# CONTINUAL LEARNING USING PSEUDO-REPLAY VIA LATENT SPACE SAMPLING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

This paper investigates continual learning in the setting of *class-incremental learning* (CIL). Although numerous techniques have been proposed, CIL remains to be a highly challenging problem due to *catastrophic forgetting* (CF). However, so far few existing techniques have made use of pre-trained image feature extractors. In this paper, we propose to use a recently reported strong pre-trained feature extractor called CLIP and a novel and yet simple pseudo-replay method to deal with CF. The proposed method is called PLS. Unlike the popular pseudo-replay approach that builds data generators to generate pseudo previous task data, PLS works in the latent space by sampling pseudo feature vectors of previous tasks from the last layer of the pre-trained feature extractor. PLS is not only simple and efficient but also does not invade data privacy due to the fact that it works in the latent feature space. Experimental results demonstrate that PLS outperforms state-of-the-art baselines by a large margin when both PLS and the baselines leverage the CLIP pre-trained image feature extractor.

## 1 INTRODUCTION

A primary goal of continual learning (CL) is to learn a sequence of tasks incrementally in a single neural network without catastrophic forgetting (CF). CF is the phenomenon that occurs when learning a new task, the system needs to modify the parameters learned for old tasks, which may cause performance degradation of the tasks learned earlier, i.e., forgetting (McCloskey & Cohen, 1989). This paper focuses on the challenging CL setting of *class-incremental learning* (CIL) (van de Ven & Tolias, 2019). In this setting, each task consists of a set of unique classes to be learned. After learning a sequence of such tasks, the resulting network can classify a test instance from any class that has been learned so far with no task related information provided.

Existing research has proposed many techniques to deal with CF. However, not many methods have used pre-trained feature extractors except those methods used in continual learning of natural language processing (NLP) tasks because using pre-trained feature extractors (e.g., those language models like BERT, RoBERTa, BART, GPT-2 and GPT-3) can achieve significantly better results and thus is a standard approach in NLP. Using pre-trained feature extractors has also been used in computer vision although to a lesser extent (Hu et al., 2021). However, with strong pre-trained feature extractors appearing in computer vision, the situation is changing. Recently, a strong pre-trained multi-modal (image and text) transformer model, called CLIP, has been reported (Radford et al., 2021). CLIP was trained using contrastive loss on a large set of 400 million image and caption pairs collected from the Internet. It has been shown that for image classification, using CLIP, a zero-shot method is as strong as fully supervised learning approaches (Radford et al., 2021). CLIP consists of an image encoder and a text encoder. In this work, we use the image encoder, which is a pre-trained feature extractor. We will see that CLIP, together with the proposed method for dealing with CF, enables CL to achieve remarkably better results than without using it.

The proposed method for dealing with CF is called PLS (*Pseudo-replay via Latent space Sampling*). PLS is a pseudo-replay method that generates pseudo feature vectors of the previous classes by sampling from the final layer of CLIP's image encoder. This is entirely different from existing pseudo-replay methods that train a data generator to generate pseudo past data. Building such data generators incrementally often need to handle CF too, which make generators complex (Lesort et al., 2019). With a pre-trained feature extractor, entirely different pseudo-data generation methods can

be designed. The generation process of PLS involves only sampling from the pre-trained feature extractor and thus has no CF problem itself, which is a major advantage (see more advantages below). The sampled feature vectors and the new task data are used together to train the new task and to deal with CF. PLS assumes that the feature vectors of a class follow a multivariate Gaussian distribution in the latent feature space, which is identified by the mean vector and covariance matrix for a class. However, a covariance matrix can be very large. For example, given a $d$ dimensional latent feature space, the number of entries needed to store the covariance matrix of a class is $d \times d$, which is not scalable for learning involving a large number of classes, e.g., 1000 classes in the ImageNet dataset. To deal with the problem, this paper proposes to compute and save a small number of eigenpairs, (eigenvalue, eigenvector), and use them to directly sample pseudo-replay feature vectors. Our experimental results show that this simple approach works very well.

Using a pre-trained feature extractor has at least three major advantages: (1) Due to the rich features in the feature extractor, it enables a CIL method to produce much better results than without using it (see Section 4.2). (2) It makes the system much more efficient because the pre-trained feature extractor is fixed in learning. Thus, learning a new task involves only training the final classification layer(s). (3) It does not invade data privacy as it samples replay data in the latent feature space. Two existing families of approaches related to PLS are *replay* and *pseudo-replay*. In a replay method, the system memorizes a small number of training samples from each previous task or class in a memory buffer. In a pseudo-replay method, the system typically learns a data generator from the previous tasks to generate pseudo-samples of the previous tasks. The replay approach is not suitable for applications that have data privacy concerns, e.g., the data of each task might be owned by a different user, and the user does not want his/her private data seen/shared by any other users or transferred to a central server. However, sharing the learned models is acceptable and is a common practice, e.g., in federated learning (Zhang et al., 2021). Pseudo-replay methods are also problematic because they generate pseudo raw data of each task, which can be too similar to the real data. Using a pre-trained feature extractor makes it possible for both replay and pseudo-replay to work in the latent feature space because the pre-trained feature extractor is fixed during learning. Without a pre-trained feature extractor, both replay and pseudo-replay need to use the raw data or the generated pseudo raw data in learning because in both cases, the feature extractor needs to be updated in incremental learning of each task using the save raw data or the generated pseudo raw data. PLS samples feature vectors from the final layer of the pre-trained feature extractor, but does not generate pseudo raw data. Thus, PLS involves no raw data after a task is trained, and does not invade the users' data privacy.

Experimental results show that the proposed method PLS markedly outperforms the latest baselines by a large margin. On average over 8 sets of experiments, its accuracy is better than the best baseline by 6.5%, where both PLS and the baselines use the pre-trained CLIP feature extractor. We will also see that baselines using the pre-trained CLIP feature extractor markedly outperforms their original versions without using a pre-trained feature extractor or using a weaker feature extractor.

## 2 Related Work

There are a large number of approaches that have been proposed for CL to overcome CF. The main techniques can be categorized into a few families. The first family of approaches is based regularizations (Kirkpatrick et al., 2017) and knowledge distillation (Li & Hoiem, 2016). They aim to minimize changes to previous learned knowledge or models (Jung et al., 2016; Camoriano et al., 2017; Fernando et al., 2017; Rannen Ep Triki et al., 2017; Seff et al., 2017; Zenke et al., 2017; Kemker & Kanan, 2018; Ritter et al., 2018; Schwarz et al., 2018; Xu & Zhu, 2018; Castro et al., 2018; Dhar et al., 2019; Hu et al., 2019; Lee et al., 2019; Liu et al., 2020b).

The second family of approaches, called *experience replay*, or *replay*, memorizes a small number of training samples of the previous tasks. In learning a new task, the saved samples are added to the samples of the new task to train the model to adapt the previous knowledge to suit both the new task and the previously learned tasks (Rusu et al., 2016; Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017; Chaudhry et al., 2019; de Masson d'Autume et al., 2019; Hou et al., 2019; Wu et al., 2019; Rolnick et al., 2019; Buzzega et al., 2020; Zhao et al., 2020; Rajasegaran et al., 2020; Liu et al., 2021). The proposed PLS does not save any samples from previous tasks. As mentioned in the introduction section, replay-based methods are not suitable for applications that have privacy concerns, which do not allow memorizing any raw data from a task.

