Supplementary Material for Learning and Transforming General Representations to Break Down Stability-Plasticity Dilemma

Kengo Murata, Seiya Ito, and Kouzou Ohara

Aoyama Gakuin University, Kanagawa, Japan

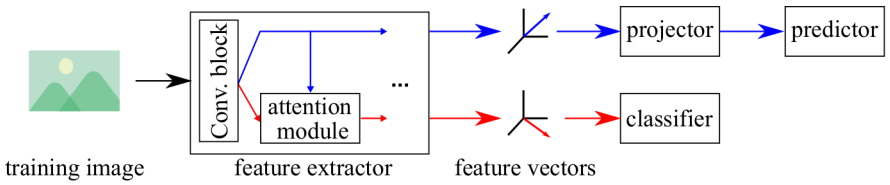

Fig. 1. The overview of the network structure of our framework.

A Details of Our Framework

First, we describe the overall network structure of our framework. As shown in Fig. 1, our network is composed of a feature extractor, projector, predictor, and classifier. The feature extractor outputs two kinds of feature vectors from an image: one is fed into the projector and predictor to calculate the self-supervised loss as shown in Fig. 1 with blue arrows, and the other one propagates along the classifier to calculate the other loss terms as shown in Fig. 1 with red arrows. The feature extractor is composed of convolutional blocks and attention modules. We modify batch normalization layers in convolutional blocks to deal with different batch statistics. More specifically, our batch normalization layers separately memorize the statistics of three types of intermediate feature vectors: vectors through convolutional blocks, vectors to which attention mechanism applies, and vectors derived from an image used for self-supervised learning.

Next, we summarize the learning process of our framework in Algorithm 1. The overall process is roughly divided into five steps. First, images are augmented through complex data augmentation (line 11). Second, the self-supervised loss is calculated from the augmented images through the feature extractor, projector, and predictor (lines 13-17). Third, the loss term for classification is calculated from the images to which simple data augmentation was applied (lines 19-26). Forth, the other losses are calculated if the model has already learned at least one task until then (lines 29-33). Finally, we backpropagate the total loss to all

Algorithm 1 Pseudocode of the learning process in our framework, PyTorch-like

```
# f: the current feature extractor
# f_pre: the previous feature extractor
# g: classifier
# proj: projector
# pred: predictor
# t: task index
# alpha: hyperparameter
# beta_cls_t, beta_kdl_t: scalar values

for x, y in loader: # load images x and labels y
    x1, x2 = aug_c(x), aug_c(x) # complex data augmentation
    # get feature vectors through the feature extractor without attention
    f1, f2 = f(x1, attn=False), f(x2, attn=False)
    p1, p2 = proj(f1), proj(f2) # get projected vectors
    q1, q2 = pred(p1), pred(p2) # get predicted vectors
    # calculate the self-supervised loss
    L = (1 - alpha) * L_ssl(p1, p2, q1, q2)

    x_cls = aug_s(x) # simple data augmentation
    # get feature vectors and source vectors through the feature extractor
    # with attention
    f_cls, zs = f(x_cls, attn=True)
    # get predicted class distributions
    y_hat = g(f_cls)
    # calculate the loss term for classification
    L += alpha * L_cls(y_hat, y)

    # case when training for the incremental tasks
    if t > 1:
        # get the previous feature vectors and source vectors
        f_cls_pre, zs_pre = f_pre(x_cls, attn=True)
        # calculate the other loss terms and derive the total loss
        L = beta_cls_t * L + beta_kdl_t * (L_kdl + L_attn(zs, zs_pre)) +
             L_other

    L.backward() # back-propagate
    for block in f.attn_modules(): # get attention modules
        for param in block.parameters():
            param.grad *= 1.0 / alpha
    for param in g.parameters():
        param.grad *= 1.0 / alpha
    update(f, h, proj, pred) # SGD update
```

the parameters, and update them after multiplying the propagated gradient of the parameters in our attention modules and the classifier (lines 35-41). The multiplication is intended to deal with the decline of the gradient norm caused by multiplication between cross-entropy loss and α. α is a hyperparameter that controls the balance between the self-supervised loss and cross-entropy loss, and used at lines 17 and 26 in Algorithm 1.

B Details of Baselines

In this section, we first describe the details of the learning procedures of baseline methods, including Co2L [2] that is the contrastive learning method we adopted. Then, we explain the motivation to choose the baseline methods.

B.1 Learning Procedures

Essentially, the loss functions of the baselines for learning the t-th task are summarized as follows:

$$\mathcal{L}_{\text{kd}} = \beta_t^{\text{cls}} \mathcal{L}_{\text{cls}} + \beta_t^{\text{kdl}} \mathcal{L}_{\text{kdl}} + \mathcal{L}_{\text{other}} \ , \tag{1}$$

where $\mathcal{L}_{\text{cls}}$, $\mathcal{L}_{\text{kdl}}$, and $\mathcal{L}_{\text{other}}$ denote the loss terms for classification, knowledge distillation, and the other purposes, respectively. In addition, β_t^{cls} and β_t^{kdl} are the scalar values that control the impact of each loss term depending on the number of classes the model learned. The methods employing knowledge distillation involve two models. We refer to the model that is being updated for the current t-th task as the current model, while the one that has already been updated for the $(t-1)$-th task as the previous model. Each model has a feature extractor and a classifier, which we call the current feature extractor f_t, the current classifier g_t, the previous feature extractor f_{t-1}, and the previous classifier g_{t-1}, respectively.

