# A MULTI-HEAD MODEL FOR CONTINUAL LEARNING VIA OUT-OF-DISTRIBUTION REPLAY

**Gyuhak Kim, Zixuan Ke, Bing Liu**
Department of Computer Science, University of Illinois at Chicago
{gkim87,zke4,liub}@uic.edu

## ABSTRACT

This paper studies *class incremental learning* (CIL) of continual learning (CL). Many approaches have been proposed to deal with catastrophic forgetting (CF) in CIL. Most methods incrementally construct a single classifier for all classes of all tasks in a single head network. To prevent CF, a popular approach is to memorize a small number of samples from previous tasks and replay them during training of the new task. However, this approach still suffers from serious CF as the parameters learned for previous tasks are updated or adjusted with only the limited number of saved samples in the memory. This paper proposes an entirely different approach that builds a separate classifier (head) for each task (called a multi-head model) using a transformer network, called MORE. Instead of using the saved samples in memory to update the network for previous tasks/classes in the existing approach, MORE leverages the saved samples to build a task specific classifier (adding a new classification head) without updating the network learned for previous tasks/classes. The model for the new task in MORE is trained to learn the classes of the task and also to detect samples that are not from the same data distribution (i.e., *out-of-distribution* (OOD)) of the task. This enables the classifier for the task to which the test instance belongs to produce a high score for the correct class and the classifiers of other tasks to produce low scores because the test instance is not from the data distributions of these classifiers. Experimental results show that MORE outperforms state-of-the-art baselines and is also naturally capable of performing OOD detection in the continual learning setting.[1]

## 1 INTRODUCTION

Continual learning (CL) is a learning paradigm in which a system learns a sequence of tasks sequentially and accumulates the knowledge learned in the process (Chen & Liu, 2018). A main challenge of CL is how to adapt the existing knowledge in learning the new task without causing catastrophic forgetting (CF) (McCloskey & Cohen, 1989). CF refers to the phenomenon that the system forgets some of the previous knowledge after learning the new task due to modifications to model parameters learned for previous tasks. This paper proposes a novel method for the challenging CL setting of *class incremental learning* (CIL) (Rebuffi et al., 2017; van de Ven & Tolias, 2019). In this setting, the system learns a sequence of tasks $< 1, \cdots, k, \cdots >$ incrementally with dataset $\mathcal{D}^k : \mathcal{X}^k \times \mathcal{Y}^k$ for each new task, which has a set of classes that are different from all the previous tasks. At any time, the system is expected to be able to classify a test instance $\boldsymbol{x}$ to one of the classes that have been learned so far without any information about the task it belongs to.

The proposed approach is a memory-based method (also called a *replay-based* method). In this method, the system saves a small fraction of training samples in a memory buffer of a fixed size after learning each task. Existing memory-based methods typically train a single head network, in which the network has only a single classifier for all tasks (Rebuffi et al., 2017). In training the new task, the algorithm replays the samples in the memory to prevent forgetting of the previously learned knowledge through joint training with the current task data. The approach learns the new task by adjusting the network parameters for previous tasks/classes while also regularizing the changes in parameters so that the knowledge learned for the previous tasks is not forgotten. However, with only a limited number of saved samples to help overcome CF, the resulting network still can forget a great deal.

This paper proposes a novel method called a *M*ulti-head model for continual learning via *O*OD *RE*play (MORE). Unlike the existing CIL methods, MORE constructs a multi-head model which consists of a set of separate classifiers built for different tasks. The parameters for one task in this setting do not interfere with parameters of other tasks as they are independent of each other, although there is a great deal of parameter sharing cross tasks. By construction, a multi-head model is robust to forgetting because MORE makes use of the highly effective *hard attention* (also called masking)

[1]The code of MORE is available at https://github.com/k-gyuhak/MORE.

mechanism in HAT (Serra et al., 2018) to protect important model parameters of each task to prevent modification and forgetting. However, HAT was originally proposed to work with the *task-incremental learning* (TIL) setting of CL, which requires the correct task identifier (task-id) for each test sample to be given at inference. The proposed technique enables our multi-head model/classifier to function without giving the task-id for each test sample.