Instead of saving some samples, the third family of approaches, called *pseudo-replay*, builds data generators of the previous tasks to generate pseudo training data of previous tasks and use them to jointly train the new task (Gepperth & Karaoguz, 2016; Kamra et al., 2017; Shin et al., 2017; Wu et al., 2018; Seff et al., 2017; Wu et al., 2018; Kemker & Kanan, 2018; Hu et al., 2019; Hayes et al., 2019; Rostami et al., 2019; Ostapenko et al., 2019; Ayub & Wagner, 2021). If privacy is a concern, this approach is also problematic as we discussed in the Introduction section. The PASS system (Zhu et al., 2021) saves feature prototypes and replay them by adding random noise. PLS is different as it draws feature vectors from the latent space and performs markedly better than PASS (see Sec. 4.2).

The fourth family of approaches uses parameter isolation. This approach is exemplified by the HAT (Serrà et al., 2018) system, which trains hard attentions or masks to protect the learned model for each task. This is a highly effective approach for task-incremental learning (TIL), but not suitable for CIL (class-incremental learning), which is our focus. This method is also used in some other TIL systems (Fernando et al., 2017; Ke et al., 2020). Another set of related methods finds a sub-network for each task by pruning (Mallya & Lazebnik, 2017; Wortsman et al., 2020; Hung et al., 2019). However, these methods are mainly for TIL and they require the task-id to be provided for each test sample. There are also several TIL systems that do not use this approach, e.g. UCL (Ahn et al., 2019), ADP (Yoon et al., 2020), CCLL (Singh et al., 2020), HyperNet (von Oswald et al., 2020), PackNet Mallya & Lazebnik (2017), CPG (Hung et al., 2019), and SupSup (Wortsman et al., 2020). Our approach is for CIL and no task-id is needed during testing.

Beyond those popular methods discussed above, there is a long list of other CL approaches, e.g., a network of experts (Aljundi et al., 2016), orthogonal projection or subspace (Zeng et al., 2019; Chaudhry et al., 2020), generalized CL (Mi et al., 2020), bilevel optimization (Liu et al., 2020a), reinforcement learning (Xu & Zhu, 2018), conceptor-aided backprop (He & Jaeger, 2018), gating networks (Masse et al., 2018; Abati et al., 2020), incremental moment matching (Lee et al., 2017), expandable networks (Li et al., 2019), online Laplace approximation (Ritter et al., 2018), etc. None of them uses pseudo-replay or pre-trained models.

Several existing systems have used pre-trained feature extractors. Almost all existing methods for continual learning of natural language processing tasks use pre-trained language models as feature extractors as using pre-trained feature extractors is a standard approach in NLP for almost any task and results in huge improvement in performance. However, we are not aware of any existing CL methods for NLP tasks use pseudo-replay. Some recent works in computer vision have used pre-trained feature extractors. For example, Hayes et al. (2019) and Hayes & Kanan (2020) used a pre-trained feature extractor for online continual learning with streaming data. Our work is not about online continual learning. PCL (Hu et al., 2021) uses a fixed feature extractor to build individual classifier for each class under the batch training scenario. None of these methods use pseudo-replay as our approach PLS. We will compare it with the proposed method PLS in the experiment section.

## 3 Proposed PLS Technique

As mentioned earlier, the proposed method PLS is a pseudo-replay method for class-incremental learning (CIL). It uses a pre-trained feature extractor, i.e., CLIP in our experiment. Unlike most previous pseudo-replay methods, which learn a data generator to generate pseudo raw data to deal with catastrophic forgetting (CF), PLS only samples feature vectors from the final layer of the pre-trained feature extractor and use them to represent features of previous task data in training the new task and to deal with CF.

Briefly, PLS works as follows. It assumes that the feature vectors obtained from the pre-trained feature extractor are drawn from a multivariate Gaussian distribution. In addition to training the model on a task, the mean $\boldsymbol{\mu}_c$ and covariance $\boldsymbol{\Sigma}_c$ of the feature vectors of each class $c$ in the task can be estimated. Consequently, these estimates can be used to sample feature vectors of the class in pseudo-replay. However, saving the full covariance matrix for each class is very memory inefficient. For example, a feature extractor with $d$ dimensional output requires $d \times d$ entries for the covariance matrix of each class. The total entries required for learning a dataset with $|C|$ classes is $d \times d \times |C|$. In our experiment, $d$ is 512 and $|C|$ is 1000 for the largest dataset (i.e. 262M entries). This may not be feasible for appliances with a small on-device memory. The proposed method PLS solves it by saving only a few eigenvalue-eigenvector pairs with limited memory budget. These eigenpairs are directly used to sample feature vectors. Below, we present the technique and describe a few variations.

In our experiment in Section 4.2, we save only 5 eigenpairs and 1 sample mean per class on average. The system already works very well.

### 3.1 APPROXIMATING COVARIANCE MATRIX

Given the data of a task, we compute the sample mean and sample covariance matrix of each class via *singular value decomposition* (SVD). For simplicity, we omit the task-id $t$ and class-id $c$ unless necessary since this discussion is about computing the statistics for each class in a task data.

Suppose we receive $n_c$ feature vectors $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_{n_c}]$ of class $c$ from a task after feature extractor. We compute the sample mean $\boldsymbol{\mu}$ as

$$\boldsymbol{\mu} = \sum_{i=1}^{n_c} \boldsymbol{x}_i / n_c \tag{1}$$

and the SVD of centered data

$$\boldsymbol{U}\tilde{\boldsymbol{\Lambda}}\boldsymbol{V}^T \overset{\text{svd}}{=} [\boldsymbol{X} - \boldsymbol{\mu}] \tag{2}$$

where $T$ means transpose. The columns of $\boldsymbol{U}$ and diagonals of $\tilde{\boldsymbol{\Lambda}}^2$ are eigenvectors and eigenvalues of unnormalized sample covariance matrix $(n_c - 1)\boldsymbol{\Sigma}$ since $[\boldsymbol{X} - \boldsymbol{\mu}][\boldsymbol{X} - \boldsymbol{\mu}]^T = \boldsymbol{U}\tilde{\boldsymbol{\Lambda}}^2\boldsymbol{U}^T$. Note that the total variance is explained by the sum of eigenvalues, $(n_c - 1)\sum_{i=1}^{d} \sigma_{ii}^2 = \sum_{i=1}^{d} \tilde{\lambda}_i^2$, where $\sigma_{ii}^2$ is the $i^{th}$ diagonal element of $\boldsymbol{\Sigma}$ and $\tilde{\lambda}_1^2 \geq \cdots \geq \tilde{\lambda}_d^2$ are the diaginal elements of $\tilde{\boldsymbol{\Lambda}}^2$. Denote the diagonal matrix with eigenvalues of normalized covariance by

$$\boldsymbol{\Lambda}^2 = \tilde{\boldsymbol{\Lambda}}^2/(n_c - 1) \tag{3}$$

We keep the first $k$ eigenpairs and discard the rest $d - k$ pairs that least explain the total variance. Denote the reduced size matrices by the same notations $\boldsymbol{U}$ and $\boldsymbol{\Lambda}^2$. We save the sample mean $\boldsymbol{\mu}$ and $(\boldsymbol{U}, \boldsymbol{\Lambda}^2)$ in memory $\mathcal{M}$ and discard $\boldsymbol{V}$ since it is not necessary for approximating the covariance.

In the actual implementation and for fair comparison, we follow the existing replay-based approach by having a fixed memory budget and adjust $k$ to fit the memory buffer. That is, $k$ varies according to the memory budget available at the time of learning the class $c$. As more tasks are learned, $k$ becomes smaller. After a new task is learned, the number of saved eigenpairs $k$ for old classes is reduced to accommodate the eigenpairs of the new class $c$.

