# 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 $\alpha$ and $\beta$ 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 [44]. 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 [44], 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 |
| $\alpha$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.5 |
| $\beta$ | 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:

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

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±0.98 | 62.32±4.13 |
| ER + Ours | 67.24±2.32 | 65.80±2.16 |
| OCM | 72.47±1.04 | 73.15±1.05 |
| OCM + Ours | 76.36±0.66 | 75.31±1.10 |
| OCM-CCLDC | 74.80±1.72 | 77.66±1.46 |
| OCM-CCLDC + Ours | 79.37±0.89 | 78.21±1.03 |

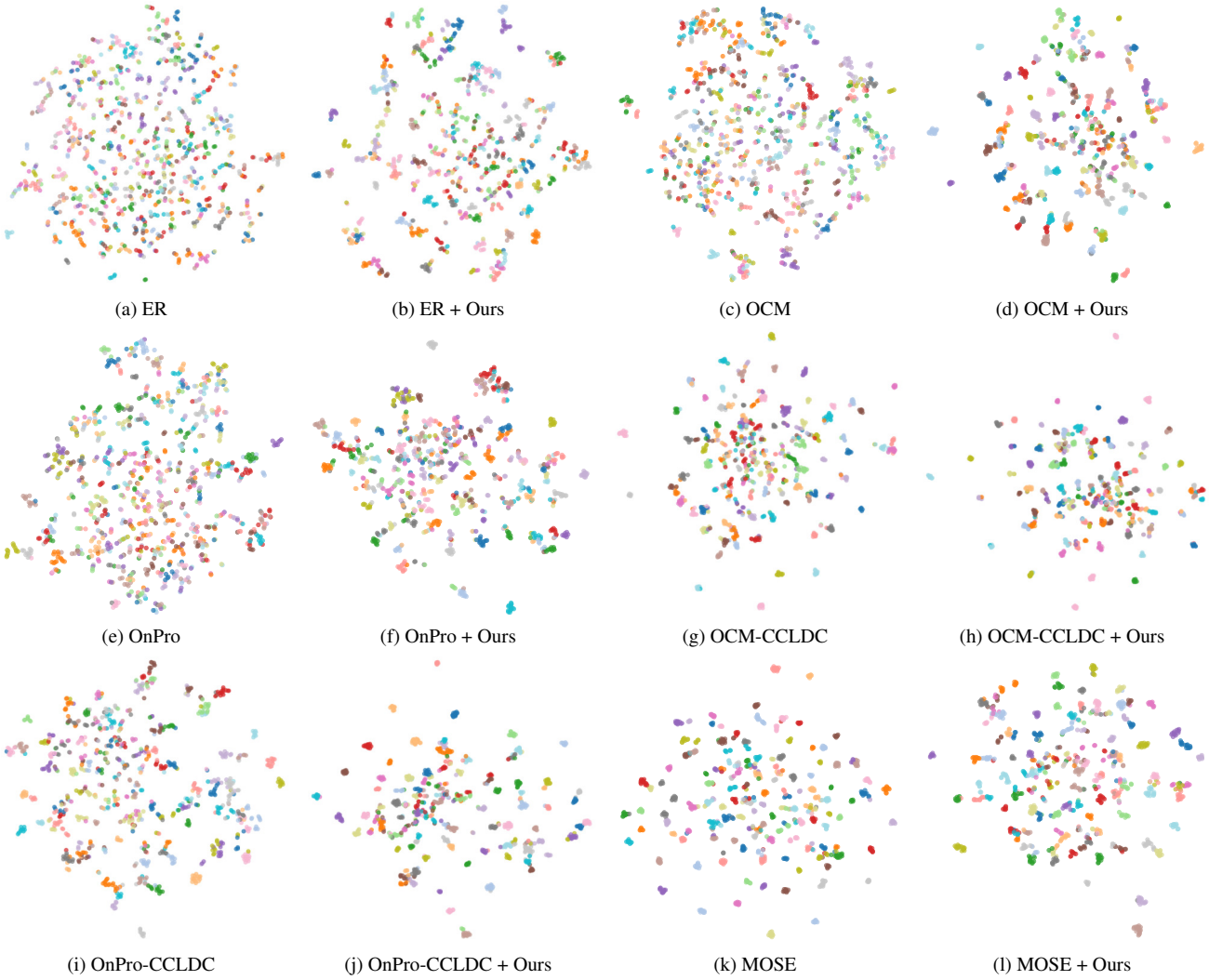
Table 6. Final average accuracy on CIFAR-10 ($M = 1k$), with and without our method on NCM and Logit predictions.

| Method | NCM Acc. ↑ | Logit Acc. ↑ |
|---|---|---|
| ER | 36.40±0.81 | 31.89±1.45 |
| ER + Ours | 36.99±0.65 | 34.55±1.66 |
| OCM | 37.76±0.70 | 35.69±1.36 |
| OCM + Ours | 39.98±1.19 | 38.97±2.28 |
| OCM-CCLDC | 40.28±1.08 | 43.34±1.51 |
| OCM-CCLDC + Ours | 44.55±1.42 | 44.40±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 ($M = 2k$).