] uses mutual information maximization to reduce feature bias and preserve past knowledge. GSA [18] proposes a gradient-based adaptive optimization method to address dynamic training bias. OnPro [45] uses online prototype equilibrium to address shortcut learning problem. Some methods introduce knowledge distillation based on replay. CCL-DC [43] introduces Collaborative Continual Learning (CCL) and Distillation Chain (DC) to enhance model plasticity. MOSE [49] alleviates forgetting by integrating multi-level supervision and reverse self-distillation. However, these models carry a risk when the features used to distill are not precise. In comparison, our method S6MOD as a plug-and-play module is applicable to most OCL methods to improve the adaptability of these models, enabling them to learn generalizable and discriminative features more efficiently.

Selective State Space Model (S6) Selective State Space Model (S6) [13] has gained increasing interest as an alternative to self-attention [40] with lower computational complexity. S6 enhances the S4 model [14] by introducing the selective scan mechanism, and its effectiveness in vision tasks has been extensively studied [26, 56]. For example, Vmamba [26] introduces SS2D, a cross-scanning mechanism for images, facilitating the extension of Mamba to process vision data. A recent study Mamba-FSCIL [23] leverages the dynamic weights and sequence modeling capability of Mamba to achieve dynamic adaptation in few-shot class-incremental learning. But their method rely on an effective selective scan learned from training data. In OCL, the online data that appears only once for training provides limited context for Mamba models, and thus poses challenges to capture a precise discretization pattern. Different from these studies, our method integrates S6 with class-conditional mixture of discretization and effectively helps to improve the adaptability for OCL. Some existing works have introduced mixture of experts (MoE) into Mamba [4, 25, 31], following the design of switch-transformer [11]. Our method differs from them in that each expert in our mixture of discretization is only a simple linear layer for computation efficiency, and discretization patterns are aggregated with our class-conditional gating to balance between maintaining stability of old knowledge and allocating capacity for learning new abilities.

3. Preliminaries

3.1. Problem Definition of OCL

OCL requires a model to continuously update itself from a data stream, with each mini-batch containing new data sampled from a changing distribution D_t. At each time step t, the model receives a mini-batch of data $\{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{n_t}$, where $x_i^{(t)}$ represents the input data, $y_i^{(t)}$ represents the corresponding labels, and n_t is the number of samples. The goal of OCL is to sequentially update the model to adapt to both gradual and task-specific shifts in the data distribution. Given a pre-trained network f_θ with parameters θ, the model updates its parameters by solving an optimization problem $\arg\min_\theta L(f_\theta(x), y)$, where L is the loss function, thereby enabling it to adapt effectively to the evolving data distribution.

3.2. State Space Models

State Space Models (SSMs) can be viewed as linear time-invariant systems. They map input sequences $x(t) \in \mathbb{R}$ to output sequences $y(t) \in \mathbb{R}$ through hidden states $h(t) \in \mathbb{R}^N$. Mathematically, these models can be represented as linear ordinary differential equations (ODE) :

$$\begin{aligned} h'(t) &= Ah(t) + Bx(t), \\ y(t) &= Ch(t), \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{N\times N}$ represents the state transition matrix, while $B \in \mathbb{R}^{N\times 1}$ and $C \in \mathbb{R}^{1\times N}$ denote the mapping matrices from input to latent state and from latent state to output, respectively, where N indicates the size of the hidden state.

To apply the SSM to real discrete data, zero-order hold (ZOH) [14] is employed. It discretizes the continuous parameters A, B, and C using the time scale parameter $\mathbf{\Delta}$. The discretized SSM equations can be rewritten as follows:

$$\begin{aligned} h_t &= \overline{A}h_{t-1} + \overline{B}x_t, \\ y_t &= Ch_t. \end{aligned} \tag{2}$$

Recently, Gu [13] proposed a new parameterization method for SSM with a selective scan mechanism, which is known as S6 and serves as the core of the Mamba model. To enhance S6's capability in processing visual data, Liu et al. [26] introduced SS2D, which can serialize image data from four different directions. Given the input data x, the output $\overline{x}$ processed by SS2D can be expressed as follows:

$$\overline{x} = \text{SS2D}(x) = \sum_{i=1}^{4} \text{S6}(\text{scan}(x, i)). \tag{3}$$

4. Proposed Method

In this section, we will first introduce the overall structure of our plug-and-play module S6MOD in Sec. 4.1. After that, we will present the detailed design of the state space model with mixture of discretization in Sec. 4.2. Then, we will introduce the class-conditional routing and its corresponding contrastive discretization loss in details in Sec. 4.3. Finally, we will specify the optimization details in Sec. 4.4.

4.1. Overall structure

To better enhance existing methods' ability in learning generalizable and discriminative features incrementally from online training data, we propose a plug-and-play module named S6MOD that can be easily applied to existing OCL methods, as shown in Fig. 1 (a). Overall, S6MOD strengthens the original method by introducing a plug-and-play branch after the backbone. It consists of a block and a fixed equiangular tight frame (ETF) classifier [50, 51]. The intermediate features $\mathbf{F}$ generated by the backbone are duplicated and sent into the classification head of the original base method, and the selective state space model (SSM) branch introduced by our method.

In our SSM branch, $\mathbf{F}$ is projected into two paths through an MLP. The first one $\mathbf{X} = f_x(\mathbf{F})$ is used to perform selective scan with our class-conditional mixture of discretization, as will be detailed in later subsections. The other one $\mathbf{Z} = f_z(\mathbf{F})$ performs a gating mechanism as commonly adopted in Mamba models [23, 26, 29]. The output feature of this branch can be formulated as:

$$\boldsymbol{\mu} = \text{SiLU}(\mathbf{Z}) \otimes \text{S6MOD}(\text{SiLU}(\text{Conv}(\mathbf{X}))). \tag{4}$$

To assist the learning of the base method for generalizable and discriminative features, we adopt a regularization, $\mathcal{L}_{\text{Diff}}$, on the predicted distribution of the base method, which is the KL divergence between P and Q and can be formulated as:

$$\mathcal{L}_{\text{Diff}} = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right), \tag{5}$$