IL-Baseline. IL-Baseline [12] implements the loss term $\mathcal{L}_{\text{cls}}$ as a cross-entropy loss and the loss term $\mathcal{L}_{\text{kdl}}$ as the original knowledge distillation loss [9]. Formally, for a training image x, its knowledge distillation loss is defined as:

$$\mathcal{L}_{\text{kdl}}^{\text{base}} = \mathcal{L}_{\text{CE}} \left(\sigma \left(\frac{(g_t \circ f_t)(x)}{T} \right), \sigma \left(\frac{(g_{t-1} \circ f_{t-1})(x)}{T} \right) \right) \ , \tag{2}$$

where $\mathcal{L}_{\text{CE}}$ and σ denote the cross-entropy loss and the softmax function, respectively. In addition, T is a hyperparameter called temperature, which we set to 2 in our experiments. To balance the impact of the cross-entropy loss and the knowledge distillation loss, IL-Baseline sets β_t^{cls} and β_t^{kdl} as follows:

$$\beta_t^{\text{cls}} = 1 - \frac{\sum_{i=1}^{t-1} |\mathcal{Y}_i|}{\sum_{i=1}^{t} |\mathcal{Y}_i|} \ , \tag{3}$$

$$\beta_t^{\text{kdl}} = \frac{\sum_{i=1}^{t-1} |\mathcal{Y}_i|}{\sum_{i=1}^{t} |\mathcal{Y}_i|} \ , \tag{4}$$

where $\mathcal{Y}_i$ is the set of labels corresponding to the i-th task.

UCIR. UCIR [10] incorporates three components: cosine normalization, less-forget constraint, and inter-class separation. First, cosine normalization works on the calculation of a classifier, which includes the dot product between the output vector of a feature extractor and the weight vectors on the classifier. More precisely, cosine normalization adds the l_2 normalization of the feature vector and the weight vectors before calculating the dot product to deal with the biased prediction problem mentioned in our main paper. UCIR uses the output of the classifier to calculate the loss term $\mathcal{L}_{\text{cls}}$, which is implemented as the

cross-entropy loss. Second, the less-forget constraint is realized by introducing the following loss term $\mathcal{L}_{\text{kdl}}^{\text{ucir}}$ into the loss function as the implementation of $\mathcal{L}_{\text{kdl}}$:

$$\mathcal{L}_{\text{kdl}}^{\text{ucir}} = \lambda_{\text{kdl}}^{\text{flat}} \cdot \mathcal{D}_{\cos}(f_t(x), f_{t-1}(x)), \tag{5}$$

where $\mathcal{D}_{\cos}$ denotes a cosine distance between its arguments, and $\lambda_{\text{kdl}}^{\text{flat}}$ is the hyperparameter that controls the impact of this loss term. In addition, UCIR sets β_t^{kdl} as $\sqrt{(\sum_{i=1}^{t} |\mathcal{Y}_i|)/|\mathcal{Y}_t|}$, while setting β_t^{cls} to 1. Third, inter-class separation is composed of the margin ranking loss that encourages the current model to learn well-separated features useful for distinguishing the classes in the past tasks from the ones in the current task. Formally, for any exemplar instance $(x_{\text{mem}}, y_{\text{mem}})$, the margin ranking loss $\mathcal{L}_{\text{other}}^{\text{ucir}}$ is computed as:

$$\mathcal{L}_{\text{other}}^{\text{ucir}} = \lambda_{\text{other}}^{\text{ucir}} \sum_{k=1}^{K} \max(m + \mathcal{D}_{\cos}(\theta^{y_{\text{mem}}}, f_t(x_{\text{mem}})) - \mathcal{D}_{\cos}(\theta^k, f_t(x_{\text{mem}})), 0), \tag{6}$$

where m is the margin threshold, and $\lambda_{\text{other}}^{\text{ucir}}$ is the hyperparameter controlling the impact of this loss term. θ^k is a weight vector for the classes $\mathcal{Y}_t$ in the classifier that is the k-th similar to $f_t(x)$ in terms of cosine similarity. We only consider the top-K weight vectors in that similarity ranking and set the value of K to 2 in our experiments following the original paper [10].

