Self-Supervised Continual Learning with Codeword Merging through Clustering

Takanori Takebayashi
Computational Intelligence Laboratory
Department of Core Informatics
Graduate School of Informatics
Osaka Metropolitan University

1. Introduction

Recently, a large amount of unlabeled data can easily be obtained in many fields. Therefore, machine learning methods that continually extract feature information from unlabeled data, known as Unsupervised Continual Learning (UCL), are actively studied. UCL methods are widely used in many fields such as mobility [1] and medical applications [2] because of their high performance and practicality. Many UCL methods show equivalent or better classification performance than supervised learning methods, however, most UCL methods show lower performance during the first few tasks in task incremental learning. One of the main reasons is that most UCL methods cannot obtain diverse features from datasets, even if they use augmentation data.

Product Quantization (PQ) is a technique that enables diverse feature extraction. PQ is originally used for compressing high-dimensional feature vectors and enabling efficient nearest neighbor search [3]. PQ works by dividing feature vectors into several sub-vectors and assigning each sub-vector to the nearest codeword (i.e., centroids) in a codebook (i.e., a feature space). In UCL models, PQ enhances feature diversity by quantized vectors which contain compact feature information and capture key features of the learned data. As a result, the models can effectively learn distinguishing patterns.

Codebook for Unsupervised Continual Learning (CUCL) [4] can learn from diverse feature information by using self-supervised learning and PQ to solve the initial performance problem. As illustrated in Fig. 1, CUCL demonstrates superior performance compared to SimSiam, a widely used self-supervised learning model in UCL.

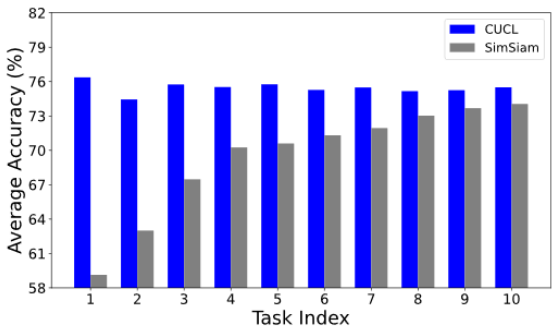


Fig. 1. Performance comparison between SimSiam and CUCL in task-incremental learning. CUCL maintains stable accuracy, while SimSiam struggles in the initial tasks.

CUCL maintains a separate codebook for each task, therefore in training steps, CUCL needs to know the current task index. We make these codebooks into a single unified codebook, and its codewords are added for each task. A unified codebook allows CUCL work without using the task index during training. However, as the number of tasks increases, the number of additional codewords in the shared codebook also increases.

In a shared codebook, many codewords are located close to each other, storing similar feature information. Therefore, merging these redundant codewords is one solution to reduce excessive codewords. We use CA+ [5] for this merging process.

CA+ is an Adaptive Resonance Theory (ART)-based clustering method. CA+ can dynamically group similar feature representations and adjust the codebook by merging redundant codewords. This process helps maintain a well-organized feature space, ensuring that important feature information is preserved while reducing unnecessary codewords. As a result, in a shared codebook, codewords are widely distributed and the number of codewords is effectively controlled.

CA+ is a variant of CAE [6] without topology. CAE is a parameter-free ART-based topological clustering algorithm capable of continual learning. Empirical studies with the synthetic and real-world datasets showed that the clustering performance of CAE is superior to state-of-the-art parameter-free/fixed algorithms. However, CAE tends to generate a large number of nodes. On the other hand, CA+ generates fewer nodes than CAE and CA+ can capture the entire distribution of data in [5].

The contributions of this paper are summarized as follows:

(i) We proposed a new Self-Supervised Learning (SSL) method that uses clustering to share the task feature information. By applying a shared codebook, our model enhances learning in continual learning settings

(ii) The original CUCL requires task indices during training. Our model removes this need by using a shared codebook, making it more flexible and easier to use in different settings.

(iii) When using a shared codebook, many similar codewords appear, which can be inefficient. We applied CA+, a clustering method based on Adaptive Resonance Theory (ART), to merge similar codewords and improve learning efficiency.

(iv) Our method is designed to work with different SSL models. It can be used in other SSL-based learning frameworks without major changes, making it adaptable to various tasks.

This paper is organized as follows. Section 2 presents preliminary knowledge for this paper. Section 3 presents the proposed CUCL with clustering. Section 4 presents numerical experiments to evaluate the effectiveness of sharing a codebook and merging redundant codewords. Section 5 concludes this paper.

2. Preliminary Knowledge

In this section, we first explain the continual learning scenario (i.e., incremental learning). Next, we describe two well-known self-supervised learning architectures. Then, we introduce the original CUCL method, and finally, we present the learning algorithm of CA+.

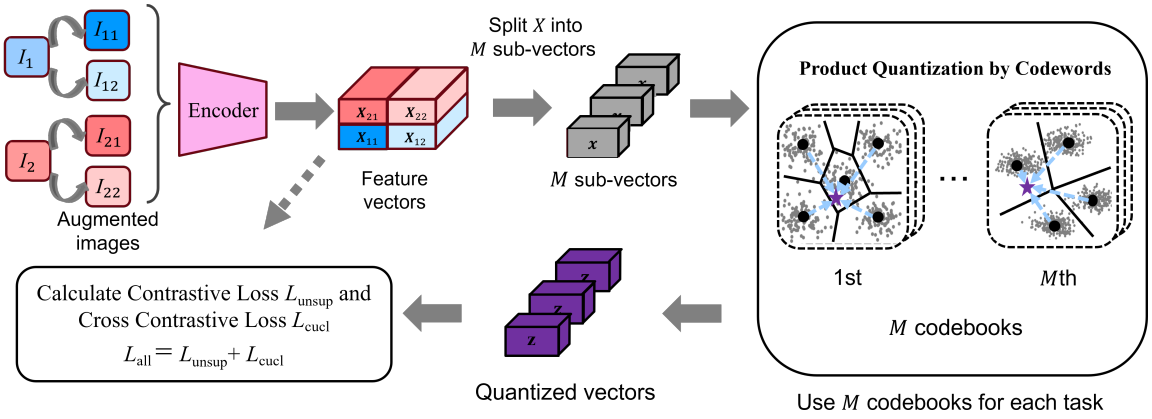


Fig. 2. Overview of the original CUCL method. Assume two datasets, I_1 and I_2, which receive data augmentation. Using an encoder, CUCL can extract four feature vectors from two augmented data. In the latent space, these feature vectors are divided into sub-vectors and each sub-vector is allocated to a respective codebook. Within each codebook, the sub-vectors are quantized into $\boldsymbol{z}$ by codewords. The model employs M codebooks per task, meaning that CUCL maintains M times the number of tasks in codebooks. Finally, the network is trained by computing the traditional unsupervised learning loss and the cross-contrastive loss.

2.1. Three types of Incremental Learning

In continual learning, data arrives sequentially instead of all at once as in traditional machine learning. This approach is known as incremental learning. There are three types: task-incremental, class-incremental, and domain-incremental learning [7].