The key novelty of MORE is that it utilizes the saved samples in the memory buffer entirely differently. Instead of replaying these samples to prevent forgetting in learning the new classes by updating the network learned for previous tasks or classes, MORE uses these saved samples to learn a new and independent classifier that is capable of detecting out-of-distribution (OOD) samples. MORE also uses a pre-trained transformer network for improved performance.[2] To ensure that the pre-trained model does not see any similar data used in continual learning, in our experiments we manually remove all the data used in pre-training that may overlap with the continual learning data (see Section 4.1).

Inspired by (Houlsby et al., 2019; Ke et al., 2021) in natural language processing, MORE does not modify the pre-trained network but only uses a trainable *adapter module* inserted at each transformer layer. In continual learning, the system trains a new task specific classifier in the shared adapter to classify the classes of the task and also determine whether a sample is OOD. The network parameters learned the previous tasks is left untouched in backpropagation as they are protected by trained hard attentions. The resulting classifier can produce a high score (e.g., prediction probability) for an in-distribution (IND) sample (the distribution of the current task training data) as it is one of the classes that the classifier has been trained for and a low score for an OOD sample as it does not belong to the IND classes of the task. These can be achieved because the network is exposed to both IND data (the current task data) and OOD data (the past data saved in the memory buffer) in training each task. The prediction for each test sample is made by comparing the output scores of all the task classifiers. Thus, the system does not need to know the task-id of a test instance. This base method is further improved by a Mahalanobis distance based technique (see below).

In summary, this paper makes the following contributions.

- It proposes a novel approach for class incremental learning (CIL). It builds a multi-head model and trains each task using samples in the replay memory as OOD data against the current task data unlike the existing methods. To the best of knowledge, this way of using the memory or replay data has not been done before.
- It further improves the above base method by using a Mahalanobis distance-based technique. The paper proposes to combine this distance factor and the softmax probability output scores of the above base method to make the final prediction decision. This method can be considered as an ensemble of the two methods.
- Since the proposed method is learned based on OOD detection for each task, it can naturally perform OOD detection in the continual learning (CL) setting to identify any OOD test instances that do not belong to any of the learned tasks so far. Again, we are unaware of any existing CL system that does continual OOD detection.

Experimental results show that the proposed method outperforms existing state-of-the-art CIL method both in terms of IND classification accuracy and continual OOD detection.

## 2 Related Work

A large number of approaches have been proposed to prevent forgetting in continual learning (see the book (Chen & Liu, 2018) and the survey (Masana et al., 2020)). Using regularizations (Kirkpatrick et al., 2017) is a popular approach. This approach approximates the importance of parameters for the previous tasks, and penalizes changes to them to prevent forgetting (Kirkpatrick et al., 2017; Zenke et al., 2017; Ritter et al., 2018; Dhar et al., 2019; Mirzadeh et al., 2021). Our approach is different as we do not use regularization.

Saving a small fraction of training data in a memory and replaying it to prevent forgetting is another very popular approach, called the *replay* method. The saved samples are used to distill the knowledge of previous tasks (Rusu et al., 2016; Rebuffi et al., 2017; Hou et al., 2019; Wu et al., 2019; Buzzega et al., 2020; Yan et al., 2021) or replayed to restrict the loss from increasing on the memory or selective representations (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018; 2021). Some researchers also proposed different replay strategies (Riemer et al., 2018; Aljundi et al., 2019b; Chaudhry et al., 2019; Liu et al., 2020a). Our method MORE also uses a memory, but it does not replay the samples to regularize the adjustments to the previous task model parameters in learning the current task so that the previous task models are minimally changed. MIR (Aljundi et al., 2019a) proposes a sampling strategy that draws maximally interfering samples from memory. MORE does not propose a sampling strategy, but uses the saved samples

[2] Using a pre-trained network is justified as such networks (or feature extractors) are increasingly used in computer vision as in natural language processing, where almost every application uses a pre-trained network. Furthermore, neuroscience research has shown that humans are born with rich feature detectors gained through millions of years of evolution (Zhaoping & Li, 2014).

as OOD data to improve the current task learning. Models of the previous tasks are not updated. Some methods (Lee et al., 2019; Liu et al., 2020b; Smith et al., 2021) use external data to improve knowledge distillation for overcoming forgetting. Our method does not use any external data or knowledge distillation.