PODNet. PODNet [6] introduces a novel classifier called a local similarity classifier and some constraints on the intermediate feature vectors. In the local similarity classifier, there are V weight vectors for each class, and the weighted average cosine similarity between each weight vector and the output vector of the feature extractor is the final output of the classifier. Formally, the calculation of the current local similarity classifier for a class y' is defined as:

$$g_t(z_t)^{y'} = \sum_{v=1}^{V} s_t^{y',v} (\overline{\theta_t^{y',v}} \cdot \overline{z_t}), \tag{7}$$

$$s_t^{y',v} = \frac{\exp(\overline{\theta_t^{y',v}} \cdot \overline{z_t})}{\sum_{i=1}^{V} \exp(\overline{\theta_t^{y',i}} \cdot \overline{z_t})}\ , \tag{8}$$

where $\theta_t^{y',v}$ is the v-th weight vector for class y', and z_t denotes the output vector of the current feature extractor f_t from an image x. We set the number of weight vectors V to 10 in our experiments following the original paper [6]. With the calculated scores $g_t(z_t)^{y'}$ of a training instance (x, y), the loss term $\mathcal{L}_{\text{cls}}$ implemented based on the NCA loss [14] is calculated as follows:

$$\mathcal{L}_{\text{cls}}^{\text{pod}} = \left[-\log \frac{\exp(\eta(g_t(z_t)^y - \delta))}{\sum_{i \neq y} \exp(\eta g_t(z_t)^i)} \right]_+, \tag{9}$$

where $[\cdot]_+$ denotes a hinge function, δ is the hyperparameter called a margin, and η is the additional learnable parameter. We set the value of δ to 0.6 in our experiments following the original paper [6]. PODNet realizes the constraints on the intermediate feature vectors by implementing the loss term $\mathcal{L}_{\text{kdl}}$ as follows:

$$\mathcal{L}_{\text{kdl}}^{\text{pod}} = \mathcal{L}_{\text{kdl}}^{\text{ucir}} + \lambda_{\text{kdl}}^{\text{spatial}} \mathcal{L}_{\text{kdl}}^{\text{spatial}} \ , \tag{10}$$

$$\mathcal{L}_{\text{kdl}}^{\text{spatial}} = \frac{1}{S} \sum_{s=1}^{S} \mathcal{L}_{\text{kdl}}^{\text{width}}(h_t^s, h_{t-1}^s) + \mathcal{L}_{\text{kdl}}^{\text{height}}(h_t^s, h_{t-1}^s) \ , \tag{11}$$

$$\mathcal{L}_{\text{kdl}}^{\text{width}}(h_t^s, h_{t-1}^s) = \sum_{c=1}^{C} \sum_{h=1}^{H} \mathcal{D}_{\cos}(\sum_{w=1}^{W} h_t^{s,c,w,h}, \sum_{w=1}^{W} h_{t-1}^{s,c,w,h}) \ , \tag{12}$$

$$\mathcal{L}_{\text{kdl}}^{\text{height}}(h_t^s, h_{t-1}^s) = \sum_{c=1}^{C} \sum_{w=1}^{W} \mathcal{D}_{\cos}(\sum_{h=1}^{H} h_t^{s,c,w,h}, \sum_{h=1}^{H} h_{t-1}^{s,c,w,h}) \ , \tag{13}$$

where $\lambda_{\text{kdl}}^{\text{spatial}}$ is the hyperparameter that controls the impact of $\mathcal{L}_{\text{kdl}}^{\text{spatial}}$, S is the total number of the intermediate feature vectors on which constraints are imposed, and h_t^s denotes the s-th constrained intermediate feature vectors derived from the current feature extractor f_t. In addition, $h_t^{s,c,w,h}$ denotes the element of h_t^s, where c stands for the channel and $w \times h$ for the column and row of the spatial coordinates. C, W, and H denote the number of dimensions in each coordinate.

BSCE. BSCE [12] has the same learning procedure as IL-Baseline [12] except for implementing $\mathcal{L}_{\text{cls}}$ as the balanced softmax cross-entropy loss. Formally, for a training sample (x, y), the balanced softmax cross-entropy loss is defined as:

$$\mathcal{L}_{\text{cls}}^{\text{bsce}} = -\log \frac{N_y \exp((g_t \circ f_t)(x)^y)}{\sum_{j \in \bigcup_{k=1}^{t} \mathcal{Y}_k} N_j \exp((g_t \circ f_t)(x)^j)} \ , \tag{14}$$

where N_y denotes the number of samples belonging to the class y in the training set, including memorized exemplars, and $(g_t \circ f_t)(x)^y$ is the output score for the class y.