- **Task-incremental learning**: Learning a series of discrete tasks sequentially.
- **Class-incremental learning**: Learning new classes gradually without access to previous class data.
- **Domain-incremental learning**: Adapting to scenarios where the data distribution changes between tasks. For example, a model might first learn from image data and later from voice data.

2.2. SimSiam

SimSiam [8] is a well-known self-supervised learning method that extracts features from data without labels. SimSiam consists of an encoder $f_\theta(\cdot)$ and a predictor $h_\theta(\cdot)$. The loss function $\mathcal{L}_{\text{SimSiam}}$ is defined using augmented data $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ as follows:

$$\mathcal{L}_{\text{SimSiam}} = \frac{1}{2}\left(D(\boldsymbol{p}_1, \boldsymbol{z}_2) + D(\boldsymbol{p}_2, \boldsymbol{z}_1)\right), \tag{1}$$

$$D(\boldsymbol{p}_i, \boldsymbol{z}_j) = -\frac{\boldsymbol{p}_i}{\|\boldsymbol{p}_i\|} \cdot \frac{\boldsymbol{z}_j}{\|\boldsymbol{z}_j\|}, \tag{2}$$

where $\boldsymbol{z}_i = f_\theta(\boldsymbol{x}_i)$ is the output of the encoder, and $\boldsymbol{p}_i = h_\theta(\boldsymbol{z}_i)$ is the predictor output ($i, j \in \{1, 2\}$). To prevent collapse, SimSiam stops the gradient update on one side of the encoder.

2.3. Barlow Twins

Barlow Twins [9] is a self-supervised learning method that reduces redundancy among latent representation dimensions while preserving their useful information. Given an input sample $\boldsymbol{x}_i$, two different augmentations are applied to obtain views $\boldsymbol{x}_i^1$ and $\boldsymbol{x}_i^2$. These are then mapped to latent representations $\boldsymbol{z}_i^1 = f_\theta(\boldsymbol{x}_i^1)$ and $\boldsymbol{z}_i^2 = f_\theta(\boldsymbol{x}_i^2)$ through a shared encoder f_θ. The loss function of Barlow Twins ensures that the representations remain similar while reducing correlations between different dimensions:

$$\mathcal{L}_{\text{Barlow Twins}} = \sum_{d=1}^{D}(1 - C_{d,d})^2 + \lambda \sum_{d=1}^{D}\sum_{d' \neq d}^{D} C_{d,d'}^2, \tag{3}$$

where the cross-correlation matrix element $C_{d,d'}$ is defined as:

$$C_{d,d'} = \frac{\sum_{b=1}^{B} \boldsymbol{z}_{b,d}^1 \boldsymbol{z}_{b,d'}^2}{\sqrt{\sum_{b=1}^{B}(\boldsymbol{z}_{b,d}^1)^2}\sqrt{\sum_{b=1}^{B}(\boldsymbol{z}_{b,d'}^2)^2}}. \tag{4}$$

Here, D represents the dimensionality of the latent representation, and λ is a hyperparameter that controls the balance between invariance and decorrelation. The cross-correlation matrix $C_{d,d'}$ measures the dependency between the d-th dimension of $\boldsymbol{z}^1$ and the d'-th dimension of $\boldsymbol{z}^2$. B is the mini-batch size, and $\boldsymbol{z}_{b,d}^1$ denotes the d-th component of the representation for sample b from the first augmented view, while $\boldsymbol{z}_{b,d'}^2$ corresponds to the d'-th component from the second augmented view.

2.4. CUCL

CUCL divides feature vectors $\boldsymbol{X}$ into $\boldsymbol{M}$ sub-vectors $(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_M)$ in the latent space. These sub-vectors are quantized $\boldsymbol{z}$ by codewords. The three main processes described are Quantization in Section 2.4.1, Loss Function in Section 2.4.2. Fig. 2 shows an overview of the original CUCL.

2.4.1. Quantization

Sub-vectors are quantized by codewords. Soft quantization [10] is used for the quantization process in CUCL. At first, for the quantization, CUCL calculates the distance between the sub-vectors $\boldsymbol{x}_i$ and the codeword $\boldsymbol{c}_{ik}$:

$$\text{dis}(\boldsymbol{x}_i, \boldsymbol{c}_{ik}) = \|\boldsymbol{x}_i - \boldsymbol{c}_{ik}\|_2^2, \tag{5}$$

where $\|\cdot\|_2^2$ represents the squared Euclidean distance. The soft quantization is processed as follows:

$$\boldsymbol{z}_i = \sum_{k=1}^{K} \frac{\exp(\text{-dis}(\boldsymbol{x}_i, \boldsymbol{c}_{ik})/\tau_q)}{\sum_{k'=1}^{K}\exp(\text{-dis}(\boldsymbol{x}_i, \boldsymbol{c}_{ik'})/\tau_q)}\boldsymbol{c}_{ik}, \tag{6}$$

where τ_q is a temperature parameter to scale the input data. CUCL can obtain diverse features through the quantization.

2.4.2. Loss Function

CUCL calculates a cross-contrastive loss between feature vectors and quantized vectors as well as the unsupervised contrastive

loss L_{unsup}. The cross-entropy is defined as follows:

$$L_{\text{cucl}} = -log\frac{\exp(\text{Cos}(\boldsymbol{X}_i, \boldsymbol{Z_j})/\tau_l)}{\sum_{n=1}^{N_B} \mathbf{1}_{[n\neq j]} \exp(\text{Cos}(\boldsymbol{X}_i, \boldsymbol{Z}_n)/\tau_l)}, \quad (7)$$

where $\text{Cos}(\cdot)$ represents cosine similarity, τ_l is a non-negative temperature parameter, and the $\mathbf{1}_{n\neq j} \in \{0, 1\}$ denotes the indicator that equates to 1 iff $n \neq j$.

Finally, in CUCL, the unsupervised contrastive loss L_{unsup} and the cross-entropy loss L_{cucl} are calculated for overall loss function L_{all} defined as:

$$L_{\text{all}} = L_{\text{unsup}} + L_{\text{cucl}}. \quad (8)$$

2.5. CIM-based ART Clustering

In this section, firstly, two major components used in CAE and CA+, a similarity measure and a kernel density estimator are explained. Next, the learning procedure of CAE and CA+ is explained. Table I shows the main notation in CAE and CA+.