*Pseudo-replay* is another popular approach (Shin et al., 2017), which constructs a separate encoder-decoder network to generate raw data similar to previous task data. During training a new task, the system generates pseudo data similar to the samples of previous tasks. The generated data and new task data are trained together to distinguish the previous and current classes (Kamra et al., 2017; Shin et al., 2017; Ostapenko et al., 2019; Rostami et al., 2019; van de Ven et al., 2020; Zhu et al., 2021). Our method MORE does not use this approach.

The list of other approaches is also extensive. *Parameter isolation* methods (Serra et al., 2018; Wortsman et al., 2020; Henning et al., 2021) are highly effective at preventing forgetting in task incremental learning, which requires the task-id for each test instance during inference. MORE borrows the hard attention (Serra et al., 2018) idea to prevent modifications to each previous task model. However, it does not need the task-id in testing and further uses memory data for OOD detection to improve CIL performances. *Gradient projection* (Zeng et al., 2019) projects gradients to a subspace orthogonal to previously learned parameter space to prevent intervention. Some such methods (Kim & Liu, 2020; Chaudhry et al., 2020; Saha et al., 2021) require task-id for more complex tasks to recall the task-specific projection at inference. PCL (Hu et al., 2021) constructs an individual classifier for each class by using a fixed pre-trained feature extractor. We build an OOD model for each task and use hard attention to protect it.

A related line of research is online CL. Contrary to batch CL where the system sees the full training data when the task arrives and can train any number of epochs, online CL only sees the training data once (training in one epoch). There are many existing online CL systems, e.g., SLDA (Hayes & Kanan, 2020), REMIND (Hayes et al., 2020) and OCM (Guo et al., 2022). (Roady et al., 2020) also proposed a new dataset. Our method is a batch CL method.

Recent approaches in task incremental learning (where the task id is required for each test instance) have attempted to use a multi-head model for CIL. CCG (Abati et al., 2020) constructs an additional network to predict the task-id. MORE does not build another network. iTAML (Rajasegaran et al., 2020) uses the correct task network by identifying the task-id of the test data in a batch. The limitation is that it requires the test data in a batch to belong to the same task. MORE is different as it can predict for a single instance at a time. HyperNet (von Oswald et al., 2020) and PMCL (Henning et al., 2021) propose an entropy based task-id prediction method. Our method enables each task model to produce better scores on OOD outputs and should also be applicable to HyperNet and PMCL. SupSup (Wortsman et al., 2020) predicts task-id by finding the optimal superpositions at inference. Our method does not require any additional operations such as entropy or optimization to predict task-id. It simply chooses the class with the maximum probability among all task model outputs. Our recent work CLOM (Kim et al., 2022) uses contrastive learning and data augmentation to build an OOD classifier for each task, but it is not a replay method and is weaker than MORE.

## 3 Proposed MORE Technique

As mentioned earlier, existing class incremental learning (CIL) methods mostly train a single-head network for all the classes learned so far (i.e. a single classifier for all the tasks) (van de Ven & Tolias, 2019). To learn each task, the parameters for previous tasks must be modified, which causes CF. To alleviate CF, the popular replay approach saves a small number of samples of previous tasks in a memory/replay buffer and use them in learning the new task. However, this results in a biased network due to sample imbalance between the saved samples and current data (Rebuffi et al., 2017). We take an entirely different approach by training a multi-head network as an adapter to a pre-trained network. Each head is an OOD detection model for a task. Hard attentions are employed to protect each task model or classifier.