Co2L. Co2L [2] employs the asymmetric supervised contrastive loss as $\mathcal{L}_{\text{cls}}$ and instance-wise relation distillation as $\mathcal{L}_{\text{kdl}}$. Let the incoming batch of B samples be $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{B}$ and the projected vectors of an image x_i be $z_{t,i}^{\text{proj}} = (g_t^{\text{proj}} \circ f_t)(x_i)$, where g_t^{proj} is the current projector. In addition, we let $S \subset \{1, \ldots, B\}$ be the set of indices of the t-th task samples in the batch $\mathcal{B}$. Then, for the incoming batch $\mathcal{B}$, the asymmetric supervised contrastive loss is defined as:

$$\mathcal{L}_{\text{cls}}^{\text{co2l}} = -\sum_{i \in S} \frac{1}{|\mathfrak{p}_i|} \sum_{j \in \mathfrak{p}_i} \log \frac{\exp\left(\overline{z_{t,i}^{\text{proj}}} \cdot \overline{z_{t,j}^{\text{proj}}} / \tau\right)}{\sum_{k \neq i} \exp\left(\overline{z_{t,i}^{\text{proj}}} \cdot \overline{z_{t,k}^{\text{proj}}} / \tau\right)}, \tag{15}$$

where $\mathfrak{p}_i$ is the index set of positive samples with respect to a sample (x_i, y_i), defined as:

$$\mathfrak{p}_i = \{j \in \{1, \ldots . B\} |\ j \neq i, y_j = y_i\}. \tag{16}$$

Instance-wise relation distillation is implemented as the dissimilarity between instance-wise similarity vectors derived through the current and previous models. Formally, for a scalar value κ', the instance-wise similarity vector $d(z_{t,i}^{\mathrm{proj}}; \kappa')$ is defined as:

$$d(z_{t,i}^{\mathrm{proj}}; \kappa') = [d_{t,i,1}, \cdots, d_{t,i,i-1}, d_{t,i,i+1}, \cdots, d_{t,i,B}] \ , \tag{17}$$

$$d_{t,i,j} = \frac{\exp(\overline{z_{t,i}^{\mathrm{proj}}} \cdot \overline{z_{t,j}^{\mathrm{proj}}} / \kappa')}{\sum_{k \neq i} \exp(\overline{z_{t,i}^{\mathrm{proj}}} \cdot \overline{z_{t,k}^{\mathrm{proj}}} / \kappa')} \ . \tag{18}$$

$$\tag{19}$$

Then, the instance-wise relation distillation loss is defined as:

$$\mathcal{L}_{\mathrm{kdl}}^{\mathrm{co2l}} = -\sum_{i=1}^{B} d(z_{t-1,i}^{\mathrm{proj}}; \kappa^*) \log d(z_{t,i}^{\mathrm{proj}}; \kappa) \ , \tag{20}$$

where κ^* and κ are hyperparameters.

B.2 Motivation

We chose IL-Baseline, UCIR, PODNet, and BSCE as baselines so that we can involve various knowledge distillation loss and bias correction methods in order to evaluate the flexibility of our framework. In fact, they involve three types of knowledge distillation loss, i.e., the constraints on the output probabilities (IL-Baseline and BSCE), the output vectors of the feature extractor (UCIR and PODNet), and the intermediate feature vectors (PODNet). Also, they employ three types of bias correction methods, i.e., no bias correction (IL-Baseline), cosine normalization (UCIR and PODNet), and the imbalance-aware loss function (BSCE). In addition, we included Co2L as the baseline in our experiment to verify whether our framework positively affects the contrastive learning methods.

C Details of Experimental Settings

C.1 Dataset

We mainly employed two image datasets for our experiments: CIFAR100 [13] and ImageNet100 [5, 10]. CIFAR100 contains 60,000 images of size 32×32 from 100 classes, and each class includes 500 training samples and 100 test samples. ImageNet100 is a subset of ImageNet1000 [5] with 100 classes randomly selected from it. ImageNet100 was constructed through Continuum [7], a python library for continual learning.

Table 1. The configuration of our feature extractor for each dataset (left: CIFAR100, right: ImageNet100). The settings of a residual block are shown in brackets with the number of blocks stacked. The setting of a convolutional layer is written as "conv, kernel size, the number of channel". Similary, "max pool, $x \times y$" means the max pooling operation with kernel size of $x \times y$. The inner brackets following fc indicate the output dimensions of the two fully connected layers in our attention module.

output size	layers
32×32	conv, 3×3, 64
32×32	$\begin{bmatrix} \text{conv}, 3 \times 3, 64 \\ \text{conv}, 3 \times 3, 64 \\ fc, [4, 64] \end{bmatrix} \times 2$
16×16	$\begin{bmatrix} \text{conv}, 3 \times 3, 128 \\ \text{conv}, 3 \times 3, 128 \\ fc, [8, 128] \end{bmatrix} \times 2$
8×8	$\begin{bmatrix} \text{conv}, 3 \times 3, 256 \\ \text{conv}, 3 \times 3, 256 \\ fc, [16, 256] \end{bmatrix} \times 2$
4×4	$\begin{bmatrix} \text{conv}, 3 \times 3, 512 \\ \text{conv}, 3 \times 3, 512 \\ fc, [32, 512] \end{bmatrix} \times 2$