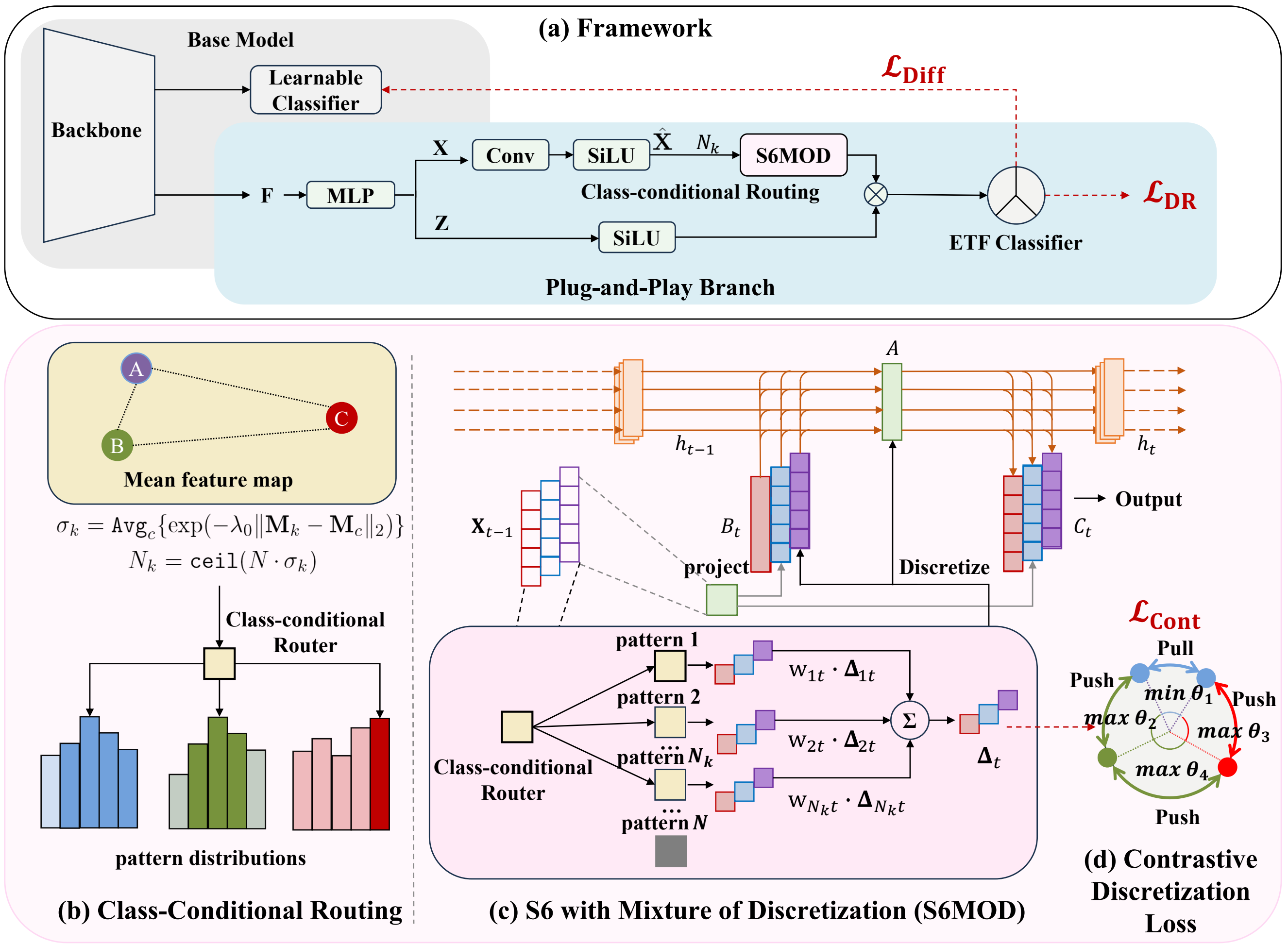

(a) Framework
Base Model
Backbone
Learnable Classifier
$\mathcal{L}_{\text{Diff}}$
X
Conv
SiLU
$\hat{\mathbf{X}}$
N_k
S6MOD
Class-conditional Routing
F
MLP
Z
SiLU
ETF Classifier
$\mathcal{L}_{\text{DR}}$
Plug-and-Play Branch
Mean feature map
$\sigma_k = \texttt{Avg}_c\{\exp(-\lambda_0\|\mathbf{M}_k - \mathbf{M}_c\|_2)\}$
$N_k = \texttt{ceil}(N \cdot \sigma_k)$
Class-conditional Router
pattern distributions
(b) Class-Conditional Routing
A
h_{t-1}
h_t
$\mathbf{X}_{t-1}$
B_t
project
Discretize
Output
C_t
pattern 1
pattern 2
pattern N_k
pattern N
$w_{1t} \cdot \boldsymbol{\Delta}_{1t}$
$w_{2t} \cdot \boldsymbol{\Delta}_{2t}$
$w_{N_k t} \cdot \boldsymbol{\Delta}_{N_k t}$
$\boldsymbol{\Delta}_t$
Class-conditional Router
(c) S6 with Mixture of Discretization (S6MOD)
$\mathcal{L}_{\text{Cont}}$
Pull
Push
$\min\theta_1$
$\max\theta_2$
$\max\theta_3$
$\max\theta_4$
Push
Push
(d) Contrastive Discretization Loss

Figure 1. Framework of S6MOD. Our method (a) introduces a plug-and-play branch after the backbone, where features are learned through S6MOD and supervised by the ETF classifier to guide the base method. S6MOD (c) utilizes MoE to enhance the discretization of SSM and applies class-conditional routing (b) to dynamically adjust the discretization based on the uncertainty. Finally, we use a contrastive discretization loss (d) to supervise the learning of both generalizable and discriminative features.

where P represents the predicted distribution of the base method, and Q refers to the output prediction after the ETF classifier of our SSM branch, *i.e.,* $Q = \text{softmax}(\mathbf{W}_{\text{ETF}}\boldsymbol{\mu})$.

4.2. S6 with Mixture of Discretization

We integrate the selective space state model with a mixture of discretization, borrowing the concept of mixture of experts [10, 19, 38, 53]. Its structure is shown in Fig. 1 (c). Similar to the typical SSM structure, we also use an MLP layer to produce the projection matrices B and C from the input features, which can be expressed as follows:

$$\mathbf{B} = f_B(\hat{\mathbf{X}}), \quad \mathbf{C} = f_C(\hat{\mathbf{X}}), \tag{6}$$

where $\hat{\mathbf{X}}$ denotes the input of the selective scan module S6MOD, *i.e.,* $\hat{\mathbf{X}} = \text{SiLU}(\text{Conv}(\mathbf{X}))$. To enable selective SSM to capture a precise discretization pattern with limited context from online data, we develop a sparse MoE system for the discretization transformation $\boldsymbol{\Delta}$ in S6 models, utilizing specialized projection layers to enrich the discretization patterns and adaptively selecting the most sensitive discretization based on the current dynamics. Specifically, each discretization candidate is produced by a projection layer as follows:

$$\boldsymbol{\Delta}_i = f_{\Delta i}(\hat{\mathbf{X}}). \tag{7}$$

The input feature $\mathbf{X}$ is also fed into a sparse gating mechanism, which produces associated importance weights w_i corresponding to each discretization $\boldsymbol{\Delta}_i$. Through the class-conditional gating that will be introduced in Sec. 4.3, we dynamically control the number of discretization patterns, N_k, to be selected for each class k based on the input features $\hat{\mathbf{X}}$. Finally, the selected discretization patterns are aggregated with their importance weights as follows:

$$\boldsymbol{\Delta} = \sum_{i \in \Omega(\hat{\mathbf{X}})} w_i \cdot \boldsymbol{\Delta}_i, \quad |\Omega(\hat{\mathbf{X}})| = N_k, \tag{8}$$

where $\Omega(\hat{\mathbf{X}})$ is the index set of the top-N_k discretization candidates of $\hat{\mathbf{X}}$, according to their importance weights.

It's noteworthy that $\mathbf{\Delta}$ is important because it controls the decay rate during the state update process, enabling the model to flexibly select and retain important information when handling long sequences, while avoiding the accumulation of redundant and irrelevant information [13], especially in cases of rapid changing states and evolving systems, as exemplified by OCL. Our method with mixture of discretization facilitates the learning of a precise selective scan pattern with limited data context in OCL.