In the following, we discuss how to draw pseudo feature vectors from the sample mean and eigenpairs.

### 3.2 PSEUDO REPLAY FROM THE FEATURE SPACE

Suppose we have obtained the sample mean $\boldsymbol{\mu}_c$ and $k$ eigenpairs $(\boldsymbol{U}_c, \boldsymbol{\Lambda}_c^2)$ for each class $c$, where $\boldsymbol{U}_c \in \mathbb{R}^{d \times k}$ and $\boldsymbol{\Lambda}_c \in \mathbb{R}^{k \times k}$. We draw random features of class $c$ from the Gaussian distribution with mean $\boldsymbol{\mu}_c$ and covariance $\boldsymbol{\Sigma}_c$ for pseudo-replay. Given a random sample $\boldsymbol{z} \in \mathbb{R}^k$ from the standard normal distribution, we draw a random feature and apply linear transformation (Gentle, 2009) as

$$\boldsymbol{x} = \boldsymbol{\mu}_c + \boldsymbol{U}_c\boldsymbol{\Lambda}_c\boldsymbol{z} \tag{4}$$

Note that we do not need to explicitly compute the covariance matrix $\boldsymbol{\Sigma}_c$ to draw sample feature vectors.

In training, we train the classifier with the current batch and generate feature vectors of classes of previous tasks in each training iteration for replay. Algorithm 1 describes the training process, where we have put comments with the symbol "//".

### 3.3 TYPES OF COVARIANCE

We have discussed our approach based on the estimation of sample covariance matrix for each class. In the following discussion, we propose another 2 estimations with more efficient memory usage in modeling the latent feature space. In Section 4.3, we show the performance difference between the estimations and discuss suitability of each method in various scenarios.

One method is to share the sample covariance across the classes within the same task. Given the eigenpairs $\{(\boldsymbol{U}_{c_j}, \boldsymbol{\Lambda}^2_{c_j})\}_{j=1}^r$ of class $c_1, \cdots, c_r$ of task $t$, we combine the sample covariance

$$\boldsymbol{\Sigma}_t = \frac{1}{r}\sum_{j=1}^{r}\sum_{i=1}^{k}\lambda^2_{i,c_j}\boldsymbol{u}_{i,c_j}\boldsymbol{u}^T_{i,c_j} \tag{5}$$

where $\lambda^2_{i,c_j}$ and $\boldsymbol{u}_{i,c_j}$ are the $i^{th}$ eigenvalue and eigenvector of class $c_j$ of task $t$, respectively. We save the leading $k$ eigenpairs of the shared covariance matrix $\boldsymbol{\Sigma}_t$ after singular value decomposition, and discard $\boldsymbol{\Sigma}_t$ to save memory as it is not necessary for sampling. We adjust the number of saved eigenpairs of the previous tasks accordingly and draw sample features of class $c_j$ from $N(\boldsymbol{\mu}_{c_j}, \boldsymbol{\Sigma}_t)$ by the sampling method discussed in Section 3.2.

Another approach is to incrementally compute a single shared covariance for all classes of the previous tasks. Given $k$ eigenpairs $\{(\boldsymbol{u}_i, \lambda^2_i)\}_{i=1}^k$ of the previously shared covariance and the eigenpairs $\{(\boldsymbol{u}_{i,c}, \lambda^2_{i,c})\}_{i=1}^k$ of a new class $c$, approximate the sample covariance by

**Algorithm 1** PLS training algorithm ($\boldsymbol{\Sigma}_c$ version)

**Require:** feature extractor $f$, classifier $g$, memory $\mathcal{M}$, sequence of tasks $\boldsymbol{D} = \{\boldsymbol{D}_t\}_{t=1}$

// CL starts

**for** each task data $\boldsymbol{D}_t \in \boldsymbol{D}$ **do**
  Obtain features $\boldsymbol{X}_t \leftarrow f(\boldsymbol{D}_t)$
  // Compute statistics
  **for all** class data $\boldsymbol{X}_c$ of $\boldsymbol{X}_t$ **do**
    Compute $\boldsymbol{\mu}_c$ using Eq. 1
    Compute $\boldsymbol{U}_c$, $\boldsymbol{\Lambda}^2_c$ using Eq. 2, Eq. 3
    $\mathcal{M} \leftarrow Update(\mathcal{M}(\boldsymbol{\mu}_c, \boldsymbol{U}_c, \boldsymbol{\Lambda}^2_c))$
  **end for**
  // Train a task
  **for** $(\boldsymbol{X}_{t,i}, \boldsymbol{y})$ in $\boldsymbol{X}_t$, until converge **do**
    $\boldsymbol{X}_s$ = $sample(\mathcal{M})$ using Eq. 4
    minimize CE$(g(\boldsymbol{X}_{t,i} \cup \boldsymbol{X}_s), \boldsymbol{y})$
  **end for**
**end for**

$$\boldsymbol{\Sigma} = \frac{1}{|C|+1}\sum_{i=1}^{k}\lambda_i\boldsymbol{u}_i\boldsymbol{u}^T_i + \lambda_{i,c}\boldsymbol{u}_{i,c}\boldsymbol{u}^T_{i,c} \tag{6}$$

where $|C|$ is the number of previous classes. We take the leading $k$ eigenpairs and draw sample features as discussed in Section 3.2. This approach is most memory efficient in learning scenarios where the number of classes or tasks is larger.

We have discussed the methods for computing the sample mean and covariance in the traditional batch learning scenario where a task data are available and loaded on the machine in the beginning of training. However, we can also compute the statistics incrementally (see below).

### 3.4 Incremental Approximation of the Statistics

The above methods need the full data of each task to compute the eigenpairs. When the training data is very large and/or the device's memory size is small, the above approaches may have some difficulty. Here we also propose an incremental version of the approach.

We take ideas from the algorithms developed for incremental principal component analysis (iPCA) (Levy & Lindenbaum, 1998; Ross et al., 2008). In iPCA, we incrementally update the singular value decomposition of the entire data from the current batch without having access to the previous data. We explain how to update the statistics of each class, but the same method can be applied to the shared covariance $\boldsymbol{\Sigma}_t$ and $\boldsymbol{\Sigma}$. We omit the task-id $t$ and class label $c$ unless necessary.

Suppose we receive a batch of $m$ feature vectors $\boldsymbol{X}_{\text{new}} = [\boldsymbol{x}_{n+1} \cdots \boldsymbol{x}_{n+m}]$ of class $c$, where $n$ is the number of seen samples of class $c$. Given the previously computed sample mean $\boldsymbol{\mu}$, we update it as

$$\tilde{\boldsymbol{\mu}} = \frac{1}{n+m}(n\boldsymbol{\mu} + m\boldsymbol{\mu}_{\text{new}}) \tag{7}$$

where $\boldsymbol{\mu}_{\text{new}}$ is the sample mean computed on the current samples $\boldsymbol{X}_{\text{new}}$. This is the sample mean for class $c$ on the data including previous and current samples.