2.5.1. Correntropy and Correntropy-induced Metric

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
$\mathbf{x}$	d-dimensional data point
$\mathcal{X}_c$	set of data points for a client c ($\mathbf{x} \in \mathcal{X}_c$)
$\mathbf{y}_k$	k-th node
$\mathcal{Y}$	set of nodes ($\mathbf{y}_k \in \mathcal{Y}$)
$\lvert\mathcal{Y}\rvert$	number of nodes in $\mathcal{Y}$
$\mathbf{y}_{s_1}$	1st winner node
$\mathbf{y}_{s_2}$	2nd winner node
$\hat{C}(\cdot)$	correntropy
$\text{CIM}(\cdot)$	correntropy-induced metric
σ	bandwidth of a kernel function in CIM
$\mathcal{S}$	set of bandwidths of a kernel function $\sigma \in \mathcal{S}$
$\mathcal{N}_k$	set of neighbor nodes of node $\mathbf{y}_k$
λ	number of active nodes
$\mathcal{A}$	set of active nodes
D	diversity of a set of active nodes
V_{s_1}	CIM value between $\mathbf{x}$ and $\mathbf{y}_{s_1}$
V_{s_2}	CIM value between $\mathbf{x}$ and $\mathbf{y}_{s_2}$
$\mathbf{R}$	matrix of pairwise similarities
$V_{\text{threshold}}$	similarity threshold (a vigilance parameter)
M_k	winning count of $\mathbf{y}_k$
$\mathcal{M}$	set of winning counts M_k ($M_k \in \mathcal{M}$)
$a(\mathbf{y}_k, \mathbf{y}_l)$	age of an edge between nodes $\mathbf{y}_k$ and $\mathbf{y}_l \in \mathcal{Y} \setminus \mathbf{y}_k$
$\mathcal{E}$	set of ages of edges ($a(\mathbf{y}_k, \mathbf{y}_l) \in \mathcal{E}$)
α_{del}	set of ages of deleted edges
a_{max}	edge deletion threshold

Correntropy [11] provides a generalized similarity measure between two arbitrary data points $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_d)$ as follows:

$$C(\mathbf{x}, \mathbf{y}) = \mathbf{E}\left[\kappa_\sigma\left(\mathbf{x}, \mathbf{y}\right)\right], \quad (9)$$

where $\mathbf{E}[\cdot]$ is the expectation operation, and $\kappa_\sigma(\cdot)$ denotes a positive definite kernel with a bandwidth σ. The correntropy is estimated as follows:

$$\hat{C}(\mathbf{x}, \mathbf{y}, \sigma) = \frac{1}{d}\sum_{i=1}^{d} \kappa_\sigma\left(x_i, y_i\right). \quad (10)$$

In this paper, we use the following Gaussian kernel in the correntropy:

$$\kappa_\sigma\left(x_i, y_i\right) = \exp\left[-\frac{(x_i - y_i)^2}{2\sigma^2}\right]. \quad (11)$$

A nonlinear metric called CIM is derived from the correntropy [11]. CIM quantifies the similarity between two data points $\mathbf{x}$ and $\mathbf{y}$ as follows:

$$\text{CIM}\left(\mathbf{x}, \mathbf{y}, \sigma\right) = \left[1 - \hat{C}(\mathbf{x}, \mathbf{y}, \sigma)\right]^{\frac{1}{2}}, \quad (12)$$

here, since the Gaussian kernel in (11) does not have the coefficient $\frac{1}{\sqrt{2\pi}\sigma}$, the range of CIM is limited to $[0, 1]$.

In general, the Euclidean distance suffers from the curse of dimensionality. However, CIM reduces this drawback since the correntropy calculates the similarity between two data points by using a kernel function. Moreover, it has also been shown that CIM with the Gaussian kernel has a high outlier rejection ability [11].

2.5.2. Kernel Density Estimator

A similarity measurement between an instance and a node has a large impact on the performance of clustering algorithms. CIM-based methods use the CIM as a similarity measurement. As defined in Eq. (12), the state of the CIM is controlled by a kernel bandwidth σ which is a data-dependent parameter. The initial state of σ is defined by training instances.

When a new node $\boldsymbol{y}_{K+1}$ is generated from $\boldsymbol{x}_n$, a kernel bandwidth σ_{K+1} is estimated from the past λ instances, i.e., $(\boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-2}, \ldots, \boldsymbol{x}_{n-\lambda})$, by using a kernel density estimation with the Gaussian kernel [12], which is defined as follows:

$$\boldsymbol{\Sigma} = \left(\frac{4}{2+d}\right)^{\frac{1}{4+d}} \boldsymbol{\Gamma}\lambda^{-\frac{1}{4+d}}, \quad (13)$$

where $\boldsymbol{\Gamma}$ denotes a rescale operator which is defined by a standard deviation of each attribute value among the λ instances. Here, $\boldsymbol{\Sigma}$ contains the bandwidth of each attribute. In this paper, the median of $\boldsymbol{\Sigma}$ is selected as a representative kernel bandwidth for the new node $\boldsymbol{y}_{K+1}$, i.e.,

$$\sigma_{K+1} = \text{median}\left(\boldsymbol{\Sigma}\right). \quad (14)$$

Note that when there is no node, the (λ+1)th instance becomes the initial node (i.e., $\boldsymbol{y}_1 = \boldsymbol{x}_{\lambda+1}$). Namely, the kernel bandwidth of $\boldsymbol{y}_1$ is estimated from the 1st to λth instances in a set of training instances $\boldsymbol{X}$.

2.5.3. CIM-based ART with Edge: CAE

CAE is a parameter-free, ART-based topological clustering algorithm capable of continual learning [6]. In general, ART-based algorithms require data-dependent parameters, such as a vigilance parameter (similarity threshold). In CAE, the similarity threshold is determined based on pairwise similarities among a certain number of nodes. The required number of nodes for calculating the similarity threshold is estimated using a Determinantal Point Processes (DPP)-based criterion [13, 14].

Additionally, an edge deletion threshold is estimated based on the age of each edge, inspired by the edge deletion mechanism of SOINN+ [15]. Empirical studies with synthetic and real-world datasets have shown that CAE outperforms state-of-the-art parameter-free and fixed-parameter algorithms in clustering performance [6].

The following sections provide the learning processes of CAE. Algorithm 1 summarizes the entire learning procedure of CAE.

Algorithm 1: Learning procedure of CAE

Input:
a set of data points $\mathcal{X}$.
Output:
a set of nodes $\mathcal{Y}$,
a set of ages of edges $\mathcal{E}$.

```
1  while existing data points to be trained do
2      Input a data point x (x ∈ X).
3      if the number of active nodes λ is not defined or the number of nodes |Y| is smaller than λ/2 or a similarity threshold
         V_threshold is not calculated then
4          Create a node as y_{|Y|+1} = x, and update a set of nodes as Y ← Y ∪ {y_{|Y|+1}}.
5          Estimate the diversity of nodes.                                   // (15),(16)
6          if the diversity of nodes is sufficient (i.e., D < 1.0e^{-6}) then
7              Calculate a similarity threshold V_threshold.                  // (18)
8      else
9          Select the 1st and 2nd nearest nodes from x (i.e., y_{s1} and y_{s2}) based on CIM.   // (19),(20)
10         Perform vigilance test, and create/update nodes and edges.         // (21) - (30)
11         Estimate an edge deletion threshold a_max.                         // (31)
12         Delete edges by using a_max.
13     if the number of presented data points is a multiple of λ then
14         Delete isolated nodes.
```

2.5.4. Estimation of Diversity of Nodes

In CAE, a similarity threshold is defined by pairwise similarities among nodes (i.e., $\mathcal{Y}$). Therefore, the diversity of nodes for calculating the similarity threshold is important to obtain an appropriate threshold value, which leads to good clustering performance.