4.3. Class-Conditional Routing

With more discretization patterns selected, the model is able to allocate more capacity for learning new knowledge, but may cause a significant shift of the model that impairs the existing ability. If we can calculate N_k based on the misclassification probability of each class k, we can dynamically control the capacity of mixture of discretization based on the learning conditions of all classes. Therefore, in order to guide the mixture of discretization to strike a balance between maintaining the stability of old knowledge and fostering the plasticity for new ability, we further design class-conditional routing. During training, we maintain the feature prototypes $\mathbf{M} = \{\mathbf{M}_c\}$ by moving average, where $\mathbf{M}_c$ is the within-class feature mean for class c. Then, we estimate the class uncertainty σ_k based on the average margin of class k with different classes,

$$\sigma_k = \texttt{Avg}|_c\{\exp(-\lambda_0\|\mathbf{M}_k - \mathbf{M}_c\|_2)\}, (9)$$

where λ_0 is a hyper-parameter and σ_k denotes the class uncertainty for class k ranging from $(0, 1)$. A large σ_k indicates that the class center k has narrow margins with the other class centers, and thus tends to be misclassified, while a small σ_k happens when class center k is distant from the other class centers with a lower likelihood of being misclassified. In inference, we replace $\mathbf{M}_k$ with the input feature $\hat{\mathbf{X}}$ and calculate its uncertainty by Eq. (9) with the feature prototypes $\mathbf{M}_c$ of all classes.

Then, we multiply the uncertainty by the number of total discretization candidates N, to get the discretization number to select for class k or input feature $\hat{\mathbf{X}}$ as follows,

$$N_k = \texttt{ceil|(N } \cdot \sigma_k), \tag{10}$$

where $\texttt{ceil}$ refers to the operation that rounds up $N \cdot \sigma_k$ to the nearest integer.

The institution is that classes that are more prone to misclassification have smaller margins with the other classes in the feature space, and thus needs to aggregate more discretization patterns to allocate additional capacity for adapting to these undeveloped abilities. Conversely, for categories that are easier to classify with low uncertainty, a smaller N_k is favored to only include the most likely discretization patterns, which can spare optimization efforts for the uncertain classes and stabilize these already acquired abilities. The combination of class-conditional routing and our mixture of discretization achieves dynamic structures dependent on input features and classes, such that the final prediction of each class selectively optimizes the corresponding discretization patterns.

To further reduce interaction among different classes and facilitate the learning of generalizable and discriminative features, we also incorporate contrastive discretization loss function $\mathcal{L}_{\text{Cont}}$, as shown in Fig. 1 (d). It encourages $\mathbf{\Delta}$ to have within-class consistency and between-class diversity as follows:

$$\mathcal{L}_{\text{Cont}} = -\frac{1}{B^2}\sum_{m=1}^{B}\sum_{n=1}^{B}(1_{y_m=y_n} - 1_{y_m\neq y_n})\frac{\mathbf{\Delta}_m \cdot \mathbf{\Delta}_n}{\|\mathbf{\Delta}_m\|\|\mathbf{\Delta}_n\|} \tag{11}$$

where B is the batch size, $\mathbf{\Delta}_m$ represents the aggregated $\mathbf{\Delta}$ by Eq. (8) for input m, y_m denotes the class of m and $1_{y_m=y_n}$ is an indicator function that takes a value of 1 when the condition $y_m = y_n$ holds, and 0 otherwise.

4.4. Optimization

In addition to the previously mentioned loss functions in Eqs. (5) and (11), we also directly used the classification loss function $\mathcal{L}_{\text{DR}}$ specifically designed for ETF classifier [50, 51] to supervise our introduced plug-and-play branch, and the loss function $\mathcal{L}_z$ for maintaining load balancing among the discretization patterns [57]. In summary, when integrating S6MOD on a base method, the overall loss function can be expressed as follows:

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{base}} + \mathcal{L}_{\text{S6MOD}}, \tag{12}$$

$$\mathcal{L}_{\text{S6MOD}} = \mathcal{L}_{\text{DR}} + \alpha \cdot \mathcal{L}_{\text{Diff}} + \beta \cdot \mathcal{L}_{\text{Cont}} + \mathcal{L}_z, \tag{13}$$

where α and β are hyperparameters.

5. Experiments

5.1. Experimental Setups

Datasets. We test three datasets that are widely used in OCL, i.e., CIFAR-10 (10 classes) [21], CIFAR-100 (100 classes) [21] and TinyImageNet (200 classes) [22]. The dataset settings we adopted are the same as those in CCLDC[45]. We divide CIFAR-10 into 5 tasks, with 2 classes per task; CIFAR-100 into 10 tasks, with 10 classes per task; and TinyImageNet into 100 tasks, with 2 classes per task. Further details about the datasets will be provided in the supplementary materials.

Table 1. Average Accuracy (%, higher is better) on three benchmark datasets with difference memory buffer size M, with and without our proposed S6MOD module. All values are averages of 10 runs.

Dataset	CIFAR10		CIFAR100			Tiny-ImageNet		
Memory Size M	500	1000	1000	2000	5000	2000	5000	10000
ER [33]	56.68±1.89	62.32±4.13	24.47±0.72	31.89±1.45	39.41±1.81	10.82±0.79	19.16±1.42	24.71±2.52
ER + Ours	**57.88±3.30**	**65.80±2.16**	**26.50±2.23**	**34.55±1.66**	**39.61±3.16**	**10.94±1.47**	**19.67±1.36**	**25.62±1.73**
OCM [17]	68.19±1.75	73.15±1.05	28.02±0.74	35.69±1.36	42.22±1.06	18.36±0.95	26.74±1.02	31.94±1.19
OCM + Ours	**70.57±1.14**	**75.31±1.10**	**32.44±1.35**	**38.97±2.28**	**45.49±1.58**	**19.12±1.60**	**27.20±1.83**	**32.31±1.72**
OnPro [45]	70.47±2.12	74.70±1.51	27.22±0.77	33.33±0.93	41.59±1.38	14.32±1.40	21.13±2.12	26.38±2.18
OnPro + Ours	**72.30±1.08**	**75.14±0.99**	**28.84±0.59**	**37.57±0.81**	**44.14±1.36**	**17.27±0.65**	**25.62±0.79**	**29.20±1.00**
OCM-CCLDC [43]	74.14±0.85	77.66±1.46	35.00±1.15	43.34±1.51	51.43±1.37	23.36±1.18	33.17±0.97	39.25±0.88
OCM-CCLDC + Ours	**74.45±1.20**	**78.21±1.03**	**36.02±1.77**	**44.40±2.26**	**52.53±0.21**	**23.68±1.02**	**33.59±0.72**	**39.53±1.02**
OnPro-CCLDC [43]	74.49±2.14	78.64±1.42	34.76±1.12	41.89±0.82	50.01±0.85	21.81±1.02	32.00±0.72	38.18±1.02
OnPro-CCLDC + Ours	**75.03±1.03**	**79.51±0.63**	**36.07±0.76**	**43.92±0.96**	**50.85±0.52**	**22.03±0.82**	**33.29±0.35**	**38.51±0.91**
MOSE [49]	61.02±1.47	70.74±1.18	35.05±0.34	45.06±0.32	54.53±0.78	18.23±0.73	30.98±0.63	38.71±0.44
MOSE + Ours	**63.74±1.55**	**72.54±0.14**	**35.43±0.62**	**45.38±0.31**	**54.75±0.52**	**19.03±0.83**	**32.03±0.81**	38.18±1.02
MOE-MOSE [49]	62.54±1.59	72.18±1.29	37.32±0.34	47.03±0.57	55.62±0.72	20.61±0.69	32.52±0.33	38.41±0.53
MOE-MOSE + Ours	**63.86±1.19**	**73.16±0.53**	**37.76±0.51**	**47.25±0.43**	**56.32±0.53**	**21.03±0.77**	**33.53±0.51**	**40.11±0.26**

Table 2. Average Forgetting (%, lower is better) on three benchmark datasets with difference memory buffer size M, with and without our proposed S6MOD module. All values are averages of 10 runs.