### 3.1 Training an OOD Detection Model

At task $k$, the system receives the training data $\mathcal{D}^k : \mathcal{X}^k \times \mathcal{Y}^k$. We train the feature extractor $\boldsymbol{z} = h(\boldsymbol{x}, k; \theta)$ and task specific classifier $f(\boldsymbol{z}; \phi_k)$ using $\mathcal{D}^k$ and the samples in the memory buffer $\mathcal{M}$. We treat the buffer data as OOD data to encourage the network to learn the current task and also detect ODD samples (the models or classifiers of the previous tasks are not touched). We achieve it by maximizing $p(y|\boldsymbol{x}, k) = \text{softmax} f(h(\boldsymbol{x}, k; \theta); \phi_k)$ for an IND sample $\boldsymbol{x} \in \mathcal{X}^k$ and maximizing $p(ood|\boldsymbol{x}, k)$ for an OOD sample $\boldsymbol{x} \in \mathcal{M}$. The additional label $ood$ is reserved for previous and future unseen classes. Figure 1(a) shows the overall idea of the proposed approach. We formulate the problem as follows.

Given the training data $\mathcal{D}^k$ of size $N$ at task $k$ and the memory buffer $\mathcal{M}$ of size $M$, we minimize the loss

$$\mathcal{L}_{ood}(\theta, \phi_k) = -\frac{1}{M+N} \sum_{(\boldsymbol{x},y)\in\mathcal{M}} \log p(ood|\boldsymbol{x}, k) + \sum_{(\boldsymbol{x},y)\in\mathcal{D}^k} \log p(y|\boldsymbol{x}, k) \tag{1}$$

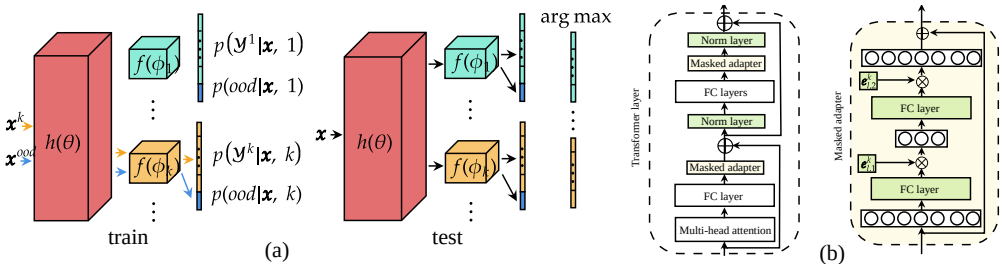


Figure 1: (a) We train the feature extractor and the task classifier $k$ at task $k$. The output values of the classifier correspond to $|\mathcal{Y}^k| + 1$ classes, in which the last class is for OOD (i.e., representing previous and unseen future classes). At inference/testing, the probability values of each task model without the OOD class are concatenated and the system chooses the class with the maximum score. (b) Transformer and adapter module. The masked adapter network consists of 2 fully connected layers and task specific masks. During training, only the masked adapters and norm layers are updated and the other parts in the transformer layers remain unchanged.

It is the sum of two categorical cross-entropy losses. The first loss is for learning OOD samples while the second loss is for learning the classes from the current task. We optimize the shared parameter $\theta$ in the feature extractor. The task specific classification parameters $\phi_k$ are independent of other tasks. The learned representation on the current data is robust to outliers or OOD data. The classifier produces more accurate outputs on both IND and OOD data.

In testing, we perform prediction by using all the task classifiers from task 1 to $k$ without the OOD class and compare the softmax probability output values as

$$\hat{y} = \arg\max \bigoplus_{1 \le j \le k} p(\mathcal{Y}^j | \boldsymbol{x}, j) \tag{2}$$

where $\bigoplus$ is the concatenation over the output space. Figure 1(a) shows the prediction rule. We are basically choosing the class with the highest probability value/score among all the task outputs.

### 3.2 Back-Updating the Previous OOD Models

Each task model works better if more diverse OOD data is provided during training. As in a replay-based approach, MORE saves an equal number of samples per class after each task (Chaudhry et al., 2019). The saved samples in the memory are used as OOD samples for each new task. Thus, in the beginning of continual learning when the system is trained on only a small number of tasks, the classes of samples in the memory are less diverse than after more tasks are learned. This makes the performance of OOD detection stronger for later tasks, but weaker in earlier tasks. To prevent this asymmetry, we update the model of each previous task so that it can also identify the samples from subsequent classes (which were unseen during the training of the previous task) as OOD samples.