The diversity D of the active node set $\mathcal{A}$ is estimated by a DPP-based criterion [13, 14] incorporating CIM as follows:

$$D = \det(\mathbf{R}), \tag{15}$$

where

$$\mathbf{R} = \left[\exp\left(1 - \mathrm{CIM}(\mathbf{y}_i, \mathbf{y}_j, \sigma)\right)\right]_{1 \le i,j \le |\mathcal{A}|}. \tag{16}$$

Here, $\det(\mathbf{R})$ is the determinant of the matrix $\mathbf{R}$, and $\mathbf{R}$ is a matrix of pairwise similarities between nodes in $\mathcal{A}$. A bandwidth σ for CIM is calculated from $\mathbf{H}$ in (13) by using the node set $\mathcal{A}$. As in (13), $\mathbf{H}$ contains the bandwidth of a kernel function in CIM. In this paper, the median of $\mathbf{H}$ is used as the bandwidth of the Gaussian kernel in CIM, i.e.,

$$\sigma = \mathrm{median}\left(\mathbf{H}\right). \tag{17}$$

In general, the diversity $D = 0$ means that the node set $\mathcal{A}$ is not diverse while $D > 0$ means $\mathcal{A}$ is diverse. In other words, the value of D becomes close to zero when a new node is created around the existing nodes.

In CAE, the value of λ is set as the two times the number of nodes (i.e., $2|\mathcal{A}|$) when the diversity D satisfies $D < 1.0\mathrm{e}^{-6}$. If the number of nodes becomes smaller than $\lambda/2$ after a node deletion process, λ is calculated again.

As shown in line 3 of Algorithm 1, the first $\lambda/2$ data points (i.e., $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_{\lambda/2}\}$) directly become nodes, i.e., $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{\lambda/2}\}$ where $\mathbf{y}_k = \mathbf{x}_k$ $(k = 1, 2, \ldots, \lambda/2)$. In addition, the bandwidth for the Gaussian kernel in CIM is assigned to each node, i.e., $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_{\lambda/2}\}$ where $\sigma_1 = \sigma_2 = \cdots = \sigma_{\lambda/2}$.

The value of λ is automatically updated in the proposed CAE algorithm. In an active node set $\mathcal{A}$, λ nodes are stored. When a new node is added to $\mathcal{A}$, an old node is removed to maintain the active node set size as λ. The addition of a new node and the removal of an old node are explained later.

2.5.5. Calculation of Similarity Threshold

The similarity threshold $V_{\text{threshold}}$ is calculated by the average of the minimum pairwise CIM values in the active node set $\mathcal{A}$ as follows:

$$V_{\text{threshold}} = \frac{1}{\lambda} \sum_{\mathbf{y}_i \in \mathcal{A}} \min_{\mathbf{y}_j \in \mathcal{A} \setminus \mathbf{y}_i} \left[\mathrm{CIM}\left(\mathbf{y}_i, \mathbf{y}_j, \mathrm{mean}(\mathcal{S})\right)\right], \tag{18}$$

where $\mathcal{S}$ is a set of bandwidths of the Gaussian kernel in CIM for $\mathcal{A}$. The bandwidth of each node in $\mathcal{A}$ is calculated by using (13) and (17) when a new node is created.

2.5.6. Selection of Winner Nodes

During the learning process of CAE, every time a data point $\mathbf{x}$ is given, two nodes that have a similar state to $\mathbf{x}$ are selected from $\mathcal{Y}$, namely the 1st winner node $\mathbf{y}_{s_1}$ and the 2nd winner node $\mathbf{y}_{s_2}$. The winner nodes are determined based on the value of CIM in line 9 of Algorithm 1 as follows:

$$s_1 = \arg\min_{\mathbf{y}_i \in \mathcal{Y}} \left[\mathrm{CIM}\left(\mathbf{x}, \mathbf{y}_i, \mathrm{mean}(\mathcal{S})\right)\right], \tag{19}$$

$$s_2 = \arg\min_{\mathbf{y}_i \in \mathcal{Y} \setminus \{\mathbf{y}_{s_1}\}} \left[\mathrm{CIM}\left(\mathbf{x}, \mathbf{y}_i, \mathrm{mean}(\mathcal{S})\right)\right], \tag{20}$$

where s_1 and s_2 denote the indices of the 1st and 2nd winner nodes, respectively. $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\mathcal{Y}|}\}$ is a set of Gaussian kernel bandwidths in CIM corresponding to a node-set $\mathcal{Y}$.

Note that the 1st winner node $\mathbf{y}_{s_1}$ becomes a new active node, and the oldest node in the active node set $\mathcal{A}$ (i.e., λ nodes in $\mathcal{Y}$) is replaced by the new one.

2.5.7. Vigilance Test

Similarities between the data point $\mathbf{x}$ and each of the 1st and 2nd winner nodes are defined in line 10 of Algorithm 1 as follows:

$$V_{s_1} = \mathrm{CIM}\left(\mathbf{x}, \mathbf{y}_{s_1}, \mathrm{mean}(\mathcal{S})\right), \tag{21}$$

$$V_{s_2} = \mathrm{CIM}\left(\mathbf{x}, \mathbf{y}_{s_2}, \mathrm{mean}(\mathcal{S})\right). \tag{22}$$

The vigilance test classifies the relationship between the data point $\mathbf{x}$ and the two winner nodes into three cases by using the similarity threshold $V_{\text{threshold}}$, i.e.,

- **Case I**

  The similarity between $\mathbf{x}$ and the 1st winner node $\mathbf{y}_{s_1}$ is larger (i.e., less similar) than $V_{\text{threshold}}$, namely:

$$V_{\text{threshold}} < V_{s_1} \leq V_{s_2}. \tag{23}$$

- **Case II**

  The similarity between $\mathbf{x}$ and the 1st winner node $\mathbf{y}_{s_1}$ is smaller (i.e., more similar) than $V_{\text{threshold}}$, and the similarity between $\mathbf{x}$ and the 2nd winner node $\mathbf{y}_{s_2}$ is larger (i.e., less similar) than $V_{\text{threshold}}$, namely:

$$V_{s_1} \leq V_{\text{threshold}} < V_{s_2}. \tag{24}$$

- **Case III**

  The similarities between $\mathbf{x}$ and the 1st and 2nd winner nodes (i.e., $\mathbf{y}_{s_1}$ and $\mathbf{y}_{s_2}$) are both smaller (i.e., more similar) than $V_{\text{threshold}}$, namely:

$$V_{s_1} \leq V_{s_2} \leq V_{\text{threshold}}. \tag{25}$$

2.5.8. Creation/Update of Nodes and Edges

Depending on the result of the vigilance test, a different operation is performed.

If the data point $\mathbf{x}$ is classified as Case I by the vigilance test (i.e., (23) is satisfied), a new node is created as $\mathbf{y}_{|\mathcal{Y}|+1} = \mathbf{x}$, and updated a node-set as $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{y}_{|\mathcal{Y}|+1}\}$. Here, the node $\mathbf{y}_{|\mathcal{Y}|+1}$ becomes a new active node, and the oldest node in the active node set $\mathcal{A}$ (i.e., λ nodes in $\mathcal{Y}$) is replaced by the new one. In addition, a bandwidth $\sigma_{|\mathcal{Y}|+1}$ for $\mathbf{y}_{|\mathcal{Y}|+1}$ is calculated by (13) and (17) with the active node set $\mathcal{A}$, and the winning count of $\mathbf{y}_{|\mathcal{Y}|+1}$ is initialized as $M_{|\mathcal{Y}|+1} = 1$.