Dataset	CIFAR10		CIFAR100			Tiny-ImageNet		
Memory Size M	500	1000	1000	2000	5000	2000	5000	10000
ER [33]	33.16±3.50	20.94±6.79	32.65±1.78	22.20±2.26	13.29±1.98	58.38±1.69	46.87±1.60	40.77±2.45
ER + Ours	**31.25±3.44**	**16.71±2.86**	**30.96±2.07**	**19.02±2.13**	**12.97±2.61**	**58.03±2.19**	**45.83±1.49**	**37.62±1.53**
OCM [17]	13.68±4.25	11.63±2.62	14.99±1.55	9.16±1.75	3.76±1.16	26.12±1.63	19.74±1.30	15.92±1.47
OCM + Ours	**13.58±3.26**	**9.68±3.47**	15.99±1.82	11.60±1.49	5.46±0.98	**24.23±2.07**	**19.12±1.87**	**14.66±1.75**
OnPro [45]	17.94±3.69	14.20±2.60	16.76±2.47	12.42±1.39	6.72±0.94	28.01±1.59	23.52±1.75	20.32±1.70
OnPro + Ours	**7.13±1.44**	**5.73±1.81**	**16.03±1.69**	**9.81±1.32**	**5.07±0.89**	**20.92±1.12**	**16.82±0.97**	**18.15±1.91**
OCM-CCLDC [43]	11.59±2.24	9.18±2.03	16.69±2.36	10.07±1.37	3.99±0.78	26.16±1.90	19.99±1.96	15.56±1.06
OCM-CCLDC + Ours	15.81±2.69	11.28±1.83	**16.64±1.85**	**9.08±2.06**	**3.48±0.31**	**25.29±1.95**	**14.89±0.74**	**11.55±1.31**
OnPro-CCLDC [43]	19.89±4.01	14.62±2.75	28.93±2.19	20.23±1.03	10.55±1.89	28.21±1.58	20.85±1.13	16.17±0.63
OnPro-CCLDC + Ours	**17.23±3.16**	**11.51±2.33**	**22.26±1.18**	**12.49±1.59**	**4.88±1.36**	**26.73±1.51**	**16.69±0.31**	**12.23±1.37**
MOSE [49]	30.36±1.69	20.27±1.27	37.54±0.43	25.89±0.45	13.60±0.59	47.16±1.41	24.96±0.62	15.51±0.33
MOSE + Ours	**27.38±1.94**	**17.92±0.34**	**37.09±0.63**	**25.80±0.39**	15.55±0.61	**46.60±1.07**	**24.39±0.72**	20.15±0.74
MOE-MOSE [49]	29.39±1.79	19.24±1.49	35.17±0.30	23.99±0.51	12.81±0.74	41.98±1.46	22.22±0.38	13.94±0.59
MOE-MOSE + Ours	**28.13±1.41**	**17.98±0.56**	**34.76±0.66**	**23.84±0.77**	14.03±0.48	**41.26±1.02**	**21.39±0.58**	14.91±0.28

Baselines. To demonstrate the effectiveness and applicability of our module, we conduct tests on 7 typical and state-of-the-art methods, including ER [33], OCM [17], OnPro [45], OCM-CCLDC [43], OnPro-CCLDC [43], MOSE[49] and MOE-MOSE [49].

Implementation details. To ensure a fair comparison, we applied the same hyperparameter settings to each baseline and the methods combined with our module (S6MOD). All the above methods use ResNet-18 as the backbone, without pretraining. For the streaming input data, we set the batch size to 10, and for the samples drawn from the buffer, the batch size is set to 64. We retain each baseline's original data augmentation methods without modifying the base methods. For more details, please refer to the supplementary materials.

5.2. Results

We combine our method with both classical and state-of-the-art approaches on CIFAR-10, CIFAR-100, and Tiny-ImageNet. The experimental results in Table 1 demonstrate the universality of our method, as it consistently improves accuracy across numerous baselines. On CIFAR-10 and CIFAR-100, our method generally leads to a 1% improvement, while on Tiny-ImageNet, we achieve significant gains under OCM, OnPro and MOE-MOSE. Notably, in settings like CIFAR100 ($M = 2k$) and Tiny-ImageNet ($M = 5k$), OnPro result in an approximate 4.2% and 4.5% improvement, respectively. It is also worth mentioning that even for state-of-the-art distillation-based methods

Table 3. Ablation studies on CIFAR-100 ($M = 2k$). The "branch" refers to using SS2D[26] as the core of the S6MOD, without routing and $\mathcal{L}_{\text{Cont}}$. "routing" represents class-conditional routing. S6MOD refers to the composition of (branch + class-conditional routing + $\mathcal{L}_{\text{Cont}}$). All values are averaged over 10 runs.

Method	Acc. ↑	AF ↓
OnPro	$33.33_{\pm 0.93}$	$12.42_{\pm 1.39}$
OnPro + branch (with $\mathcal{L}_{\text{Diff}}$)	$36.61_{\pm 1.04}$	$10.28_{\pm 1.59}$
OnPro + branch + routing	$36.92_{\pm 0.72}$	$8.85_{\pm 0.81}$
OnPro + branch + $\mathcal{L}_{\text{Cont}}$	$36.93_{\pm 0.84}$	$9.70_{\pm 1.42}$
OnPro + S6MOD	$37.57_{\pm 0.81}$	$9.81_{\pm 1.32}$

like CCLDC and MOE-MOSE, the performance improvements are quite substantial. For instance, OnPro-CCLDC with the S6MOD integration achieve an impressive 79.51% on CIFAR10 ($M = 1k$), and MOE-MOSE with S6MOD reach 56.32% on CIFAR100 ($M = 5k$) and 40.1% on Tiny-ImageNet ($M = 10k$).

In addition, as shown in Table 2, our module is also highly effective in reducing model forgetting, achieving lower forgetting rates in most settings, with some cases showing particularly significant improvements. This can be attributed to our class-conditional routing. In a few baseline settings, the inclusion of our module led to a higher forgetting rate, but this does not imply that the model is more prone to forgetting. In these cases, we think the higher forgetting rates are often due to the stronger learning capabilities of the model. Analyzing both Table 1 and Table 2, we can find that even in these settings with higher forgetting rates, the accuracy performance remains quite competitive.