Given the previous sample mean $\boldsymbol{\mu}$ obtained in Eq. 1, and $k$ leading eigenpairs $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times k}$ and $\boldsymbol{U} \in \mathbb{R}^{d\times k}$ obtained in Eq. 2 and Eq. 3, construct a block matrix $[\sqrt{n-1}\boldsymbol{U}\boldsymbol{\Lambda} \quad \hat{\boldsymbol{B}}]$, where $\hat{\boldsymbol{B}} = [\boldsymbol{X}_{\text{new}} - \boldsymbol{\mu}_{\text{new}} \quad \sqrt{nm/(n+m)}(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})] \in \mathbb{R}^{d\times m+1}$ is the centered data with one additional column. We compute SVD of the block matrix

$$\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Lambda}}\tilde{\boldsymbol{V}}^T \overset{\text{svd}}{=} [\sqrt{n-1}\boldsymbol{U}\boldsymbol{\Lambda} \quad \hat{\boldsymbol{B}}] \tag{8}$$

As discussed previously, we keep $k$ leading pairs of singular values and orthonormal vectors and discard $\tilde{\boldsymbol{V}}$ to save memory. The columns of $\tilde{\boldsymbol{U}}$ and diagonals of $\tilde{\boldsymbol{\Lambda}}^2$ are $k$ eigenvectors and eigenvalues of unnormalized sample covariance matrix $(n+m-1)\boldsymbol{\Sigma}$ with some constant (refer to Appendix C for details). We sample pseudo-features with the new $k$ eigenpairs and sample mean as Section 3.2.

# 4 EXPERIMENTS

**Evaluation Datasets:** Three image classification benchmark datasets are used in our experiments.

(1). **CIFAR-10** (Krizhevsky & Hinton, 2009)[1]: This dataset consists of 60,000 32x32 color images of 10 classes. 50,000 images are used for training and 10,000 are used for testing.

(2). **CIFAR-100** (Krizhevsky & Hinton, 2009)[2]: This dataset consists of 60,000 samples of 32x32 color images of 100 classes with 500 and 100 images per class for training and testing, respectively.

(3). **ImageNet-1000** (Russakovsky et al., 2015)[3]: This dataset consists of 1,281,167 natural images of 1,000 classes for training. Since the test data do not have labels, we use the validation data as in (Rebuffi et al., 2017) for testing.

**Baseline Systems**: We consider 10 class-incremental learning (CIL) baselines of different types as the proposed PLS is a CIL system. For CIL approaches that do not save or generate any samples from previous tasks, we compare against **OWM** (Zeng et al., 2019). For replay-based methods, we compare with **LwF** (Li & Hoiem, 2017) improved by (Liu et al., 2020a), **iCaRL** (Rebuffi et al., 2017), **A-GEM** (Chaudhry et al., 2018), **BiC** (Wu et al., 2019), **DER++** (Buzzega et al., 2020), and **HAL** (Chaudhry et al., 2021). For pseudo-replay methods, we compare with the latest systems **EEC** (Ayub & Wagner, 2021) and **PASS** (Zhu et al., 2021). For methods that use a pre-trained feature extractor, we compare with the latest system **PCL** (Hu et al., 2021).

## 4.1 TRAINING DETAILS

We run public codes for the baselines (links are in Appendix E). For all baselines and our system PLS, we use CLIP (Radford et al., 2021) pre-trained network except for PCL (Hu et al., 2021) as PCL works better using its own pre-trained network trained using the ImageNet data. The other baselines work markedly better with the fixed CLIP feature extractor. For PLS, we fine-tune the linear classifier on top of the fixed CLIP (512 dimensions) for each task. For each dataset of $|C|$ classes, we conduct experiments using two different memory sizes $|\mathcal{M}| = (5+1)|C|$ and $|\mathcal{M}| = (10+1)|C|$, i.e., saving $5$ eigenpairs and $1$ sample mean and $10$ eigenpairs and $1$ sample mean per class when the last task is learned, respectively. This is equivalent to saving $6|C|$ and $11|C|$ feature vectors for the replay-based baselines. For PLS and pseudo-replay baselines, we sample feature vectors as many as the batch size following the pseudo-replay methods in (Ayub & Wagner, 2021; Zhu et al., 2021). We use 10% of training data for hyper-parameter search to select a good set of hyper-parameters for all methods. Additional training details are as follow.

For CIFAR-10, we use the memory size of $|\mathcal{M}| = 60$ or $110$ based on the memory size allocation scheme described above. We split 10 classes into 5 tasks, where each task has 2 consecutive classes. We train our system for 1 epochs with batch size of 16. We use Adam optimization (Kingma & Ba, 2014) with learning rate 0.001.

For CIFAR-100, we use the memory size $|\mathcal{M}| = 600$ or $1100$, again based on the memory size allocation scheme described above. We conduct experiments for 10 tasks and 20 tasks. We split the classes in consecutive order. For 10 and 20 tasks, we train for 10 and 5 epochs, respectively. For both tasks, we use batch size of 16 and Adam optimization with learning rate 0.1.

For ImageNet-1000, we set the memory size as $|\mathcal{M}| = 6000$ or $11000$ also based on the memory size allocation scheme described above. We split 1000 classes into 10 tasks with 100 classes per task. We train our system for 1 epoch with batch size of 256, and use Adam optimizer with learning rate 0.1.

[1]https://www.cs.toronto.edu/ kriz/cifar.html

[2]https://www.cs.toronto.edu/ kriz/cifar.html

[3]https://image-net.org

### 4.2 TEST RESULTS AND COMPARISONS

We compare our method against the baselines by two metrics: average classification accuracy and average forgetting rate. We compute the average classification accuracy of a network on all the classes learned after training the last task. The average forgetting rate is defined as $\mathcal{F}^t = \sum_{j=1}^{t-1}(A_j^{\text{init}} - A_j^t)/(t-1)$ where $A_j^{\text{init}}$ is the classification accuracy of task $j^{th}$ data right after learning it and $A_j^t$ is the accuracy of $j^{th}$ task after learning the final task $t$. We report the average result over 3 runs with different random seed. In Appendix A, we also report *average incremental accuracy* of the systems.

**Using Pre-Trained Feature Extractor by PSL and baselines**. Table 1 shows that PLS and and its incremental version, iPLS, consistently outperform the baselines on all datasets by large margins in both memory sizes. The Diff. row shows the performance difference between our approach and the best baseline. PLS and iPLS performs similarly. The average column shows that on average, PLS improves the best baseline by 6.5%. The smallest performance gap, 2.3%, between PSL and the best baseline occurs in CIFAR10-5T, where PLS achieves 93.0% accuracy. The gap becomes greater for more challenging datasets. With a larger memory (i.e., $|\mathcal{M}| = (10+1)|C|$), PLS achieves 72.3% and 67.0% and the best baseline methods do 65.4% and 60.3% in CIFAR100-20T and ImageNet-10T, respectively.

The pseudo-replay baseline EEC displays strong performance in relatively easier datasets such as CIFAR10 and CIFAR100. However, when it is trained for the challenging dataset (i.e. ImageNet), its accuracy drops significantly as training a generative model is difficult in continual learning as noted in (Lesort et al., 2019). In fact, we could not get EEC to work with 1000 classes. The reported results in the table are for only 50 classes. PLS, on the other hand, does not need to train a generative model as it directly samples features from multivariate Gaussian distributions. This allows PLS and iPLS to have consistent performances throughout the data settings. Note that OWM, EEC, PASS, and PCL have only one result for each data setting as they do not use a memory buffer. For PCL, we use its own pre-trained feature extractor as it works poorly with CLIP. As PCL mainly works in the one class per task setting, it does not work well when there are many classes in a task. We thus copied its results of one-class per task, which covers the case of multiple classes per task. For ImageNet, there is no result because PCL's feature extractor is trained using ImageNet supervised data.