If the data point $\mathbf{x}$ is classified as Case Ⅱ by the vigilance test (i.e., (24) is satisfied), first, the winning count of $\mathbf{y}_{s_1}$ is updated as follows:

$$M_{s_1} \leftarrow M_{s_1} + 1. \tag{26}$$

Then, $\mathbf{y}_{s_1}$ is updated as follows:

$$\mathbf{y}_{s_1} \leftarrow \mathbf{y}_{s_1} + \frac{1}{M_{s_1}} (\mathbf{x} - \mathbf{y}_{s_1}), \tag{27}$$

here, the node $\mathbf{y}_{s_1}$ becomes a new active node, and the oldest node in the active node set $\mathcal{A}$ (i.e., λ nodes in $\mathcal{Y}$) is replaced by the new one.

When updating the node, the difference between $\mathbf{x}$ and $\mathbf{y}_{s_1}$ is divided by M_{s_1}. Thus, the change of the node position is smaller when M_{s_1} is larger. This is because the information around the node, where data points are frequently given, is important and should be held by the node.

The age of each edge connected to the 1st winner node $\mathbf{y}_{s_1}$ is also updated as follows:

$$a(\mathbf{y}_{s_1}, \mathbf{y}_k) \leftarrow a(\mathbf{y}_{s_1}, \mathbf{y}_k) + 1 \quad (\mathbf{y}_k \in \mathcal{N}_{s_1}), \tag{28}$$

where $\mathcal{N}_{s_1}$ is a set of all neighbor nodes of the node $\mathbf{y}_{s_1}$.

If the data point $\mathbf{x}$ is classified as Case Ⅲ by the vigilance test (i.e., (25) is satisfied), the same operations as Case Ⅱ (i.e., (26)-(28)) are performed. In addition, if there is an edge between $\mathbf{y}_{s_1}$ and $\mathbf{y}_{s_2}$, an age of the edge is reset as follows:

$$a(\mathbf{y}_{s_1}, \mathbf{y}_{s_2}) \leftarrow 1. \tag{29}$$

In the case that there is no edge between $\mathbf{y}_{s_1}$ and $\mathbf{y}_{s_2}$, a new edge is defined with an age of the edge by (29).

After updated the edge information, the neighbor nodes of $\mathbf{y}_{s_1}$ are updated as follows:

$$\mathbf{y}_k \leftarrow \mathbf{y}_k + \frac{1}{10M_k} (\mathbf{x} - \mathbf{y}_k) \quad (\mathbf{y}_k \in \mathcal{N}_{s_1}). \tag{30}$$

Apart from the above operations in Cases I-Ⅲ, the nodes with no edges are deleted (and removed from the active node set $\mathcal{A}$) every λ data points for the noise reduction purpose (i.e., the node deletion interval is the presentation of λ data points), which is performed in line 14 of Algorithm 1.

With respect to the active node set $\mathcal{A}$, its update rules are summarized as follows. In Case I, a new node is directly created by the data point $\mathbf{x}$ and added to $\mathcal{A}$. In Case Ⅱ and Case Ⅲ, the updated winner node in (27) is added to $\mathcal{A}$. In all cases, the oldest active node is removed from $\mathcal{A}$. Then, in line 14 of Algorithm 1, all active nodes with no edges are removed. After this removal procedure, the number of active nodes can be smaller than λ.

2.5.9. Estimation of Edge Deletion Threshold

CAE estimates an edge deletion threshold based on the ages of the current edges and the deleted edges, which is inspired by the edge deletion mechanism of SOINN+ [15].

The edge deletion threshold a_{max} is defined as follows:

$$a_{\text{max}} = \bar{\alpha}_{\text{del}} \frac{|\alpha_{\text{del}}|}{|\alpha_{\text{del}}| + |\alpha|} + a_{\text{thr}} \left(1 - \frac{|\alpha_{\text{del}}|}{|\alpha_{\text{del}}| + |\alpha|}\right), \tag{31}$$

where α_{del} is the set of ages of all the deleted edges during the learning process, $|\alpha_{\text{del}}|$ is the number of elements in α_{del}, $\bar{\alpha}_{\text{del}}$ is the arithmetic mean of α_{del}, α is the set of ages of edges which connect to $\mathbf{y}_{s_1}$ ($\alpha \subset \mathcal{E}$), and $|\alpha|$ is the number of elements in α. The coefficient a_{thr} is defined as follows:

$$a_{\text{thr}} = \alpha_{0.75} + \text{IQR}(\alpha), \tag{32}$$

where $\alpha_{0.75}$ is the 75th percentile of elements in α, and $\text{IQR}(\alpha)$ is the interquartile range.

The edge deletion threshold a_{max} is updated each time the age of an edge increases, which is performed in line 11 of Algorithm 1.

2.5.10. Deletion of Edges

If an edge has an age greater than the edge deletion threshold a_{max}, the edge is deleted and the set of ages of deleted edges α_{del} is updated, which are performed in line 12 of Algorithm 1.

2.6. CA+

Although CAE [6] is a state-of-the-art parameter-free ART-based topological clustering algorithm, CAE tends to generate a large number of nodes. The main reason for this phenomenon is that the diversity D of the node set $\mathcal{A}$ defined in (15) is unlikely to be $D < 1.0\text{e}^{-6}$. In other words, the pairwise similarity matrix $\mathbf{R}$ defined in (16) is not appropriate in the case where a large number of nodes is not preferable. In CA+, a correntropy-based pairwise similarity matrix $\mathbf{R}$ is used, which is defined as follows:

$$\mathbf{R} = \left[\exp\left(\hat{C}(\mathbf{y}_i, \mathbf{y}_j, \sigma)\right)\right]_{1 \leq i,j \leq |\mathcal{A}|}, \tag{33}$$

where $\hat{C}(\cdot)$ is correntropy which is defined in (10).

In comparison to the definition in (16), the definition in equation (33) tends to have a larger difference in the values of the elements of $\mathbf{R}$. As a result, the value of D is close to zero (i.e., $D < 1.0\text{e}^{-6}$) when the diversity of the node set $\mathcal{A}$ is small.

Additionally, CA+ does not have edges because CA+ is designed to preserve data for clients in federated learning. The learning procedure is almost the same, and the differences between CA+ and CAE are summarized as follows:

- For CA+, the learning processes related to the edge information are removed from Algorithm 1, namely lines 10-14. As a result, the output of CA+ is only a node set $\mathcal{Y}$.
- Since all the nodes of CA+ are isolated, a process for deleting isolated nodes (lines 14 in Algorithm 1) is removed.
- In CA+, the weight updating rule in (30) uses the edge information for defining the neighbor nodes $\mathcal{N}_{s_1}$ (see also lines 10 in Algorithm 1). Since CA+ has no edges, CA+ updates only the 2nd winner node $\mathbf{y}_{s_2}$ as follows:

$$\mathbf{y}_{s_2} \leftarrow \mathbf{y}_{s_2} + \frac{1}{100 M_{s_2}} \left(\mathbf{x} - \mathbf{y}_{s_2} \right), \tag{34}$$

where M_{s_2} is the winning count of $\mathbf{y}_{s_2}$.