Output size	Layers
112×112	conv, 7×7, 64
	max pool, 3×3
56×56	$\begin{bmatrix} \text{conv}, 3 \times 3, 64 \\ \text{conv}, 3 \times 3, 64 \\ fc, [4, 64] \end{bmatrix} \times 2$
28×28	$\begin{bmatrix} \text{conv}, 3 \times 3, 128 \\ \text{conv}, 3 \times 3, 128 \\ fc, [8, 128] \end{bmatrix} \times 2$
14×14	$\begin{bmatrix} \text{conv}, 3 \times 3, 256 \\ \text{conv}, 3 \times 3, 256 \\ fc, [16, 256] \end{bmatrix} \times 2$
7×7	$\begin{bmatrix} \text{conv}, 3 \times 3, 512 \\ \text{conv}, 3 \times 3, 512 \\ fc, [32, 512] \end{bmatrix} \times 2$

C.2 Data Augmentation

In our framework, two kinds of data augmentation are utilized: one for the calculation of the self-supervised loss and the other for the calculation of the other loss functions. We refer to the former as the complex data augmentation and the latter as the simple data augmentation, respectively. We utilized the data augmentation techniques used in SimSiam [4] as the complex data augmentation and the ones used in PODNet [6] as the simple data augmentation. In the following, we describe the details of the data augmentation for each dataset using the PyTorch [15] notations.

CIFAR100. The complex data augmentation was composed of the following data augmentation techniques:

- RandomResizedCrop with scale in $[0.2, 1.0]$;
- RandomHorizontalFlip with a probability of 0.5;
- ColorJitter that changes brightness, contrast, saturation, hue strength of $\{0.4, 0.4, 0.4, 0.1\}$ with an applying probability of 0.8; and
- RandomGrayScale with an applying probability of 0.2.

As for the simple data augmentation, we applied RandomCrop with the padding size of 4 and RandomHorizontalFlip with a probability of 0.5.

Table 2. The parameter spaces for hyperparameter tuning. The first column indicates the method involving the corresponding hyperparameters. "Common" is the method name assigned to hyperparameters that are involved in every method for the sake of expediency.

method	hyperparameter	parameter space
UCIR	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
	$\lambda_{\mathrm{other}}^{\mathrm{ucir}}$	$\{0.5 \cdot i \mid i \in [1, 5]\}$
	m	$\{0.1, 0.3, 0.5, 0.7\}$
PODNet	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
	$\lambda_{\mathrm{kdl}}^{\mathrm{spatial}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
Our framework	α	$\{0.3, 0.5, 0.7\}$
	$\lambda_{\mathrm{attn}}^{\mathrm{map,new}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
	$\lambda_{\mathrm{attn}}^{\mathrm{map,old}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
	$\lambda_{\mathrm{attn}}^{\mathrm{source}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$
Common	lr^{cont}	{2e-4, 1e-3, 2e-3, 1e-2, 2e-2}

ImageNet100. As for the complex data augmentation, GaussianBlur with a standard deviation in $[0.1, 0.2]$ was applied in conjunction with the ones applied to CIFAR100. The simple data augmentation was composed of the following data augmentation techniques:

- RandomResizedCrop with an output size of 224;
- RandomHorizontalFlip with a probability of 0.5; and
- ColorJitter that only changes a brightness strength of 63/255.

C.3 Network Structure

We used ResNet18 [8] as the Convolutional Neural Network (CNN) backbone of the feature extractor. Our attention modules were attached to residual blocks in a similar manner to SE-ResNet18 [11], and composed of a fully connected layer with D units following a fully connected layer with D/r units where D and r denote the output channels and a reduction ratio, respectively. The reduction ratio was set to 16. Table 1 shows the configuration of our feature extractor for each dataset. We used a slightly different network structure from the settings shown in Table 1 for PODNet [6] when applying it to ImageNet100 according to its official implementation[1]. More specifically, the kernel size of the first convolutional layer was set to 3×3. As for a projector and predictor, we used Multi-Layer Perceptrons (MLPs) with one hidden layer. The projector was composed of two fully connected layers with 2048 units. On the other hand, the predictor was formed as a bottleneck structure by setting the hidden layer dimension to 512 while the output layer dimension to 2048.

[1] https://github.com/arthurdouillard/incremental_learning.pytorch

Table 3. The values of hyperparameters for each dataset.