**Without using Pre-trained Feature Extractor**. We run the original code of baselines without using the CLIP pre-trained feature extractor to motivate the benefits of using pre-trained network. We use ResNet-18 (He et al., 2016) as the backbone architecture and train the network using memory size 2000 on CIFAR100-10T (10 tasks). The results are reported on the rightmost column in Table 1. All the baselines perform significantly poorer without the pre-trained feature extractor. There are no results for PCL, PLS, and iPLS here as they were designed for use with pre-trained feature extractor.

**Forgetting Rate**. We compare the forgetting rate using CIFAR10-5T and CIFAR100-10T with the smaller memory size (i.e. $|\mathcal{M}| = 5|C|+1$). Figure 1 shows that our methods PLS and iPLS have the smallest forgetting rates on CIFAR10-5T. Although iCaRL and BiC suffer less from forgetting than PLS and iPLS on CIFAR100-10T, their accuracies are 61.4 and 66.9, respectively, which are much lower than that of PLS (71.2%). We provide forgetting rates on the other datasets in Appendix B.

**Comparison between PLS and iPLS**. We have proposed PLS and its incremental version, iPLS, for different learning scenarios. In Table 1, the two methods show similar performance across the datasets. However, PLS is slightly faster than iPLS as PLS obtains singular value decomposition once in the beginning of training, whereas iPLS computes it at each batch. Using NVIDIA RTX 3090 under the same experiment setting, we measure the training time spent on CIFAR100-10T. PLS took $56.6 \pm 0.53$ minutes while iPLS took $59.8 \pm 1.56$ minutes.

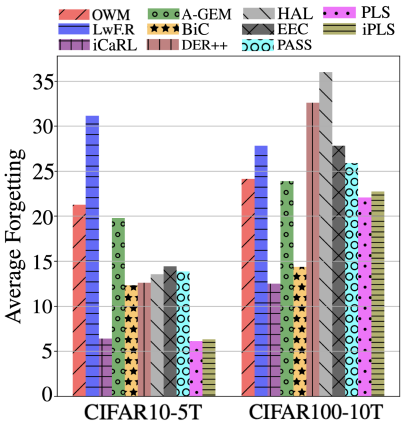



Figure 1: Average forgetting rate (%). The lower the rate, the better the method is.

Table 1: Average classification accuracy after the final task. '-XT' means X tasks. We report the average incremental accuracy in Appendix A. The column Average indicates the average of results of each method over the datasets and memory budgets. The row Diff. gives the difference in accuracy between our method and the best baseline in each column. The results on the rightmost column are the accuracy without fixed pre-trained feature extractor. We highlight the best result in each column in bold. EEC† indicates that we follow the original paper for ImageNet experiment, where 50 classes are split into 5 tasks because we are unable to run the official code for 1000 classes on our system. With CLIP pre-trained feature extractor, the *incremental average accuracy* improves to 42% which is greatly higher than 35.2% in the original paper. Cells without results are explained in the text.

| | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | ImageNet-10T | | Average | CIFAR100-10T |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|\mathcal{M}\| =$ | 60 | 110 | 600 | 1100 | 600 | 1100 | 6000 | 11000 | | 2000 |
| OWM | 79.4±3.38 | | 56.9±2.23 | | 56.6±1.67 | | 17.1±2.67 | | 52.5 | 29.0±0.7 |
| LwF.R | 72.1±0.11 | 71.2±1.45 | 61.1±1.94 | 61.6±2.29 | 65.6±0.20 | 62.6±2.63 | 54.4±0.02 | 54.3±0.05 | 62.9 | 45.3±0.8 |
| iCaRL | 89.4±0.00 | 89.2±0.00 | 61.4±0.20 | 63.6±0.05 | 61.4±0.23 | 63.8±0.08 | 58.7±0.00 | 60.3±0.01 | 68.5 | 51.4±1.0 |
| A-GEM | 82.0±1.42 | 81.2±1.22 | 55.6±0.06 | 61.2±0.04 | 55.7±0.05 | 61.1±0.17 | 21.4±0.27 | 21.0±0.18 | 54.9 | 9.4±0.2 |
| BiC | 78.0±1.94 | 82.5±3.71 | 66.9±2.70 | 68.8±1.60 | 62.2±4.58 | 64.2±1.18 | 60.5±0.25 | 60.3±1.17 | 67.9 | 51.3±0.6 |
| DER++ | 88.6±0.64 | 90.8±0.46 | 63.4±0.40 | 66.5±0.30 | 63.0±1.74 | 65.4±1.92 | 56.6±0.70 | 59.4±0.37 | 69.2 | 55.3±0.1 |
| HAL | 86.2±0.88 | 88.1±0.73 | 53.2±0.13 | 57.4±0.64 | 53.9±0.81 | 56.9±0.31 | 48.4±0.22 | 52.9±0.31 | 62.1 | 11.9±0.7 |
| EEC† | 86.8±0.34 | | 64.2±0.46 | | 62.5±0.23 | | 18.1±0.06 | | 57.9 | 10.9±0.4 |
| PASS | 86.8±0.65 | | 62.0±0.97 | | 63.1±1.11 | | 59.0±0.34 | | 67.7 | 35.8±0.3 |
| PCL | 84.9 | | 63.6 | | 63.6 | | | | | |
| PLS | **93.0**±0.12 | **93.1**±0.08 | **71.2**±0.38 | 72.5±1.00 | **71.7**±0.28 | 72.0±1.18 | **65.6**±0.22 | 66.8±0.12 | **75.7** | |
| iPLS | **93.0**±0.09 | 93.0±0.08 | 70.6±0.42 | **72.8**±0.56 | 70.8±0.60 | **72.3**±0.83 | 65.5±0.17 | **67.0**±0.38 | 75.6 | |
| Diff. | +3.6 | +2.3 | +4.3 | +4 | +6.1 | +6.9 | +5.1 | +6.7 | +6.5 | |

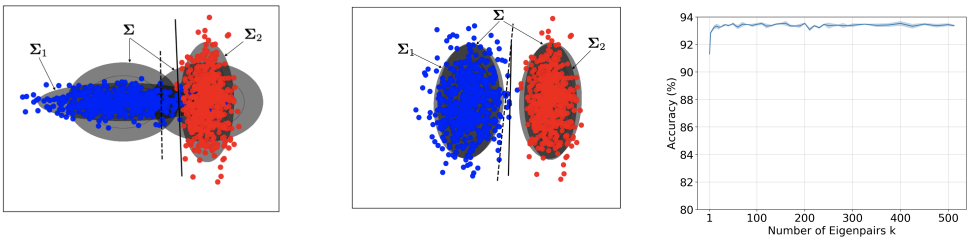
Table 2: Comparison of proposed covariances. $\mathbf{\Sigma}_c$ is the covariance used in Table 1. $\mathbf{\Sigma}_t$ is the shared covariance across classes within a task. $\mathbf{\Sigma}$ is the shared covariance across all classes. Each accuracy is the average classification accuracy after the final task is learned. The accuracy of $\mathbf{\Sigma}$ in ImageNet with $|\mathcal{M}| = 11000$ is copied from $|\mathcal{M}| = 6000$ as its actual memory usage (1512) is the same.

| | | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | ImageNet-10T | | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\mathcal{M} =$ | 60 | 110 | 600 | 1100 | 600 | 1100 | 6000 | 11000 | |
| PLS | $\mathbf{\Sigma}_c$ | 93.0±0.12 | 93.1±0.08 | 71.2±0.38 | 72.5±1.00 | 71.7±0.28 | 72.0±1.18 | 65.6±0.22 | 66.8±0.12 | 75.7 |
| | $\mathbf{\Sigma}_t$ | 92.6±0.02 | 92.3±0.16 | 71.4±0.62 | 71.8±0.11 | 72.2±0.49 | 72.2±0.11 | 66.1±0.03 | 66.2±0.18 | 75.6 |
| | $\mathbf{\Sigma}$ | 92.4±0.23 | 92.5±0.15 | 71.1±0.42 | 71.1±0.71 | 70.4±0.71 | 69.6±0.41 | 65.2±0.13 | 65.2±0.13 | 74.7 |
| iPLS | $\mathbf{\Sigma}_c$ | 93.0±0.09 | 93.0±0.08 | 70.6±0.42 | 72.8±0.56 | 70.8±0.60 | 72.3±0.83 | 65.5±0.17 | 67.0±0.38 | 75.6 |
| | $\mathbf{\Sigma}_t$ | 90.8±0.07 | 91.7±0.02 | 71.9±0.76 | 72.3±0.96 | 69.5±0.95 | 71.1±0.62 | 66.3±0.09 | 66.5±0.39 | 75.0 |
| | $\mathbf{\Sigma}$ | 91.7±0.31 | 92.2±0.12 | 71.9±0.30 | 72.0±0.71 | 71.2±0.32 | 71.1±0.71 | 65.9±0.09 | 65.9±0.09 | 75.2 |

### 4.3 Analysis and Ablation Study

**Comparison of Covariances**. We have proposed three methods to approximate sample covariances, where each method assumes different property in the latent space. We compare the performance of proposed methods in Table 2. In terms of *performance*, the three methods are similar. $\mathbf{\Sigma}_c$ outperforms the other methods on average, but the difference between the second best method is only $0.1\%$ in PLS and $0.4\%$ in iPLS. In terms of *memory usage*, some methods consume less than the others. We fix the memory size and reduce the number of eigenpairs after each task to accommodate the statistics of new classes. When the number of classes $|C|$ is larger than $d$ (feature dimension), $\mathbf{\Sigma}$ consumes less memory than $\mathbf{\Sigma}_c$ as $\mathbf{\Sigma}_c$ requires at least $|C|$ eigenpairs to approximate each covariance while $\mathbf{\Sigma}$ requires at most $d$ eigenpair. For instance, in ImageNet, $\mathbf{\Sigma}_c$ uses all the memory $6000$ (i.e. $5 \times 1000 + 1000$, where $5$ is the number of eigenpairs saved per class in learning the last task and $1000$ is the number of classes). On the other hand, $\mathbf{\Sigma}$ uses only $1512$ (i.e. $512 + 1000$, where $512$ is the feature dimension $d$ and $1000$ is the number of classes). For a similar reason, $\mathbf{\Sigma}_t$ consumes more memory than $\mathbf{\Sigma}$ when the number of tasks is larger than $d$.

**Effect of Shared Covariance Matrix**. We have proposed two types of covariances: 1) individual covariance for each class $\mathbf{\Sigma}_c$ and 2) a shared covariance $\mathbf{\Sigma}_t$ or $\mathbf{\Sigma}$. $\mathbf{\Sigma}_t$ assumes that every class in a task shares the same covariance matrix while $\mathbf{\Sigma}$ assumes a single shared covariance throughout the training process. These assumptions may not be true in some applications. Thus, despite PLS/iPSL shows similar performance in the datasets regardless of covariance types in Table 2, it can perform differently depending on the covariance type. We construct a synthetic dataset of two classes that follow distributions $N(\boldsymbol{\mu}_1, \mathbf{\Sigma}_1)$ and $N(\boldsymbol{\mu}_2, \mathbf{\Sigma}_2)$ to explain PLS performance when one type of




(a) (b) (c)

Figure 2: (a) and (b) show decision boundaries for two classes (*best viewed in color*). The solid line and dashed line are decision boundaries found using individual covariance per class $\mathbf{\Sigma}_c$ and shared covariance $\mathbf{\Sigma}$, respectively. (a) shows that the decision line by shared covariance (i.e. dashed line) is severely shifted when the covariances construct different shapes whereas the solid line separates the clusters properly. (b) shows that the two decision lines are equally good when class covariances form a similar shape. (c) shows accuracy change over the number of saved eigenpairs per class.

covariance is more advantageous than the others. In Figure 2(a), we display the datasets and overlay the contours of their covariances $\mathbf{\Sigma}_1$, $\mathbf{\Sigma}_2$ and the shared covariance $\mathbf{\Sigma}$. The decision boundary (solid line) found by PLS using individual covariances properly separates the two classes. However, PLS finds a boundary (dashed line) shifted toward the blue class when $\mathbf{\Sigma}$ is used as $\mathbf{\Sigma}$ does not reflect the true distributions. The accuracy by the solid line is 97.3% while that of the dashed line is 93.7%. In Figure 2(b), the two classes follow similar covariances (i.e. $\mathbf{\Sigma}_1 \simeq \mathbf{\Sigma}_2$). PLS finds decision boundaries that properly separate the two classes, in which the solid line achieves 100% accuracy and the dashed line achieves 99.7%.

**Number of Eigenpairs and Performance**. We study the number of eigenpairs and its impact on classification accuracy. In the preceding experiments, we fix the memory size and change the number of eigenpairs ($k$) in memory after each task is learned like in the replay-based baselines. In this experiment, we fix the number of pairs per class and study how accuracy changes over $k$ in CIFAR10-5T. We first train PLS without saving any eigenpairs (i.e. $k = 0$), but sampling from $N(\boldsymbol{\mu}_c, \boldsymbol{I})$ with unit covariance matrix $\boldsymbol{I}$ whose diagonal entries are 1 and the rest are 0. By simply distinguishing the sample means, we obtain 47.1% accuracy. Figure 2(c) shows the classification accuracy as a function of $k = 1, \cdots, 512$. PLS greatly improves and already achieves 91.3% accuracy with 1 eigenpair per class. This leading pair explains 9.8% of total variance. The first 5 eigenpairs explain 30.7% of the total variance and the accuracy stabilizes at 93.1%. With full eigenpairs (i.e,. $k = 512$), the accuracy is 93.37%. This implies that PLS requires little memory to perform very well.

**PLS and Zero-Shot Property of CLIP**. CLIP has been shown to work well in the zero-shot setting (Radford et al., 2021). We show that PLS achieves much higher accuracy on all the datasets than CLIP zero-shot accuracy, which demonstrates that the strong performance of PLS is not due to the zero-shot property of CLIP. The details are given in Appendix D.

## 5 Conclusion

This paper proposes a simple and highly effective method (called PLS) working with a pre-trained feature extractor to perform class-incremental learning (CIL) and to deal with *catastrophic forgetting* (CF). PLS is a pseudo-replay method, but it is very different from existing pseudo-replay methods that train a data generator to generate pseudo replay data. Instead, PLS works in the latent feature space by sampling feature vectors of previous classes. As explained in the introduction section, this makes the system suitable for applications that have data privacy concerns which prohibit the system from building data generators. Experimental results show that PLS outperforms both the classic and state-of-the-art recent baselines by a large margin even when both baselines and PLS leverage the pre-trained CLIP feature extractor.

## REFERENCES


Davide Abati, Jakub Tomczak, Tijmen Blankevoort, Simone Calderara, Rita Cucchiara, and Ehteshami Bejnordi. Conditional channel gated networks for task-aware continual learning. In *CVPR*, pp. 3931–3940, 2020.