5.3. Ablation Studies

As mentioned in Sec. 1, the S6MOD module we propose is very simple and can be directly applied to existing baselines. By adding an extra branch and using the $\mathcal{L}_{\text{Diff}}$ to supervise the original classification head, we can achieve a significant improvement. To demonstrate this, we conduct an ablation study where we only add the extra branch and $\mathcal{L}_{\text{Diff}}$ to the baseline. The results in Table 3 confirm this, showing a 3.3% improvement with supervision from the extra branch alone.

In addition, to verify the effectiveness and adaptability of class-conditional routing and contrastive discretization loss $\mathcal{L}_{\text{Cont}}$, we conduct separate ablation experiments. As shown in Table 3, when added to the base method with the extra branch, the improvements brought by adding class-conditional routing or $\mathcal{L}_{\text{Cont}}$ alone are not significant, with accuracy gains of only 0.31% and 0.32%, respectively. However, when both components are used together, the improvement becomes significant, with an approximate 1% increase in accuracy, and the forgetting rate still decreases. This indicates that when our module is correctly combined, it can significantly enhances the model's adaptability by learning more generalizable and discriminative features through dynamic parameters.

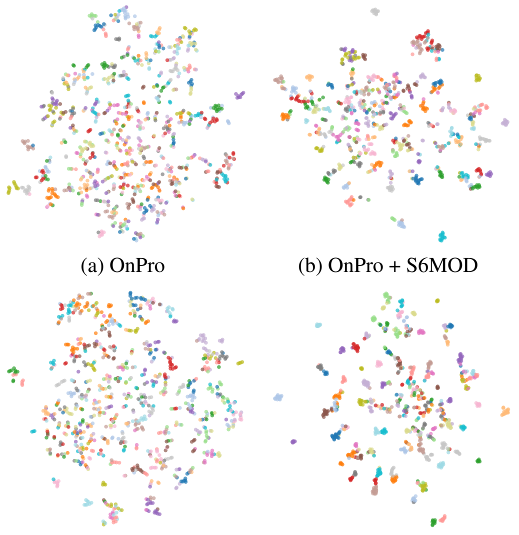
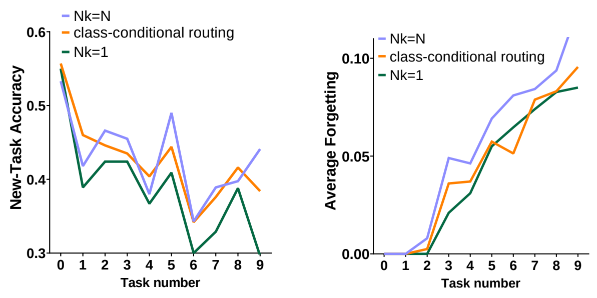


Figure 2. t-SNE visualization of memory data at the end of training on CIFAR-100 ($M = 2k$), showcasing baseline in (a) and (c) and baseline combined with S6MOD in (b) and (d). Different colors represent different classes.

5.4. Analysis

Analysis of Feature Embeddings. To verify that S6MOD improves the adaptability of the baseline and helps it learn more discriminative features, we use t-SNE [39] visualization for analysis. We compare with the baseline under the CIFAR-100 ($M = 2k$) setting. As shown in Fig. 2, we visualize the features of the samples in the buffer after the final training, which are the features $\mathbf{F}$ output by the backbone just before they are passed to the classifier for classification. The features shown in Fig. 2 (a) and (c) are very scattered, with only a few classes showing some degree of aggregation. Most classes do not have clear separations between other classes, making it difficult to distinguish. In contrast, Fig. 2 (b) and (d) present the feature distribution after adding our module. We can observe that many categories have noticeably converged, forming small clusters with consistent intra-class compactness and clear inter-class separations. These features are much easier to distinguish and more conducive to classification by the classifier, which aligns with the significant improvements we demonstrate in the experiments.

Analysis of Class-conditional Routing. We further investigate the effectiveness of class-conditional routing. Under the CIFAR-100 ($M = 2k$) setting, we design experiments to compare the impact of a fixed N_k and dynamically calculated N_k using class-conditional routing. Specifically, we set $N_k = N$, meaning all patterns are activated, and $N_k = 1$, meaning only the pattern with the highest prob-



(a) New-Task Accuracy (b) Average Forgetting

Figure 3. Impact of dynamically selecting different N_k values on the ability to learn new tasks and prevent forgetting of old tasks: New-Task accuracy represents the model's accuracy on the current task. We conduct experiments by setting $N_k = 1$, $N_k = N$, and calculating N_k through class-conditional routing. The dataset used is CIFAR-100 ($M = 2k$).

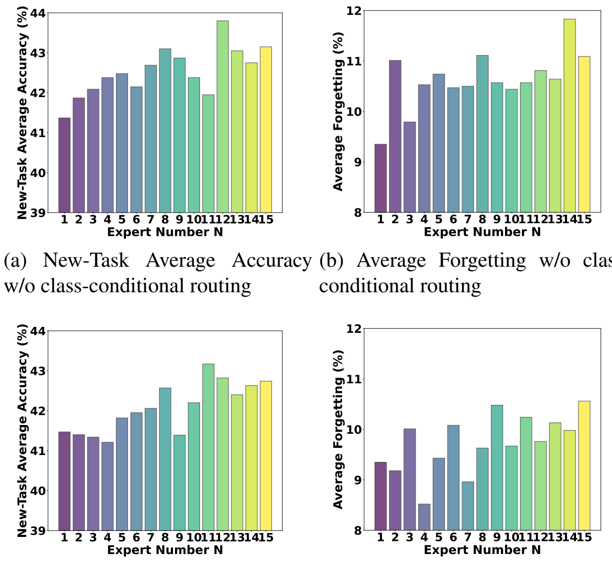
ability is activated. To facilitate analysis, we present the accuracy of the model on the latest task (the current task) in Fig. 3 (a). As shown in Fig. 3 (a), when $N_k = N$, all patterns are activated, and thanks to the inclusion of more patterns with different weights and larger parameters, the model has greater capacity to adapt to new tasks, resulting in a stronger ability to learn new knowledge. When $N_k = 1$, only the pattern with the highest probability is activated, significantly reducing the model's ability to learn the new task. However, as shown in Fig. 3 (b), when $N_k = N$, since all patterns are updated by the current task, the model has a higher risk of forgetting. When $N_k = 1$, only the pattern with the highest probability is updated, and tasks do not interfere with each other, so forgetting is less likely. Combining the analysis of the entire Fig. 3, the effect of our proposed class-conditional routing is both intuitive and significant. It not only helps the model achieve strong learning ability for new tasks, comparable to the N_k setting, but also provides excellent anti-forgetting performance, similar to $N_k = 1$. This is due to the dynamic adjustment of N_k based on uncertainty: for samples that are prone to misclassification, class-conditional routing assigns a larger N_k, enabling more extensive learning. For easier-to-classify samples, the model's learning ability is higher, and class-conditional routing assigns a smaller N_k to enhance anti-forgetting.

Sensitivity Analysis of Hyper-parameters α and β. α and β are the weights for $\mathcal{L}_{\text{Diff}}$ and $\mathcal{L}_{\text{Cont}}$, respectively. We conduct experiments on the sensitivity of them, and the results are shown in Table 4. It can be observed that when the weights are low, the optimization effect of the loss is relatively weak, both accuracy and forgetting rate do not perform well. Gradually increasing the weights, we find that after a certain point, the impact of the weights on the results becomes relatively stable, with no more drastic changes.