method	hyperparameter	value (CIFAR100)	value (ImageNet100)
IL-Baseline *w/* Ours	lr^{cont}	2e-2	2e-3
	α	0.7	0.3
	$\lambda_{\mathrm{attn}}^{\mathrm{map,new}}$	1.5	7.0
	$\lambda_{\mathrm{attn}}^{\mathrm{map,old}}$	4.5	6.0
	$\lambda_{\mathrm{attn}}^{\mathrm{source}}$	4.5	7.5
UCIR	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	7.5	-
	$\lambda_{\mathrm{other}}^{\mathrm{ucir}}$	2.0	-
	m	0.7	-
UCIR *w/* Ours	lr^{cont}	2e-2	2e-3
	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	2.5	3.5
	$\lambda_{\mathrm{other}}^{\mathrm{ucir}}$	1.0	1.5
	m	0.7	0.5
	α	0.7	0.3
	$\lambda_{\mathrm{attn}}^{\mathrm{map,new}}$	2.0	2.0
	$\lambda_{\mathrm{attn}}^{\mathrm{map,old}}$	3.5	0.5
	$\lambda_{\mathrm{attn}}^{\mathrm{source}}$	4.0	2.0
PODNet	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	2.5	-
	$\lambda_{\mathrm{kdl}}^{\mathrm{spatial}}$	2.0	-
PODNet *w/* Ours	lr^{cont}	2e-2	1e-3
	$\lambda_{\mathrm{kdl}}^{\mathrm{flat}}$	1.0	3.0
	$\lambda_{\mathrm{kdl}}^{\mathrm{spatial}}$	2.0	4.5
	α	0.7	0.3
	$\lambda_{\mathrm{attn}}^{\mathrm{map,new}}$	9.0	4.5
	$\lambda_{\mathrm{attn}}^{\mathrm{map,old}}$	4.0	9.5
	$\lambda_{\mathrm{attn}}^{\mathrm{source}}$	7.5	10.0
BSCE *w/* Ours	lr^{cont}	2e-2	2e-2
	α	0.7	0.7
	$\lambda_{\mathrm{attn}}^{\mathrm{map,new}}$	2.5	3.0
	$\lambda_{\mathrm{attn}}^{\mathrm{map,old}}$	1.5	7.5
	$\lambda_{\mathrm{attn}}^{\mathrm{source}}$	4.0	0.5

C.4 Hyperparameter Tuning

We tuned hyperparameters through a pseudo-task sequence based on the existing hyperparameter tuning process [3] for the class incremental learning scenario. Specifically, we constructed the pseudo-task sequence by dividing the classes belonging to the first task into three sub-tasks consisting of 30, 10, and 10 classes, respectively. Through this pseudo-task sequence, we performed a hold-out validation for hyperparameters of each method. The learning rate lr^{cont} was tuned only for the tasks excluding the first one. We searched optimal hyperparameters from the parameter spaces shown in Table 2 using Optuna [1]. Note that we excluded the learning rate lr^{cont} from the search targets in the case of CIFAR100 to shrink the search space and set it to 2e-2. Table 3 shows the resulting hyper-

parameters. IL-Baseline and BSCE are not listed in Table 3 because they do not have any hyperparameters to tune.

C.5 Evaluation Metrics

We mainly evaluated models through average incremental accuracy (AIA) [16] for an evaluation set, which is defined as:

$$\mathrm{AIA} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sum_{k=1}^{i}|\mathcal{D}_k^{\mathrm{ev}}|}\sum_{j=1}^{i}R_{i,j}, \tag{21}$$

where N denotes the length of a task sequence, $\mathcal{D}_k^{\mathrm{ev}}$ denotes the evaluation set of the k-th task, and $R_{i,j}$ denotes the number of samples in $\mathcal{D}_j^{\mathrm{ev}}$ whose class labels were correctly predicted by the model that has been optimized for the i-th task. In addition, we used the average accuracy for the first task $\mathrm{AA}^{\mathrm{old}}$ and the average accuracy for the new tasks $\mathrm{AA}^{\mathrm{new}}$ as the additional evaluation metrics. Formally, they are defined as:

$$\mathrm{AA}^{\mathrm{old}} = \frac{1}{N-1}\sum_{i=2}^{N}\frac{R_{i,1}}{|\mathcal{D}_1^{\mathrm{ev}}|}\ , \tag{22}$$

$$\mathrm{AA}^{\mathrm{new}} = \frac{1}{N-1}\sum_{i=2}^{N}\frac{R_{i,i}}{|\mathcal{D}_i^{\mathrm{ev}}|}\ . \tag{23}$$

In addition, the last accuracy LA is defined as:

$$\mathrm{LA} = \frac{1}{\sum_{k=1}^{N}|\mathcal{D}_k^{\mathrm{ev}}|}\sum_{j=1}^{N}R_{N,j}. \tag{24}$$

C.6 Configurations of Comparison Experiment with Co2L

To verify the effect of our framework on contrastive learning methods, we reported the comparison experiment between two kinds of Co2L [2] with and without our framework on CIFAR10 with 200 exemplars in our main paper. In that experiment, we adopted exactly the same experimental configurations as the ones of the original paper [2]. More specifically, we constructed a task sequence by equally dividing the whole class set into five, meaning each task contained two classes. As for the network structure, we used ResNet18 and 2-layer MLP as the feature extractor and projector for supervised contrastive learning. The projector has 512 and 128 units in its hidden and output layers, respectively. We note that this projector is entirely different from a projector used for self-supervised learning. All parameters were optimized through stochastic gradient descent (SGD) with a momentum of 0.9. For the first task, the network was trained with 500 epochs using a cosine learning rate decay for the base learning rate of 0.5 and weight decay of 1e-4. Then, it was further optimized using the

Table 4. The search spaces for hyperparameters of Co2L with our framework and the values resulting from the search.