In CA+, $\mathbf{y}_{s_2}$ does not need significant movement because more nodes are generated than in CA+. Therefore, the coefficient in (34) is set to $1/100$ in CA+ whereas in CAE, it is $1/10$ in CAE.

3. Proposed Method: CUCL_CA+

In this section, we first explain CUCL_shared, which shares the codebook (the space used for quantization) across tasks. Then, we describe CUCL_CA+, which merges codewords in the shared codebook using CA+ (introduced in Section 2.6) to control the number of codewords.

3.1. Codebook Sharing in CUCL: CUCL_shared

In the conventional CUCL method, each task uses eight separate sub-codebooks, leading to a total of 8×M codebooks for M tasks. Because of this design, we must manage task indices to determine which codebook to use, which becomes a problem if the task information is not known.

To address this issue, we propose CUCL_shared, which uses a single shared codebook across all tasks. Fig 3 shows that In CUCL_shared, we add a specified number of codewords (N_add) at the start of each task and then allow these codewords to move to their optimal positions through training. Since all tasks use the same codebook, there is no need to manage task indices.

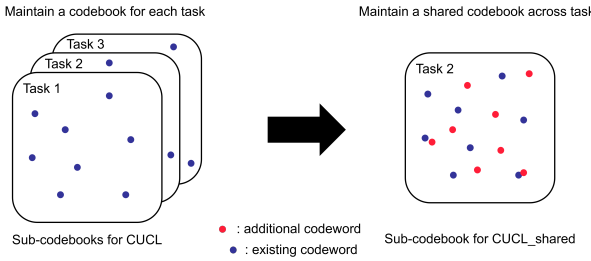


Fig. 3. Differences between CUCL and CUCL_shared+: This illustration shows that on the left, how codebooks increase for each new task in the existing method (CUCL). In contrast, on the right, it illustrates how the proposed method, CUCL_shared, adds new codewords to the existing shared codebook at the start of Task 2. In this way, CUCL_shared continues to use the same codebook across tasks while only adding the necessary codewords when needed.

3.2. Merging Redundancy Codewords by CA+: CUCL_CA+

Although CUCL_shared eliminates the need for task index management, the number of codewords can still grow indefinitely as more tasks are added. Moreover, while the positions of codewords are optimized during training, some codewords may end up clustered together, causing redundancy. To address this problem, we use CA+, introduced in sec 2.6, to merge nearby codewords and reduce redundancy in Fig. 4.

Because CA+ can dynamically and continuously cluster codewords, it efficiently merges them by considering their distribution. As shown in Algorithm 2, CUCL_CA+ adds new codewords at the start of each task (line 4 in Algorithm 2) and then merges them with CA+ at the end of the task (line 20 in Algorithm 2). This approach allows us to add codewords while still merging redundant ones, preventing the codebook from growing excessively.

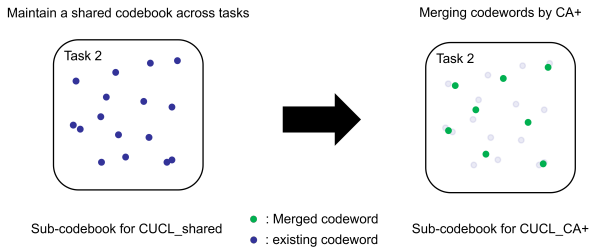


Fig. 4. Differences between CUCL_shared and CUCL_CA+: This illustration shows, on the left, the codebook for Task 2 in the first version of proposed method (i.e., CUCL_shared), and on the right, how codewords are merged in the second version of proposed method (i.e., CUCL_CA+). As shown, CUCL_CA+ merges closely located codewords to reduce redundancy.

3.3. Codeword Merging with CA+

4. Numerical Experiments

In this section, we evaluate the performance of our proposed method in the task-incremental learning scenario.

4.1. Compared Algorithm

We use two SSL architectures, Simsiam and Barlow twins, with four configurations: Finetune, CUCL, CUCL_shared, and CUCL_CA+. Finetune is a SSL architecture without any proposal. CUCL introduces Product Quantization to SSL architectures. CUCL_shared refers to methods where the number of codewords increases by 8, 10, 50, or 100 at the start of each task. CUCL_CA+ is a version of CUCL_shared with merging methods.

4.2. Experimental Setting

We use ResNet18 [44] as a backbone of Simsiam and Barlow Twins. The experimental parameters are as follows: the batch size is set to 256, and the number of training epochs is 200. All other parameters follow the original settings [4]. To mitigate the effect of randomness, each experiment is conducted three times, and the average results are shown.

We evaluate the trained encoder network under two conditions: In-Distribution (ID) and Out-of-Distribution (OOD). In the ID condition, the training and test data are the same, while in the OOD condition, they are different.

4.3. Dataset

The datasets for the ID condition are CIFAR-100 [16] and Tiny-ImageNet [17] and they are divided into 10 tasks with 10 classes each. For the OOD condition, the training data consists of Tiny-ImageNet, while the test datasets include CIFAR-10, CIFAR-100, SVHN, and STL-10.

4.4. Evaluation Metrics

The performance metrics often used in continual learning are Average Accuracy ($\mathrm{ACC_T}$) which is defined as:

$$\mathrm{ACC_T} = \frac{1}{T} \sum_{i=1}^{T} A_{T,i}, \tag{35}$$

Algorithm 2: Learning procedure of the proposed method CUCL_CA+

Input: Training dataset I, batch size S, the number of additional codewords N_{add}, number of epochs N_{epoch}, number of codebooks M, replay buffer size B
Output: Trained network

while task is remained **do**
  for *epoch* $= 1$ *to* N_{epoch} **do**
    if task $\neq 1$ **then**
      Adding N_{add} codewords to existing codewords in M codebooks
    end
    Split dataset I into mini-batches
    foreach mini-batch S in I **do**
      foreach data point I_i in S **do**
        $I_{i1}, I_{i2} \leftarrow$ Perform data augmentation on I_i to obtain its augmented version
        $\boldsymbol{X}_{i1}, \boldsymbol{X}_{i2} \leftarrow$ Encode I_{i1}, I_{i2} to obtain feature vectors
        Split $\boldsymbol{X}$ into the Mth sub-vectors $(\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots,\ \boldsymbol{x}_i,\ \ldots,\ \boldsymbol{x}_M)$
        $\boldsymbol{z_i} \leftarrow$ Quantize each $\boldsymbol{x_i}$ in each codebook // (6)
      end
      Calculate a loss function for mini-batch M
      Update the network using the calculated loss // (7),(8)
    end
  end
  Calculate the distance between data and its nearest codewords
  B feature vectors are selected as a replay buffer
  Codewords are merged by CA+ to reduce the redundancy
end

where $A_{T,i}$ is the performance on the i-th task after training on the T-th task. $\mathrm{ACC_T}$ measures the average classification performance across all tasks after the model has been trained sequentially on the T-th tasks. In addition, we use Mean $\mathrm{ACC_T}$ defined as follows:

$$\mathrm{MAA} = \frac{1}{T}\sum_{j=1}^{T}\left(\frac{1}{j}\sum_{i=1}^{j} A_{j,i}\right). \tag{36}$$

The MAA is computed by taking the average of the accuracy for tasks at each training point. A high MAA indicates a continual learning model that consistently achieves high classification accuracy across the training process.