At task $k$, we update the previous task models $(j = 1, \cdots, k-1)$ as follows. Denote the samples of task $j$ in memory $\mathcal{M}$ by $\tilde{\mathcal{D}}^j$. We construct a new dataset using the current task dataset and the samples in the memory buffer. We randomly select $|\mathcal{M}|$ samples from the training data $\mathcal{D}^k$ and combine it with the remaining samples in $\mathcal{M}$ after removing the IND samples $\tilde{\mathcal{D}}^j$ of task $j$. We do not use the entire training data $\mathcal{D}^k$ as we do not want a large sample imbalance between IND and OOD. Denote the new dataset by $\tilde{\mathcal{M}}$. Using the data, we update only the parameters $\phi_j$ of the classifier for task $j$ with the feature representations frozen by minimizing the loss

$$\mathcal{L}(\phi_j) = -\frac{1}{2M} \sum_{(\boldsymbol{x},y) \in \tilde{\mathcal{M}}} \log p(ood|\boldsymbol{x}, j) + \sum_{(\boldsymbol{x},y) \in \tilde{\mathcal{D}}^j} \log p(y|\boldsymbol{x}, j) \tag{3}$$

We reduce the loss by updating the parameters of classifier $j$ to maximize the probability of the class if the sample belongs to task $j$ and maximize the OOD probability otherwise.

### 3.3 Improving Prediction Performance by a Distance Based Technique

We further improve the prediction in Eq. 2 by introducing a distance based factor used as a coefficient to the softmax probabilities in Eq. 2. It is quite intuitive that if a test instance is close to a class, it is more likely to belong to the class. We thus propose to combine this new distance factor and the softmax probability output of the task $k$ model to make the final prediction decision. In some sense, this can be considered as an ensemble of the two methods.

We define the distance based coefficient $s^k(\boldsymbol{x})$ of task $k$ for the test instance $\boldsymbol{x}$ by the maximum of inverse Mahalanobis distance (Lee et al., 2018) between the feature of $\boldsymbol{x}$ and the Gaussian distributions of the classes in task $k$ parameterized by the mean $\boldsymbol{\mu}_j^k$ of the class $j$ in task $k$ and the sample covariance $\boldsymbol{S}^k$. They are estimated by the features of class $j$'s training data for each class $j$ in task $k$. If a test instance is from the task, its feature should be close to the distribution that the instance belongs to. Conversely, if the instance is OOD to the task, its feature should not be close to any of the distributions of the classes in the task. More precisely, for task $k$ with class $y_1, \cdots, y_{|\mathcal{Y}^k|}$ (where $|\mathcal{Y}^k|$ represents the number of classes in task $k$), we define the coefficient $s^k(\boldsymbol{x})$ as

$$s^k(\boldsymbol{x}) = \max\left[c/\text{MD}(\boldsymbol{x}; \boldsymbol{\mu}_{y_1}^k, \boldsymbol{S}^k), \cdots, c/\text{MD}(\boldsymbol{x}; \boldsymbol{\mu}_{y_{1+|\mathcal{Y}^k|}}^k, \boldsymbol{S}^k)\right] \tag{4}$$

where $c$ is a positive constant and $\text{MD}(\boldsymbol{x}; \boldsymbol{\mu}_j^k, \boldsymbol{S}^k)$ is the Mahalanobis distance. The coefficient is large if at least one of the Mahalanobis distances is small but the coefficient is small if all the distances are large (i.e. the feature is far from all the distributions of the task). The parameters $\boldsymbol{\mu}_j^k$ and $\boldsymbol{S}^k$ can be computed and saved when each task is learned. The mean $\boldsymbol{\mu}_j^k$ is computed using the training samples $\mathcal{D}_j^k$ of class $j$ as follows,

$$\boldsymbol{\mu}_j^k = \sum_{\boldsymbol{x} \in \mathcal{D}_j^k} h(\boldsymbol{x}, j)/|\mathcal{D}_j^k| \tag{5}$$

and the covariance $\boldsymbol{S}^k$ of task $k$ is the mean of covariances of the classes in task $k$,

$$\boldsymbol{S}^k = \sum_{j \in \mathcal{Y}^k} \boldsymbol{S}_j^k/|\mathcal{Y}^k| \tag{6}$$