hyperparameter	parameter space	value
τ	$\{0.1, 0.5, 1.0\}$	0.5
κ	$\{0.1, 0.2\}$	0.2
κ^*	$\{0.01, 0.05, 0.1\}$	0.1
α	$\{0.3, 0.5, 0.7\}$	0.5
$\lambda_{\text{attn}}^{\text{map,new}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$	1.0
$\lambda_{\text{attn}}^{\text{map,old}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$	3.5
$\lambda_{\text{attn}}^{\text{source}}$	$\{0.5 \cdot i \mid i \in [1, 20]\}$	0.5

remaining tasks one by one with 100 epochs per task. Note that a classifier needs to be optimized independently because the supervised contrastive learning does not contain the optimization of the classifier. Thus, we trained the classifier after the optimization of other network components with a class-balanced sampling strategy. It was optimized with 100 epochs using the exponential learning rate decay for the base learning rate of 1.0 and no weight decay. Training images were augmented through the complex data augmentation, which we used on the CIFAR100 experiments, and passed to the network with a batch size of 512. The hyperparameters were selected through the training on the entire task sequence using Optuna [1] based on the last accuracy for the validation set consisting of 10% of the training samples randomly drawn. It is noted that although this tuning procedure is not realistic due to using all the training samples, we dared to use it because it was adopted in the original paper [2]. Table 4 shows the search spaces for the hyperparameters of Co2L with our framework and the values resulting from the search. We configured the hyperparameters of Co2L without our framework following the paper [2]. Namely, τ, κ, and κ^* were set to 0.5, 0.2, and 0.01, respectively.

D Supplementary Experiments

D.1 The Effect of Attention and Data Augmentation

Our framework introduces attention techniques and complex data augmentation, neither of which is employed by the baseline methods. Since they might affect the resulting values of AIA, we compared the baseline methods with and without attention techniques and additional data augmentation on CIFAR100. To this end, we used SE-ResNet18 [11] as the feature extractor employing attention techniques, which contains the same number of parameters as our feature extractor. Table 5 shows the comparison results, where the scores in the first row of the four rows corresponding to each baseline are the same as the baseline scores reported in our main paper. As shown in Table 5, there exist some cases in which introducing attention techniques and the complex data augmentation increases AIA. However, we emphasize that the improvements in AIA are significantly

Table 5. Average incremental accuracy of the baselines with and without attention techniques and additional data augmentation on CIFAR100. "SE-ResNet18 (modified)" is the same network structure as SE-ResNet18 except that the SE block is modified for our framework. The third column indicates whether the complex data augmentation described in section C.2 is applied. Each result is in the form of the average ± standard deviation obtained from three independent trials using different random seeds.

method	network	complex DA	5-phase	10-phase
IL-Baseline	ResNet18		52.92 ±0.15	43.14 ±0.31
	SE-ResNet18		51.74 ±0.17	42.78 ±0.13
	ResNet18	✓	51.02 ±0.07	42.42 ±0.31
	SE-ResNet18	✓	53.27 ±0.40	44.21 ±0.21
IL-Baseline *w/* Ours	SE-ResNet18 (modified)	✓	56.49 ±0.20	48.13 ±0.51
UCIR	ResNet18		69.29 ±0.15	63.16 ±0.12
	SE-ResNet18		69.15 ±0.17	62.26 ±0.14
	ResNet18	✓	66.31 ±0.32	63.01 ±0.24
	SE-ResNet18	✓	66.78 ±0.33	64.01 ±0.18
UCIR *w/* Ours	SE-ResNet18 (modified)	✓	71.03 ±0.28	66.24 ±0.42
PODNet	ResNet18		68.99 ±0.36	66.64 ±0.15
	SE-ResNet18		69.43 ±0.18	66.70 ±0.21
	ResNet18	✓	64.14 ±0.34	62.20 ±0.39
	SE-ResNet18	✓	64.62 ±0.08	62.72 ±0.09
PODNet *w/* Ours	SE-ResNet18 (modified)	✓	70.61 ±0.12	69.02 ±0.05
BSCE	ResNet18		71.64 ±0.16	64.87 ±0.39
	SE-ResNet18		71.36 ±0.12	65.01 ±0.25
	ResNet18	✓	70.64 ±0.30	65.54 ±0.15
	SE-ResNet18	✓	71.64 ±0.21	66.48 ±0.15
BSCE *w/* Ours	SE-ResNet18 (modified)	✓	74.09 ±0.24	70.30 ±0.38

smaller than those made by our framework. Actually, the maximum increase in AIA resulting from the introduction of attention techniques and complex data augmentation is only 1.61 points that is observed for BSCE in the 10-phase setup, whereas our framework increases the AIA of BSCE by 5.43 points in the same setup.

Next, we compared ResNet18 and SE-ResNet18 in average accuracy for the first task to confirm whether the original SE block decreases stability. As shown in Table 6, ResNet18 achieves higher average accuracies than SE-ResNet18 does except for the case of PODNet in the 5-phase setup. This decrease in accuracy implies that the introduction of the SE block could decline the stability of a model, while it could be avoided by sophisticated CIL methods such as PODNet.