4.5. Comparison of CUCL with CUCL_shared

In this experiment, we investigate the impact on accuracy when utilizing a shared codebook. Table II presents the experimental results comparing the Finetune method, the conventional CUCL approach, and the shared codebook variant, denoted as CUCL_shared. Overall, the CUCL-based methods demonstrate higher accuracy than the Finetune approach. However, CUCL_shared exhibits slightly lower performance compared to the conventional CUCL method. Nevertheless, due to its advantage of eliminating the need for task index management and enabling continuous learning without explicit consideration of task boundaries, the proposed method is considered particularly beneficial in scenarios where the task structure is not predetermined.

4.6. Comparison of CUCL_shared with CUCL_CA+

In this experiment, we examine how merging codewords using CA+ affects accuracy. Table III presents the experimental results comparing the first proposed method CUCL_shared, which uses a shared codebook, with CUCL_CA+, which integrates redundant codewords in the shared codebook by applying CA+.

Overall, the results indicate that accuracy improves as the number of additional codewords increases. This is because retaining more codewords allows for richer information to be utilized. Moreover, the codeword merging method using CA+ shows sufficiently competitive performance compared with CUCL_shared. In addition, since redundant codewords are removed, CUCL_CA+ reduces memory usage, making it advantageous for practical applications.

4.7. Comparison of all algorithms on OOD datasets

In this experiment, we discuss the performance differences on OOD data for all methods. The motivation for this experiment is that even if existing methods (for example, CUCL) achieve high accuracy on ID data, such high accuracy might be due to overfitting, which is why it is necessary to evaluate the generalization performance on OOD data.

Our proposed methods, CUCL_shared and CUCL_CA+, showed slightly lower accuracy than existing methods on ID data. However, they performed competitively on OOD data. In particular, when using SimSiam, both CUCL_shared and CUCL_CA+ achieved higher accuracy than the existing methods. This suggests that learning across tasks with a shared codebook leads to balanced feature learning without relying too much on task-specific information.

Furthermore, CUCL_CA+ does not require task index management and can remove redundant codewords, which reduces memory usage. With its competitive accuracy, especially in terms of generalization, CUCL_CA+ is a promising method for practical use.

4.8. Analysis of the Number of Codeword before and after Mergning

In this analysis, we examine how the number of additional codewords influences performance when merging codewords using CA+, based on the results in Table III.

Fig. 5 show the number of codewords before and after merging in two scenarios: adding eight codewords at a time (Figs. (a) and (b)) and adding fifty codewords at a time (Figs. (c) and (d)). When adding eight codewords at a time (Figs. (a) and (b)), the difference between the green bars (i.e., before merging) and the black bars (i.e., after merging) is about one-third of the total. In contrast, when adding fifty codewords at a time (Figs. (c) and (d)), more than half of the codewords are removed after merging.

During training, codewords move to optimal positions according to the loss function. However, adding fifty codewords at a time

causes many of these optimized codewords to be removed, effectively resetting over half of them each time, which seems to lower accuracy. On the other hand, when adding eight codewords at a time, fewer codewords are removed, so the total number of codewords remains more stable. As a result, redundancy is removed while preserving the learned structure, leading to more efficient training and improved accuracy.

5. Conclusion

This paper proposed a self-supervised learning method for PQ that utilizes a unified codebook. In our proposed method, models do not need to use the current task index because of the shared codebook. Additionally, CA+ can control the number of codewords without a significant drop in accuracy. Numerical experiments demonstrated that CUCL_CA+ achieves better performance in the OOD condition, indicating that a shared codebook enhances generalization. Since our proposed method is based on SSL architectures, it can be easily adapted to other SSL models without modifying the core architecture.

As a future research topic, we will extend our approach to a class-incremental learning setting, ensuring that task information is not required during testing. While our current method does not use task information during training, the next step is to make it independent of task indexing in the evaluation phase as well. Additionally, we will investigate the use of shared codewords as a buffer to retain previously learned representations and facilitate knowledge transfer across tasks.

Acknowledgments

I express my deepest gratitude to Professor Yusuke Nojima and Associate Professor Naoki Masuyama at the Department of Core Informatics, Graduate School of Informatics, Osaka Metropolitan University, for their dedicated guidance and support in conducting this research and writing my master's thesis.

I would also like to extend my sincere appreciation to Professor Katsuhiro Honda and Associate Professor Hitoshi Hojo at the Department of Core Informatics, Graduate School of Informatics, Osaka Metropolitan University, for their valuable insights and advice in summarizing this research.

References

[1] C. Luo, X. Yang, and A. Yuille, "Self-supervised pillar motion learning for autonomous driving," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3182–3191, 2021.

[2] S. C. Huang, A. Pareek, M. E. K. Jensen, M. P. Lungren, S. Yeung, and A. S. Chaudhari, "Self-supervised learning for medical image classification: A systematic review and implementation guidelines," *NPJ Digital Medicine*, vol. 6, 2023.

[3] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[4] C. Cheng, J. Song, X. Zhu, J. Zhu, L. Gao, and H. Shen, "CUCL: Codebook for unsupervised continual learning," The 31st ACM International Conference on Multimedia, pp. 1729–1737, 2023.

[5] N. Masuyama, Y. Nojima, Y. Toda, C. K. Loo, H. Ishibuchi, and N. Kubota, "Privacy-preserving continual federated clustering via adaptive resonance theory," *IEEE Access*, vol. 12, pp. 139692–139710, 2024.

[6] N. Masuyama, T. Takebayashi, Y. Nojima, C. K. Loo, H. Ishibuchi, and S. Wermter, "A parameter-free adaptive resonance theory-based topological clustering algorithm capable of continual learning," *arXiv preprint arXiv:2305.01507*, 2023.

[7] D. W. Zhou, Q. W. Wang, Z. H. Qi, H. J. Ye, D. C. Zhan, and Z. Liu, "Class-incremental learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 9851–9873, 2024.

[8] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15750–15758, June 2021.

[9] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *International Conference on Machine Learning*, pp. 12310–12320, PMLR, 2021.

[10] Y. K. Jang and N. I. Cho, "Self-supervised product quantization for deep unsupervised image retrieval," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[11] W. Liu, P. P. Pokharel, and J. C. Príncipe, "Correntropy: Properties and applications in non-Gaussian signal processing," *IEEE Transactions on Signal Processing*, vol. 55, no. 11, pp. 5286–5298, 2007.

[12] D. J. Henderson and C. F. Parmeter, "Normal reference bandwidths for the general order, multivariate kernel density derivative estimator," *Statistics & Probability Letters*, vol. 82, no. 12, pp. 2198–2205, 2012.

[13] A. Kulesza and B. Taskar, "Determinantal point processes for machine learning," *Foundations and Trends® in Machine Learning*, vol. 5, no. 2–3, pp. 123–286, 2012.

[14] J. Parker-Holder, A. Pacchiano, K. Choromanski, and S. Roberts, "Effective diversity in population based reinforcement learning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, no. 1515 in NIPS'20, (Red Hook, NY, USA), pp. 18050–18062, Curran Associates Inc., December 2020.