where $\boldsymbol{S}_j^k = \sum_{\boldsymbol{x} \in \mathcal{D}_j^k} (\boldsymbol{x} - \boldsymbol{\mu}_j^k)^T(\boldsymbol{x} - \boldsymbol{\mu}_j^k)/|\mathcal{D}_j^k|$ is the sample covariance of class $j$. By multiplying the coefficient $s^k(\boldsymbol{x})$ to the original softmax probabilities $p(\mathcal{Y}^k|\boldsymbol{x}, k)$, the task output $p(\mathcal{Y}^k|\boldsymbol{x}, k)s^k(\boldsymbol{x})$ increases if $\boldsymbol{x}$ is from task $k$ and decreases otherwise. The final prediction is made by (which replaces Eq. 2)

$$y = \arg\max \bigoplus_{1 \le k \le t} p(\mathcal{Y}^k|\boldsymbol{x}, k)s^k(\boldsymbol{x}), \tag{7}$$

### 3.4 Preventing Forgetting by Hard Attention Masks

We have discussed how to train a task model capable of CIL prediction as an OOD classifier. In this section, we use notations from (Kim et al., 2022) and explain how to use the hard attention mechanism (Serra et al., 2018) to protect the feature extractor of each task model. In learning a task, a set of embeddings are trained to protect the important neurons so that the corresponding parameters are not interfered by subsequent tasks. The importance of a neuron is measured by the 0-1 pseudo-step function, where 0 indicates important and 1 indicates not important (and thus trainable).

The hard attention mask is an output of sigmoid function $u$ with a hyper-parameter $s$ as

$$\boldsymbol{a}_l^k = u(s\boldsymbol{e}_l^k), \tag{8}$$

where $\boldsymbol{e}_l^k$ is a learnable embedding at layer $l$ of task $k$. Since the step function is not differentiable, a sigmoid function with a large $s$ is used to approximate it. Sigmoid is approximately a 0-1 step function with a large $s$. The attention is multiplied to the output $\boldsymbol{h}_l = \text{ReLU}(\boldsymbol{W}_l\boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$ of layer $l$,

$$\boldsymbol{h}_l' = \boldsymbol{a}_l^k \otimes \boldsymbol{h}_l \tag{9}$$

The $j$th element $a_{j,l}^k$ in the attention mask blocks (or unblocks) the information flow from neuron $j$ at layer $l$ if its value is 0 (or 1). With 0 value of $a_{j,l}^k$, the corresponding parameters in $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$ can be freely changed as the output values $\boldsymbol{h}_l'$ are not affected. The neurons with non-zero mask values are necessary to perform the task, and thus need a protection for catastrophic forgetting.

We modify the gradients of parameters that are important in performing the previous tasks $(1, \cdots, k-1)$ during training task $k$ so they are not interfered. Denote the accumulated mask by

$$\boldsymbol{a}_l^{<k} = \max(\boldsymbol{a}_l^{<k-1}, \boldsymbol{a}_l^{k-1}) \tag{10}$$

where $\max$ is an element-wise maximum and the initial mask $\boldsymbol{a}_l^0$ is a zero vector. It is a collection of mask values at layer $l$ where a neuron has value 1 if it has ever been activated previously. The gradient of parameter $w_{ij,l}$ is modified as

$$\nabla w_{ij,l}' = \left(1 - \min\left(a_{i,l}^{<k}, a_{j,l-1}^{<k}\right)\right) \nabla w_{ij,l} \tag{11}$$

where $a_{i,l}^{<k}$ is the $i$th unit of $\boldsymbol{a}_l^{<t}$. The gradient flow is blocked if both neurons $i$ in the current layer and $j$ in the previous layer have been activated. We apply the mask for all layers except the last layer. The parameters in last layer do not need to be protected as they are task-specific parameters.