Table 4. Impact of hyper-parameters α and β on performance in the CIFAR-100 dataset ($M = 2k$). α and β are the weights for $\mathcal{L}_{\text{Diff}}$ and $\mathcal{L}_{\text{Cont}}$, respectively. Acc. and AF represent the average accuracy and the average forgetting over 5 runs, respectively.

Impact of hyper-parameter α					
Value	0.01	0.1	0.5	1	5
Acc.	36.80±1.17	37.09±1.09	37.55±1.08	37.47±0.94	37.57±0.81
AF	10.49±1.01	9.65±0.39	9.05±1.25	9.16±2.11	9.81±1.32

Impact of hyper-parameter β					
Value	0.1	1	5	10	25
Acc.	36.50±1.24	37.30±1.03	37.40±0.33	37.26±0.61	37.57±0.81
AF	10.24±0.43	10.12±1.09	10.10±0.60	9.98±0.66	9.81±1.32



(a) New-Task Average Accuracy w/o class-conditional routing (b) Average Forgetting w/o class-conditional routing

(c) New-Task Average Accuracy (d) Average Forgetting

Figure 4. Impact of pattern number N. The dataset used is CIFAR-100 ($M = 2k$).

Sensitivity Analysis of total discretization patterns number N. We conduct experiments on the total number of discretization patterns in the module. The results are shown in Fig. 4. As the number N increases, the model gains a stronger ability to learn new tasks, but at the same time, its tendency to forget becomes more pronounced. Furthermore, by comparing (a)(b) and (c)(d) in Fig. 4, we can conclude that class-conditional routing can significantly reduce forgetting with only a slight decrease in new task accuracy, which in turn leads to an overall improvement in performance. This indirectly supports the effectiveness of class-conditional routing in our ablation studies.

6. Conclusion

In this paper, we propose a plug-and-play module S6MOD to enhance the adaptability of existing OCL methods by introducing selective state space models (SSMs) with a class-conditional mixture of discretization. By integrating a dynamic discretization mechanism and leveraging class-conditional strategies, our method can efficiently allocate discretization patterns based on class uncertainty, improving both the model's generalization and its ability to adapt to new data. Experimental results on multiple OCL datasets demonstrate that our method consistently optimize existing techniques, contributing to more robust OCL systems.

References

[1] Assyr Abdulle, E Weinan, Björn Engquist, and Eric Vanden-Eijnden. The heterogeneous multiscale method. *Acta Numerica*, 21:1–87, 2012. 2

[2] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, pages 139–154, 2018. 2

[3] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *NeurIPS*, 32, 2019. 1

[4] Quentin Anthony, Yury Tokpanov, Paolo Glorioso, and Beren Millidge. Blackmamba: Mixture of experts for state-space models. *arXiv preprint arXiv:2402.01771*, 2024. 3

[5] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *NeurIPS*, 33: 15920–15930, 2020. 2

[6] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, pages 532–547, 2018. 1

[7] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018. 2

[8] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023. 1

[9] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *PAMI*, 44(7):3366–3385, 2021. 1, 2

[10] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314*, 2013. 4

[11] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. 3

[12] Qiang Gao, Xiaojun Shan, Yuchen Zhang, and Fan Zhou. Enhancing knowledge transfer for task incremental learning with data-free subnetwork. In *Advances in Neural Information Processing Systems*, pages 68471–68484. Curran Associates, Inc., 2023. 2

[13] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 1, 2, 3, 5

[14] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. 2, 3

[15] John Guckenheimer and Philip Holmes. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Springer Science & Business Media, 2013. 2

[16] Nuwan Gunasekara, Bernhard Pfahringer, Heitor Murilo Gomes, and Albert Bifet. Survey on online streaming continual learning. In *IJCAI*, pages 6628–6637, 2023. 1, 2

[17] Yiduo Guo, Bing Liu, and Dongyan Zhao. Online continual learning through mutual information maximization. In *ICML*, 2022. 1, 2, 6

[18] Yiduo Guo, Bing Liu, and Dongyan Zhao. Dealing with cross-task class discrimination in online continual learning. In *CVPR*, 2023. 1, 2

[19] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. 4

[20] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017. 2

[21] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 5

[22] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015. 5

[23] Xiaojie Li, Yibo Yang, Jianlong Wu, Bernard Ghanem, Liqiang Nie, and Min Zhang. Mamba-fscil: Dynamic adaptation with selective state space model for few-shot class-incremental learning. *arXiv preprint arXiv:2407.06136*, 2024. 1, 2, 3

[24] Zhizhong Li and Derek Hoiem. Learning without forgetting. *PAMI*, 40(12):2935–2947, 2017. 2

[25] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. 1, 3

[26] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model. *ArXiv*, abs/2401.10166, 2024. 1, 2, 3, 7

[27] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *NeurIPS*, 30, 2017. 2

[28] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022. 1, 2

[29] Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022. 3

[30] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019. 1, 2

[31] Maciej Pióro, Kamil Ciebiera, Krystian Król, Jan Ludziejewski, Michał Krutul, Jakub Krajewski, Szymon Antoniak, Piotr Miłoś, Marek Cygan, and Sebastian Jaszczur. Moe-mamba: Efficient selective state space models with mixture of experts. *arXiv preprint arXiv:2401.04081*, 2024. 3

[32] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, pages 2001–2010, 2017. 1, 2

[33] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *NeurIPS*, 32, 2019. 1, 2, 6

[34] Jiacheng Ruan and Suncheng Xiang. Vm-unet: Vision mamba unet for medical image segmentation. *arXiv preprint arXiv:2402.02491*, 2024. 1

[35] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 2

[36] Minhyuk Seo, Hyunseo Koh, Wonje Jeung, Minjae Lee, San Kim, Hankook Lee, Sungjun Cho, Sungik Choi, Hyunwoo Kim, and Jonghyun Choi. Learning equi-angular representations for online continual learning. In *CVPR*, pages 23933–23942, 2024. 2

[37] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, pages 4548–4557. PMLR, 2018. 2

[38] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. 4

[39] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9 (86):2579–2605, 2008. 7

[40] A Vaswani. Attention is all you need. *NeurIPS*, 2017. 2

[41] Liyuan Wang, Mingtian Zhang, Zhongfan Jia, Qian Li, Chenglong Bao, Kaisheng Ma, Jun Zhu, and Yi Zhong. Afec: Active forgetting of negative transfer in continual learning. *NeurIPS*, 34:22379–22391, 2021. 2

[42] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: theory, method and application. *PAMI*, 2024. 1, 2

[43] Maorong Wang, Nicolas Michel, Ling Xiao, and Toshihiko Yamasaki. Improving plasticity in online continual learning via collaborative learning. In *CVPR*, pages 23460–23469, 2024. 1, 2, 6

[44] Yujie Wei, Jiaxin Ye, Zhizhong Huang, Junping Zhang, and Hongming Shan. Online prototype learning for online continual learning. In *ICCV*, pages 18764–18774, 2023. 1

[45] Yujie Wei, Jiaxin Ye, Zhizhong Huang, Junping Zhang, and Hongming Shan. Online prototype learning for online continual learning. In *ICCV*, 2023. 1, 2, 5, 6