[15] C. Wiwatcharakoses and D. Berrar, "SOINN+, a self-organizing incremental neural network for unsupervised learning from noisy data streams," *Expert Systems with Applications*, vol. 143, p. 113069, 2020.

[16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep. 0, University of Toronto, 2009.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

Author Biography

Takanori Takebayashi

I obtained my B.S. in Computer Science from Osaka Prefecture University, Japan, in 2023. I am a master's student in the Department of Core Informatics, Osaka Metropolitan University. My research interests include clustering and continual learning.

TABLE II
COMPARISON OF PERFORMANCE UNDER THE ID CONDITION

Dataset	Architecture	Finetune	CUCL	CUCL_shared (8)	CUCL_shared (10)	CUCL_shared (50)	CUCL_shared (100)
CIFAR-100	SimSiam	69.444 (4.664)	**74.885** (0.725)	72.930 (0.905)	72.903 (0.904)	73.692 (1.232)	73.626 (1.052)
	Barlow Twins	69.220 (2.476)	**71.905** (1.741)	70.592 (2.186)	70.812 (2.359)	71.165 (2.010)	70.845 (2.505)
Tiny-ImageNet	SimSiam	52.835 (7.775)	**61.774** (3.011)	60.157 (2.635)	60.175 (2.666)	61.015 (2.009)	61.443 (2.141)
	Barlow Twins	56.860 (2.556)	**59.982** (1.941)	58.316 (2.181)	58.526 (2.477)	58.419 (2.520)	58.520 (0.905)

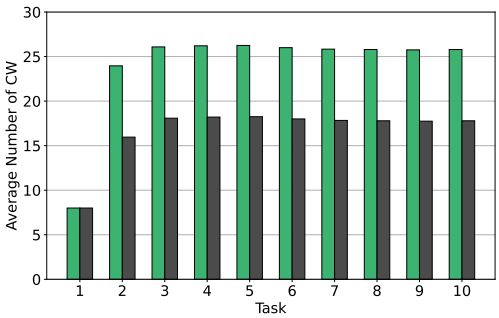
TABLE III
COMPARISON OF PERFORMANCE UNDER THE ID CONDITION

Dataset	Architecture	CUCL_shared (8)	CUCL_CA+ (8)	CUCL_shared (10)	CUCL_CA+ (10)	CUCL_shared (50)	CUCL_CA+ (50)	CUCL_shared (100)	CUCL_CA+ (100)
CIFAR-100	SimSiam	72.930 (0.905)	72.948 (0.927)	72.903 (0.904)	73.107 (0.778)	**73.692** (1.232)	73.564 (1.293)	73.626 (1.052)	73.629 (1.048)
	Barlow Twins	70.592 (2.186)	70.715 (2.151)	70.812 (2.359)	70.811 (2.572)	**71.165** (2.010)	71.131 (2.401)	70.845 (2.505)	70.621 (2.528)
Tiny-ImageNet	SimSiam	60.157 (2.635)	60.236 (2.573)	60.175 (2.666)	60.420 (2.713)	**61.015** (2.009)	60.865 (2.010)	61.443 (2.141)	61.273 (2.098)
	Barlow Twins	58.316 (2.181)	58.304 (2.206)	58.526 (2.477)	58.573 (2.466)	**58.419** (2.520)	58.417 (2.548)	58.520 (0.905)	58.458 (2.248)

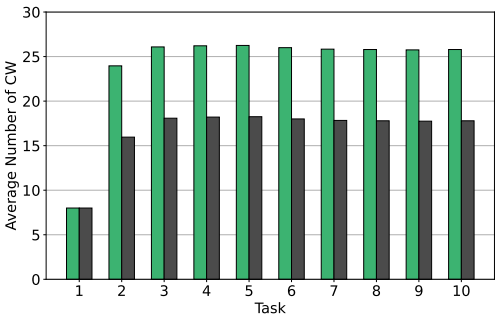
TABLE IV
AVERAGE ACCURACY UNDER THE OOD CONDITION

Dataset	Architecture	Finetune	CUCL	CUCL_shared (8)	CUCL_CA+ (8)	CUCL_shared (50)	CUCL_CA+ (50)
CIFAR-10	SimSiam	82.570 (5.320)	83.910 (5.542)	**84.143** (4.960)	84.057 (4.799)	84.093 (5.350)	83.833 (5.336)
	Barlow Twins	83.287 (5.797)	**84.043** (5.381)	83.127 (5.396)	83.397 (5.756)	83.363 (5.064)	83.337 (5.381)
CIFAR-100	SimSiam	54.523 (3.924)	57.370 (3.637)	57.660 (3.958)	57.803 (3.624)	**58.153** (3.384)	57.910 (3.446)
	Barlow Twins	55.273 (3.534)	**58.117** (3.405)	55.323 (5.396)	55.437 (4.080)	55.890 (4.121)	55.297 (3.601)
SVHN	SimSiam	24.489 (0.517)	26.720 (0.630)	**26.925** (0.672)	26.621 (0.322)	26.086 (0.700)	25.905 (0.874)
	Barlow Twins	25.603 (0.906)	**26.681** (0.631)	26.311 (0.568)	26.330 (0.501)	24.980 (0.381)	25.162 (0.274)
STL-10	SimSiam	86.754 (6.940)	88.804 (5.792)	88.696 (5.589)	89.050 (5.368)	**89.221** (5.701)	88.958 (5.675)
	Barlow Twins	**87.867** (6.215)	87.704 (5.935)	87.758 (5.768)	87.308 (5.946)	87.575 (5.858)	87.567 (6.055)

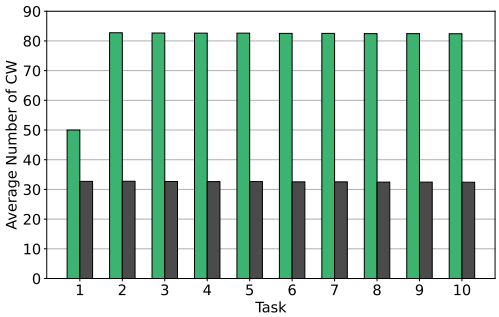
The number in parentheses next to the method name indicates the number of additional codewords at the start of the task.
The bold values in the table represent the best results, and the values in parentheses indicate the standard deviation.



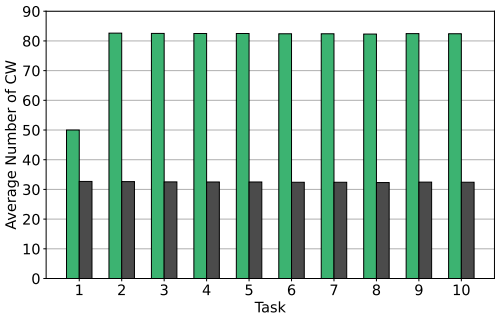
(a) The number of CW before and after merging 8 additional CW in CIFAR-100 for SimSiam



(b) The number of CW before and after merging 8 additional CW in CIFAR-100 for Barlow Twins



(c) The number of CW before and after merging 50 additional CW in CIFAR-100 for SimSiam



(d) The number of CW before and after merging 50 additional CW in CIFAR-100 for Barlow Twins

Fig. 5. The Number of Codewords before and after Mergnin