Hongjoon Ahn, Sungmin Cha, Donggyu Lee, and Taesup Moon. Uncertainty-based continual learning with adaptive regularization. In *NeurIPS*, 2019.

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. *CoRR, abs/1611.06194*, 2, 2016.

Ali Ayub and Alan Wagner. {EEC}: Learning to encode and regenerate images for continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=lWaz5a9lcFU.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and SIMONE CALDERARA. Dark experience for general continual learning: a strong, simple baseline. In *NeurIPS*, 2020.

Raffaello Camoriano, Giulia Pasquale, Carlo Ciliberto, Lorenzo Natale, Lorenzo Rosasco, and Giorgio Metta. Incremental robot learning of new objects with fixed update time. In *ICRA*, 2017.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, pp. 233–248, 2018.

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

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019.

Arslan Chaudhry, Naeemullah Khan, Puneet K. Dokania, and Philip H. S. Torr. Continual learning in low-rank orthogonal subspaces, 2020.

Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6993–7001, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16861.

Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. In *NeurIPS*, 2019.

Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *CVPR*, 2019.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

James E Gentle. *Computational statistics*, volume 308. Springer, 2009.

Alexander Gepperth and Cem Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016.

Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop on Continual Learning in Computer Vision*, pp. 220–221, 2020.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. *arXiv preprint arXiv:1910.02509*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Xu He and Herbert Jaeger. Overcoming Catastrophic Interference using Conceptor-Aided Backpropagation. In *ICLR*, 2018.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *CVPR*, pp. 831–839, 2019.

Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. In *ICLR*, 2019.

Wenpeng Hu, Qi Qin, Mengyu Wang, Jinwen Ma, and Bing Liu. Continual learning by using information of each class holistically. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7797–7805, 2021.

Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning. In *NeurIPS*, volume 32, 2019.

Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016.

Nitin Kamra, Umang Gupta, and Yan Liu. Deep Generative Dual Memory Network for Continual Learning. *arXiv preprint arXiv:1710.10368*, 2017.

Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. In *NeurIPS*, 2020.

Ronald Kemker and Christopher Kanan. FearNet: Brain-Inspired Model for Incremental Learning. In *ICLR*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and Others. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical Report TR-2009, University of Toronto, Toronto.*, 2009.

Kibok Lee, Kimin Lee, Jinwoo Shin, and Honglak Lee. Overcoming catastrophic forgetting with unlabeled data in the wild. In *CVPR*, 2019.

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, pp. 4655–4665, 2017.

Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat. Generative models from the perspective of continual learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Avraham Levy and Michael Lindenbaum. Sequential karhunen-loeve basis extraction and its application to images. In *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269)*, volume 2, pp. 456–460. IEEE, 1998.

Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*, 2019.

Zhizhong Li and Derek Hoiem. Learning Without Forgetting. In *ECCV*, pp. 614–629. Springer, 2016.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *CVPR*, 2020a.

Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *CVPR*, 2021.

Yu Liu, Sarah Parisot, Gregory Slabaugh, Xu Jia, Ales Leonardis, and Tinne Tuytelaars. More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning. In *ECCV*, pp. 699–716. Springer International Publishing, 2020b. doi: 10.1007/978-3-030-58574-7_42. URL https://doi.org/10.1007/978-3-030-58574-7_42.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In *NeurIPS*, pp. 6470–6479, 2017.

Arun Mallya and Svetlana Lazebnik. PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning. *arXiv preprint arXiv:1711.05769*, 2017.

Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *arXiv preprint arXiv:1802.01569*, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Fei Mi, Lingjing Kong, Tao Lin, Kaicheng Yu, and Boi Faltings. Generalized class incremental learning. In *CVPR*, 2020.

Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *CVPR*, pp. 11321–11329, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, Ling Shao, and Ming-Hsuan Yang. An adaptive random path selection approach for incremental learning, 2020.

Amal Rannen Ep Triki, Rahaf Aljundi, Matthew Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *ICCV*, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, pp. 5533–5542, 2017.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *NeurIPS*, 2018.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. In *NeurIPS*, 2019.

David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International journal of computer vision*, 77(1):125–141, 2008.

Mohammad Rostami, Soheil Kolouri, and Praveen K. Pilly. Complementary learning for overcoming catastrophic forgetting using experience replay. In *IJCAI*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

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

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

Ari Seff, Alex Beatson, Daniel Suo, and Han Liu. Continual learning in generative adversarial nets. *arXiv preprint arXiv:1705.08395*, 2017.

Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, pp. 2994–3003, 2017.

Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. *NeurIPS*, 2020.

Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *ICLR*, 2020.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *NeurIPS*, volume 33, pp. 15173–15184. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/ad1f8bb9b51f023cdc80cf94bb615aa9-Paper.pdf.

Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. In *NeurIPS*, 2018.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, 2019.

Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *NeurIPS*, 2018.

Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *ICLR*, 2020. URL https://openreview.net/forum?id=r1gdj2EKPB.

Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continuous learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 2019.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pp. 3987–3995, 2017.

Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

Bowen Zhao, Xi Xiao, Guojun Gan, Bin Zhang, and Shu-Tao Xia. Maintaining discrimination and fairness in class incremental learning. In *CVPR*, pp. 13208–13217, 2020.

Fei Zhu, Xu-Yao Zhang, Chuang Wang, Fei Yin, and Cheng-Lin Liu. Prototype augmentation and self-supervision for incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5871–5880, June 2021.

## A AVERAGE INCREMENTAL ACCURACY

In the main paper, we report the average classification accuracy $A_t$ after the final task $t$. Here, we report the *average incremental accuracy* $\mathcal{A} = \sum_{k=1}^{t} A_k/t$, where $A_k$ is the average classification accuracy over all the learned tasks until task $k$. Average incremental accuracy measures the model performance throughout the learning process. Table 3 shows that PLS and iPLS outperform the baselines by large margin, especially in more challenging datasets such as CIFAR100 and ImageNet. Note that PCL is not included here because we cannot run the system as the official code does not provide the pre-trained feature extractor. EEC on CIFAR100-10T without pre-trained CLIP is lower (31.3%) than the original paper (65.6%, which is still much lower than with CLIP (76.2%)) as we cannot reproduce the result from the official code despite extensive hyper-parameter search.

Table 3: Average incremental learning. The best result in each column is highlighted in bold.

| | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | ImageNet-10T | | Average | CIFAR100-10T |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|\mathcal{M}\| =$ | 60 | 110 | 600 | 1100 | 600 | 1100 | 6000 | 11000 | | 2000 |
| OWM | 86.9±1.97 | | 68.9±2.14 | | 69.3±1.78 | | 30.8±0.70 | | 64.0 | 41.9 |
| LwF.R | 69.1±0.03 | 68.5±1.09 | 64.0±0.33 | 63.4±2.56 | 69.2±1.94 | 66.0±1.80 | 52.0±0.24 | 51.9±0.19 | 63.0 | 65.4 |
| iCaRL | 93.0±0.00 | 93.1±0.00 | 73.1±0.08 | 73.8±0.02 | 73.9±0.07 | 74.7±0.03 | 63.0±0.00 | 63.6±0.00 | 76.2 | 68.4 |
| A-GEM | 85.8±0.50 | 83.8±0.44 | 67.7±0.02 | 72.2±0.02 | 68.3±0.05 | 72.8±0.07 | 34.0±0.10 | 34.1±0.20 | 64.8 | 22.3 |
| BiC | 84.4±1.53 | 86.6±2.61 | 77.7±2.76 | 79.7±0.87 | 71.3±4.05 | 77.4±0.77 | 64.2±0.55 | 64.0±2.01 | 75.7 | 68.7 |
| DER++ | 92.0±0.47 | 93.0±0.26 | 77.4±0.15 | 78.8±0.22 | 75.9±0.11 | 78.7±0.29 | 63.7±0.21 | 65.2±0.24 | 78.1 | 67.6 |
| HAL | 89.8±0.71 | 91.4±0.33 | 67.7±0.71 | 70.0±0.64 | 67.8±0.56 | 70.0±0.39 | 60.0±0.13 | 62.2±0.08 | 73.4 | 23.6 |
| EEC† | 93.4±0.20 | | 76.2±0.28 | | 78.1±0.10 | | 42.0±0.89 | | 72.4 | 31.3 |
| PASS | 91.1±0.13 | | 72.4±0.22 | | 71.7±0.14 | | 60.8±0.09 | | 74.0 | 52.0 |
| PLS | **95.2**±0.06 | **95.4**±0.07 | **81.1**±0.09 | **82.2**±0.29 | 80.9±0.05 | 81.8±0.23 | **70.2**±0.11 | **71.1**±0.05 | **82.2** | |
| iPLS | 95.1±0.03 | **95.4**±0.07 | 80.8±0.20 | 82.0±0.17 | **81.0**±0.32 | **82.1**±0.04 | 70.1±0.11 | **71.1**±0.10 | **82.2** | |
| Diff. | +1.8 | +2.3 | +3.4 | +2.5 | +2.9 | +3.4 | +6.0 | +5.9 | +4.1 | |

## B FORGETTING RATE

In the main content, we show the forgetting rate of each system on CIFAR10-5T and CIFAR100-10T. This section presents the results on the other two settings that are not reported in the main paper due to space limitation. We can make a similar observation in Figure 3 as in the main paper. Some baseline methods such as LwF.R, iCaRL, BiC, and PASS forget less, but their accuracy values are much lower (refer to Table 1 in the main paper) as they are not able to adapt to new tasks. PLS and iPLS, however, show competitive results in preventing forgetting while achieving much higher accuracies than the baselines. Our approaches are flexible to learn new tasks while preventing forgetting effectively.

## C ADDITIONAL DERIVATION DETAILS

We have claimed that the orthonormal matrix $\tilde{\boldsymbol{U}}$ and a diagonal matrix $\tilde{\boldsymbol{\Lambda}}^2$ obtained from Eq. 8 are eigenvectors and eigenvalues of unnormalized sample covariance $(n+m-1)\boldsymbol{\Sigma} + r$, where $r$ is a constant. Denote the previous sample mean by $\boldsymbol{\mu}$ and the eigenpairs of previous covariance $\boldsymbol{\Sigma}_{\text{old}}$ by $(\boldsymbol{U}, \boldsymbol{\Lambda}^2)$. Following (Ross et al., 2008), we provide more details about the claim. Since $\hat{\boldsymbol{B}} = [\boldsymbol{X}_{\text{new}} - \boldsymbol{\mu}_{\text{new}} \;\; \sqrt{nm/(n+m)}(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})]$ and $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \boldsymbol{\Sigma}_{\text{old}}$,

$$\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Lambda}}^2\tilde{\boldsymbol{U}}^T = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Lambda}}\tilde{\boldsymbol{V}}^T[\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Lambda}}\tilde{\boldsymbol{V}}^T]^T \tag{9}$$

$$= [\sqrt{n-1}\boldsymbol{U}\boldsymbol{\Lambda} \;\; \hat{\boldsymbol{B}}][\sqrt{n-1}\boldsymbol{U}\boldsymbol{\Lambda} \;\; \hat{\boldsymbol{B}}]^T \tag{10}$$

$$= (n-1)\boldsymbol{U}\boldsymbol{\Lambda}^2\boldsymbol{U}^T + \hat{\boldsymbol{B}}\hat{\boldsymbol{B}}^T \tag{11}$$

$$= (n-1)\boldsymbol{\Sigma}_{\text{old}} + (m-1)\boldsymbol{\Sigma}_{\text{new}} + \frac{nm}{n+m}(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})^T \tag{12}$$

$$= (n+m-1)\boldsymbol{\Sigma} + r \tag{13}$$

where $r = nm/(n+m)(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})(\boldsymbol{\mu}_{\text{new}} - \boldsymbol{\mu})^T$.

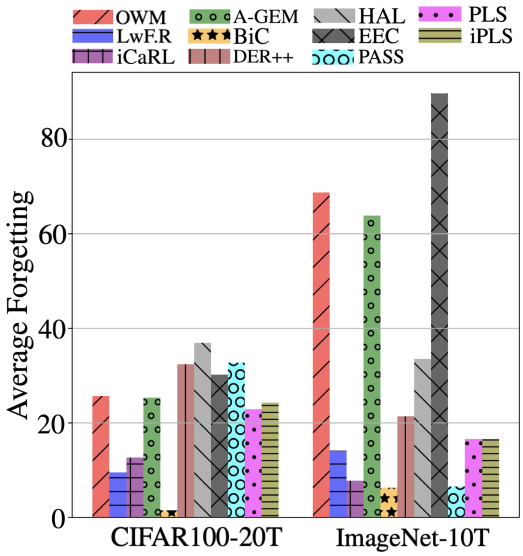



Figure 3: Average forgetting rate (%) on CIFAR100-20T and ImageNet-10T. The lower the value, the better the system is.

## D Zero-Shot Accuracy of CLIP

The multi-modal pre-trained network CLIP is proposed for zero-shot transfer. Given that the text labels of training data are provided, the CLIP text encoder generates weights, and the network predicts the classes based on the pairwise cosine similarity between the feature outputs and the generated weights. Continual learning does not assume that the text information of training data is available. In continual learning, the learners sequentially acquire a new set of knowledge rather than just associating the encoded information as in zero-shot classification. However, we report the zero-shot accuracy of CLIP on the datasets we used in the experiments to demonstrate that our proposed method PLS is not just because of the zero-shot property of CLIP. We run CLIP zero-shot prediction using the official code and report the results in Table 4. The result shows that PLS and iPLS are better than CLIP zero-shot classification despite that PLS and iPLS do not require text information.

Table 4: Zero-shot accuracy of CLIP when the feature extractor, ViT-B/32, is used and the text labels of training data are available.

| | CIFAR10 | CIFAR100 | ImageNet |
|---|---|---|---|
| CLIP | 88.8 | 61.7 | 59.7 |
| PLS | 93.1 | 72.5 | 66.8 |
| iPLS | 93.0 | 72.8 | 67.0 |

## E Public Codes used for Baselines

The links below are the public codes we used to produce the baseline results in our experiments.

- OWM: https://github.com/beijixiong3510/OWM
- LwF.R: https://github.com/yaoyao-liu/class-incremental-learning/tree/main/mnemonics-training
- iCaRL: https://github.com/yaoyao-liu/class-incremental-learning/tree/main/mnemonics-training
- A-GEM: https://github.com/aimagelab/mammoth

- BiC: https://github.com/yaoyao-liu/class-incremental-learning/tree/main/mnemonics-training
- DER++: https://github.com/aimagelab/mammoth
- HAL: https://github.com/aimagelab/mammoth
- PASS: https://github.com/Impression2805/CVPR21_PASS
- EEC: https://github.com/aliayub7/EEC
- PCL: https://github.com/morning-dews/PCL
- CLIP zero-shot: https://github.com/openai/CLIP