[46] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *NeurIPS*, 33:15173–15184, 2020. 2

[47] Liang Xie, Yibo Yang, Deng Cai, and Xiaofei He. Neural collapse inspired attraction–repulsion-balanced loss for imbalanced learning. *Neurocomputing*, 527:60–70, 2023. 2

[48] Zhaohu Xing, Tian Ye, Yijun Yang, Guang Liu, and Lei Zhu. Segmamba: Long-range sequential modeling mamba for 3d medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 578–588. Springer, 2024. 1

[49] Hongwei Yan, Liyuan Wang, Kaisheng Ma, and Yi Zhong. Orchestrate latent expertise: Advancing online continual learning with multi-level supervision and reverse self-distillation. In *CVPR*, pages 23670–23680, 2024. 1, 2, 6

[50] Yibo Yang, Shixiang Chen, Xiangtai Li, Liang Xie, Zhouchen Lin, and Dacheng Tao. Inducing neural collapse in imbalanced learning: Do we really need a learnable classifier at the end of deep neural network? In *NeurIPS*, pages 37991–38002. Curran Associates, Inc., 2022. 2, 3, 5

[51] Yibo Yang, Haobo Yuan, Xiangtai Li, Zhouchen Lin, Philip Torr, and Dacheng Tao. Neural collapse inspired feature-classifier alignment for few-shot class incremental learning. *arXiv preprint arXiv:2302.03004*, 2023. 2, 3, 5

[52] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 2

[53] Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193, 2012. 4

[54] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pages 3987–3995. PMLR, 2017. 2

[55] Zhisheng Zhong, Jiequan Cui, Yibo Yang, Xiaoyang Wu, Xiaojuan Qi, Xiangyu Zhang, and Jiaya Jia. Understanding imbalanced semantic segmentation through neural collapse. In *CVPR*, pages 19550–19560, 2023. 2

[56] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024. 1, 2

[57] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022. 5

Enhancing Online Continual Learning with Plug-and-Play State Space Model and Class-Conditional Mixture of Discretization

Supplementary Material

A. Implementation details

A.1. Training details

In Table 5, we provide the hyper-parameter settings for our method when ER is used as the baseline. As shown in the table, for the same dataset, we tend to set the total number of patterns N to a fixed value and set α and β to a ratio of $1:5$. When we need to reduce the impact of our module, we can proportionally decrease the weights. Actually, different hyper-parameter settings help to unleash the potential of various classifiers (linear classifier, ETF classifier, NCM classifier). hyper-parameters not mentioned in the table remain consistent with the original baseline.

A.2. Dataset

As stated in Sec. 5 (Experiments), we primarily conduct experimental validation on three datasets: CIFAR10, CIFAR100, and TinyImageNet. It is important to note that the sample sizes and the number of classes vary across these datasets, which may lead to the use of different hyper-parameters in our method. Our experimental implementation follows the guidelines of CCLDC [43]. Specifically:

CIFAR-10 is a dataset composed of 10 classes, which we divide into 5 tasks, with each task containing 2 classes. It includes a total of 50,000 training samples and 10,000 test samples, with image dimensions of 32×32.

CIFAR-100 consists of 100 classes, divided into 10 tasks, with each task containing 10 classes. It also contains 50,000 training samples and 10,000 test samples, with image dimensions of 32×32.

TinyImageNet comprises 200 classes, divided into 100 tasks, with each task containing 2 classes. It includes 100,000 training samples and 10,000 test samples, with image dimensions of 64×64.

A.3. Pseudo-code

To facilitate understanding and usage of our proposed plug-and-play module, S6MOD, we provide pseudo-code in Algorithm 1 to demonstrate how to integrate S6MOD with the current baseline. For simplicity, we omit the workflows of $\mathcal{L}_{\text{DR}}$ and $\mathcal{L}_{\text{z}}$, as well as the samples in the memory buffer.

Algorithm 1 PyTorch-like pseudo-code of S6MOD to integrate to other baselines.

```
# model: the whole model
# model.logits: logit function of model (base
↪  classification)
# model.S6MOD: obtain features using S6MOD
# model.ETF: ETF logit function of model (ETF
↪  classification)
# cos_sim: cosine similarity calculation function
# optim: optimizer for model
for x, y in dataloader:
  # Baseline loss
  pred_base = model.logits(x)
  loss_base = criterion_baseline(model, x, y)

  # S6MOD loss
  fea, deltas = model.S6MOD(x)
  pred_etf = model.ETF(fea)

  loss_Diff = kl_div(pred_base, pred_etf)
  loss_Cont = 0
  for i in range(len(y)):
    for j in range(i+1, len(y)):
        if y[i]==y[j]:
            loss_Cont -= cos_sim(deltas[i], deltas[j])
        else:
            loss_Cont += cos_sim(deltas[i], deltas[j])


  # hyperparameters alpha and beta
  loss_S6MOD = loss_DR + alpha*loss_Diff +
  ↪  beta*loss_Cont + loss_z
  loss = loss_base + loss_S6MOD

  optim.zero_grad()
  loss.backward()
  optim.step()
```

A.4. Metrics

We use three commonly employed evaluation metrics Average Accuracy (Acc), Average Forgetting (AF) and New-Task Average Accuracy (N-Acc) in the main text [43], and we will introduce their definitions in detail here.

In continual learning, after each task t is completed, the model needs to be tested on all previously learned tasks $\{1, 2, \ldots, t\}$. The Acc is defined as:

$$Acc_T = \frac{1}{T}\sum_{t=1}^{T} A_{t,T}, \tag{14}$$

where T is the total number of tasks, and $A_{t,T}$ is the test accuracy on task t after learning task T:

$$A_{t,T} = \frac{\sum_{i=1}^{N_t} 1(\hat{y}_{i,t} = y_{i,t})}{N_t}. \tag{15}$$

Here, N_t is the number of samples in task t, $\hat{y}_{i,t}$ is the predicted class of the i-th sample, and $y_{i,t}$ is the true class of

Dataset	CIFAR10		CIFAR100			Tiny-ImageNet		
Memory Size M	500	1000	1000	2000	5000	2000	5000	10000
N	10	10	8	8	8	10	10	10
α	1	1	1	1	1	1	1	0.5
β	5	5	5	5	5	5	5	2.5

Table 5. The hyper-parameter settings for our S6MOD on ER.

the i-th sample.

The Average Forgetting (AF) is the average of the forgetting rates over all tasks. It provides an overall measure of how much the model forgets across all previously learned tasks as new tasks are added. A low AF indicates that the model effectively retains knowledge from previous tasks, while a high AF suggests that the model suffers from significant forgetting when learning new tasks. AF is defined as:

$$AF = \frac{1}{T-1}\sum_{t=2}^{T} FR_t, \tag{16}$$

where T is the total number of tasks. FR_t is the Forgetting Rate for task t, defined as:

$$FR_t = \max_{i\in\{1,\ldots,t-1\}} \left(A_{i,i} - A_{i,t}\right), \tag{17}$$

where $A_{i,i}$ is the accuracy on task i after learning task i, and $A_{i,t}$ is the accuracy on task i after learning task t. The AF is averaged over all tasks after the first one, as the first task does not cause any forgetting.