D.2 Sensitivity Analysis

To verify the sensitivity of hyperparameters of our framework, we compared various hyperparameter settings on CIFAR100 with the 5-phase setup. Comparing the results in Fig. 2, we can confirm that the range of the change of AIA is

Table 6. The comparison between ResNet18 and SE-ResNet18 on CIFAR100 in average accuracy for the first task. Each result is in the form of the average ± standard deviation obtained from three independent trials using different random seeds.

method	network	5-phase	10-phase
IL-Baseline	ResNet18	38.09 ± 0.34	32.26 ± 0.40
	SE-ResNet18	36.04 ± 0.28	31.72 ± 0.09
UCIR	ResNet18	66.82 ± 0.23	60.26 ± 0.09
	SE-ResNet18	66.75 ± 0.33	58.88 ± 0.31
PODNet	ResNet18	68.38 ± 0.50	66.84 ± 0.15
	SE-ResNet18	69.06 ± 0.37	66.82 ± 0.32
BSCE	ResNet18	68.95 ± 0.49	62.29 ± 0.76
	SE-ResNet18	67.83 ± 0.26	61.81 ± 0.42

around 3 for α, while around 0.4 for the other parameters. Thus, we can say AIA is sensitive to the value of α but not to the values of the other parameters.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: ACM SIGKDD. pp. 2623–2631 (2019)
2. Cha, H., Lee, J., Shin, J.: Co2l: Contrastive continual learning. In: ICCV. pp. 9516–9525 (2021)
3. Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with A-GEM. In: ICLR (2019)
4. Chen, X., He, K.: Exploring simple siamese representation learning. In: CVPR. pp. 15750–15758 (2021)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR. pp. 248–255 (2009)
6. Douillard, A., Cord, M., Ollion, C., Robert, T., Valle, E.: Podnet: Pooled outputs distillation for small-tasks incremental learning. In: ECCV. pp. 86–102 (2020)
7. Douillard, A., Lesort, T.: Continuum: Simple management of complex continual learning scenarios. arXiv preprint arXiv:2102.06253 (2021)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
9. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
10. Hou, S., Pan, X., Loy, C.C., Wang, Z., Lin, D.: Learning a unified classifier incrementally via rebalancing. In: CVPR. pp. 831–839 (2019)
11. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: CVPR. pp. 7132–7141 (2018)
12. Jodelet, Q., Liu, X., Murata, T.: Balanced softmax cross-entropy for incremental learning. In: ICANN. vol. 12892, pp. 385–396 (2021)
13. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
14. Movshovitz-Attias, Y., Toshev, A., Leung, T.K., Ioffe, S., Singh, S.: No fuss distance metric learning using proxies. In: ICCV. pp. 360–368 (2017)

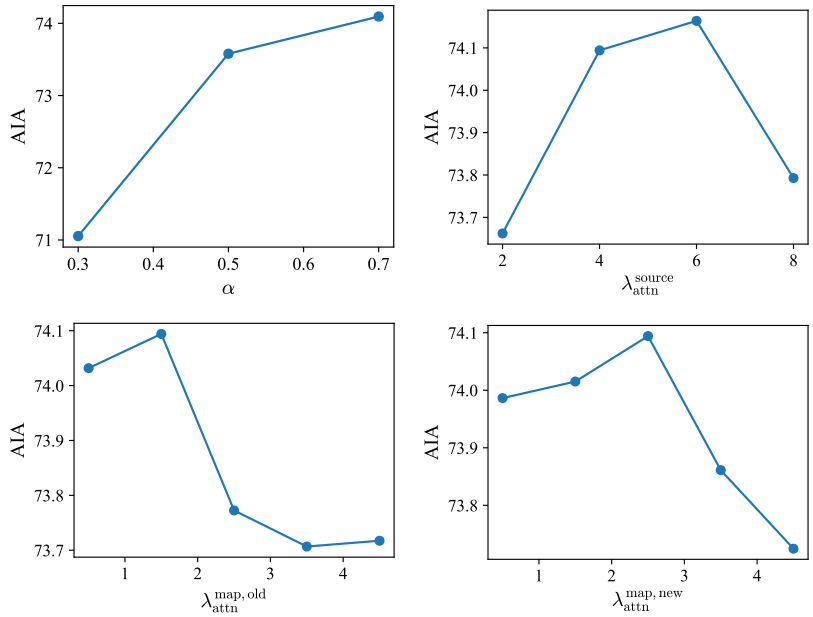

Fig. 2. The change of AIA of BSCE with our framework on CIFAR100 in the 5-phase setup when varying its hyperparameters α, $\lambda_{\text{attn}}^{\text{source}}$, $\lambda_{\text{attn}}^{\text{map,old}}$, and $\lambda_{\text{attn}}^{\text{map,new}}$. In each plot, the values of the other parameters than the one specified at the horizontal axis are the same as the ones in Table 3.

15. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS. vol. 32, pp. 8026–8037 (2019)
16. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: CVPR. pp. 2001–2010 (2017)