The New-Task Average Accuracy (N-Acc) is the average accuracy of the model on all tasks when they are first learned. This metric provides an overall measure of how well the model performs on each task at the time it is introduced, without considering any changes in performance as other tasks are learned later. N-Acc is defined as:

$$\text{N-Acc} = \frac{1}{T}\sum_{t=1}^{T} A_{t,t}, \tag{18}$$

where T is the total number of tasks and $A_{t,t}$ is the accuracy on task t immediately after task t is learned, *i.e.*, when the model first encounters the task. This metric directly reflects the model's ability to learn new tasks.

B. Extra Experiments

B.1. Performance with NCM classifier.

The Nearest Class Mean (NCM) classifier is a simple yet effective classification method, often used as a component in continual learning scenarios. To further demonstrate that our method also learns more generalizable and discriminative features with NCM, we use an NCM classifier to test our method. As shown in Table 6 and Table 7, our method achieves superior performance when using the NCM classifier. This indicates that our method is also compatible with NCM classifier to learn more discriminative features.

Method	NCM Acc. ↑	Logit Acc. ↑
ER	$64.31_{\pm 0.98}$	$62.32_{\pm 4.13}$
ER + Ours	$67.24_{\pm 2.32}$	$65.80_{\pm 2.16}$
OCM	$72.47_{\pm 1.04}$	$73.15_{\pm 1.05}$
OCM + Ours	$76.36_{\pm 0.66}$	$75.31_{\pm 1.10}$
OCM-CCLDC	$74.80_{\pm 1.72}$	$77.66_{\pm 1.46}$
OCM-CCLDC + Ours	$79.37_{\pm 0.89}$	$78.21_{\pm 1.03}$

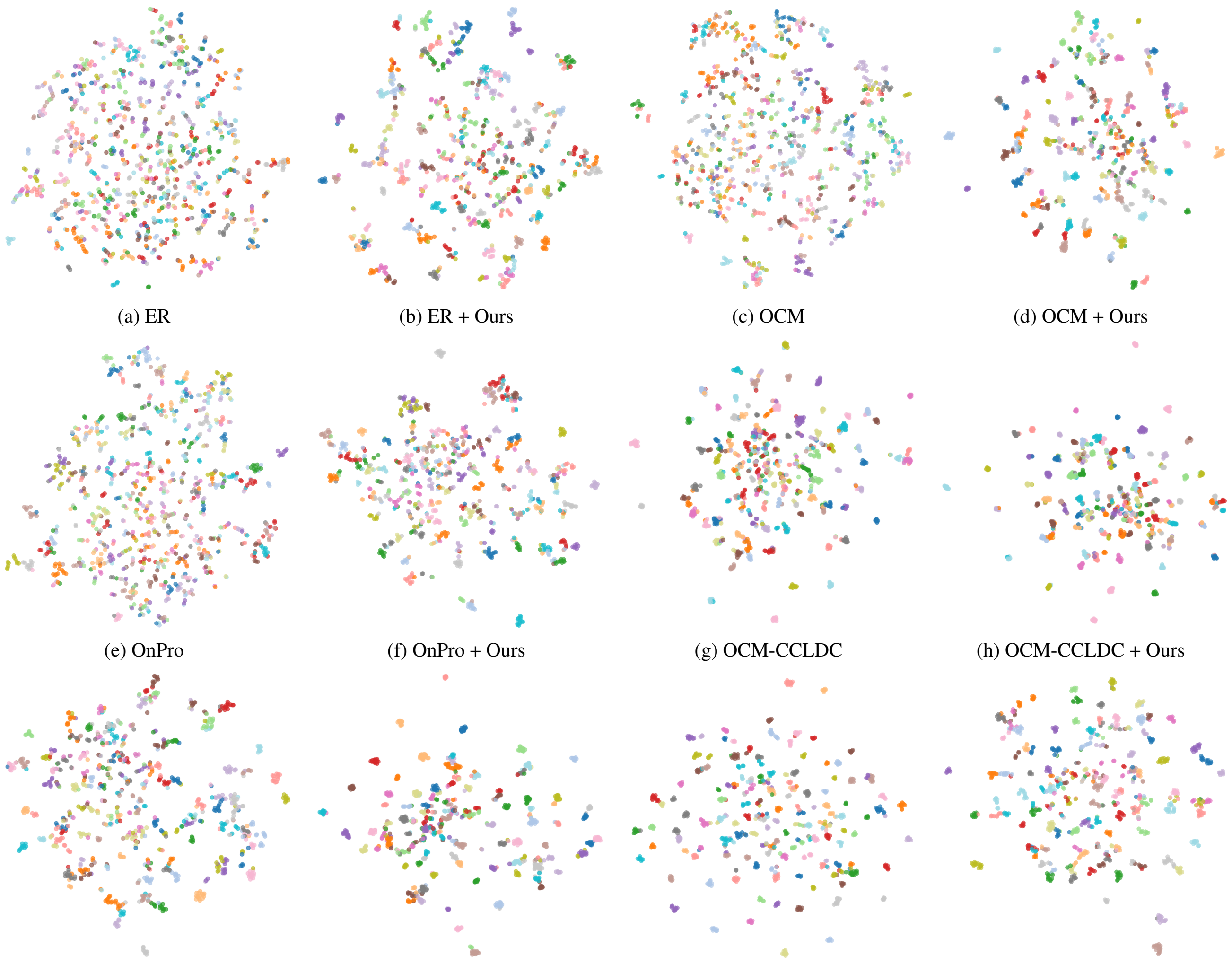
Table 6. Final average accuracy on CIFAR-10 ($M = 1k$), with and without our method on NCM and Logit predictions.

Method	NCM Acc. ↑	Logit Acc. ↑
ER	$36.40_{\pm 0.81}$	$31.89_{\pm 1.45}$
ER + Ours	$36.99_{\pm 0.65}$	$34.55_{\pm 1.66}$
OCM	$37.76_{\pm 0.70}$	$35.69_{\pm 1.36}$
OCM + Ours	$39.98_{\pm 1.19}$	$38.97_{\pm 2.28}$
OCM-CCLDC	$40.28_{\pm 1.08}$	$43.34_{\pm 1.51}$
OCM-CCLDC + Ours	$44.55_{\pm 1.42}$	$44.40_{\pm 2.26}$

Table 7. Final average accuracy on CIFAR-100 ($M = 2k$), with and without our method on NCM and Logit predictions..

B.2. More T-SNE visualization.

As described in Sec. 1 (Introduction) and demonstrated in "Analysis of Feature Embedding," incorporating S6MOD helps the model learn more generalizable and discriminative features. To further validate this, we present comprehensive t-SNE visualizations in Fig. 5, explicitly showcasing the superiority of our method on more baseline methods. Given that the MOSE and MOE-MOSE structures are identical, with the only difference being during inference, we only report the features of MOSE here.



(a) ER (b) ER + Ours (c) OCM (d) OCM + Ours

(e) OnPro (f) OnPro + Ours (g) OCM-CCLDC (h) OCM-CCLDC + Ours

(i) OnPro-CCLDC (j) OnPro-CCLDC + Ours (k) MOSE (l) MOSE + Ours

Figure 5. T-SNE visualization of features before classification of memory data at the end of training on CIFAR-100 ($M = 2k$).