A regularization is introduced to encourage sparsity in $\boldsymbol{a}_l^k$ and parameter sharing with $\boldsymbol{a}_l^{<k}$. The capacity of a network depletes when $\boldsymbol{a}_l^{<k}$ becomes 1-vector in all layers. Despite a set of new neurons can be added in network at any point in training for more capacity, we utilize resources more efficiently by minimizing the loss

$$\mathcal{L}_r = \lambda \frac{\sum_l \sum_i a_{i,l}^k \left(1 - a_{i,l}^{<k}\right)}{\sum_l \sum_i \left(1 - a_{i,l}^{<k}\right)} \tag{12}$$

where $\lambda$ is a hyper-parameter. The final objective to train a comprehensive task network without forgetting is

$$\mathcal{L} = \mathcal{L}_{ood} + \mathcal{L}_r \tag{13}$$

where $\mathcal{L}_{ood}$ is Eq. 1 for training an OOD model discussed in Sec. 3.1 after incorporating the trainable embedding $\boldsymbol{e}^k$. We describe the whole training and prediction process in Appendix A.

## 4 Experiments

**Evaluation Datasets**: We use three image classification benchmark datasets in our experiments.[3]

(1). **CIFAR-10** (Krizhevsky & Hinton, 2009) : This is an image classification dataset consisting of 60,000 32x32 color images of 10 classes, with 6,000 images per class. We use 5,000 samples for training and 1,000 samples for testing.

(2). **CIFAR-100** (Krizhevsky & Hinton, 2009) : This dataset consists of 60,000 32x32 color images of 100 classes, with 600 images per class. We use 500 samples for training and 100 for testing.

(3). **Tiny-ImageNet** (Le & Yang, 2015): This dataset consists of 120,000 64x64 color images of 200 classes, with 600 images per class, in which 500, 50, and 50 images are for training, validation, and testing, respectively. Following the previous works (Zhu et al., 2021), we use the validation data for testing as the test data has no labels.

**Baseline Systems**: We compare our method MORE with 6 state-of-the-art baselines. For gradient modification methods, we consider **OWM** (Zeng et al., 2019). For replay-based methods, we compare against **iCaRL** (Rebuffi et al., 2017), **A-GEM** (Chaudhry et al., 2018), **EEIL** (Castro et al., 2018), **GD** (Lee et al., 2019) without external data, **DER++** (Buzzega et al., 2020), and **HAL** (Chaudhry et al., 2021). For pseudo-replay method, we include **PASS** (Zhu et al., 2021). For multi-head methods, we compare with HAT (Serra et al., 2018) using the task-id prediction method proposed in HyperNet (von Oswald et al., 2020).[4]

### 4.1 Pre-trained Network

We pre-train a vision transformer (Touvron et al., 2021) using a subset of the ImageNet data (Russakovsky et al., 2015) and apply the pre-trained network to all baselines and our method. To ensure that there is no overlapping of data between ImageNet and our experimental datasets, we manually removed 389 classes from the original 1000 classes in ImageNet that are similar/identical to the classes in CIFAR-10, CIFAR-100, or Tiny-ImageNet. We pre-train the network with the remaining subset of 611 classes of ImageNet.

Using the pre-trained network, both our system and the baselines improve dramatically compared to their versions without using the pre-trained network. For instance, the two best baselines (DER++ and PASS) in our experiments achieve the average classification accuracy of 66.89 and 68.25 (after the final task) with the pre-trained network over 5 experiments while they achieve only 46.88 and 32.42 without using the pre-train network.

[3]We conduct additional experiment using SVHN (very different from pre-training data) and include the result in Appendix B.

[4]CCG (Abati et al., 2020), iTAML (Rajasegaran et al., 2020), HyperNet (von Oswald et al., 2020), and SupSup (Wortsman et al., 2020) are not included because: CCG's code is not released. iTAML requires a batch of test data and each batch must be from the same task. When a single test instance is provided, its accuracy on CIFAR100-10T is 33.5%, which is much lower than many other baselines. HyperNet and SupSup cannot work with the transformer adapter due to their architectures. The CIL performances of HyperNet and SupSup with ResNet-18 on CIFAR100-10T are 29.6 and 33.1, respectively, which are even lower than that of iTAML. Our earlier system CLOM (Kim et al., 2022) (which is also based on OOD detection as well) is not included as it is much weaker (up to 15% lower than MORE in accuracy) and its approach of using contrastive learning and data augmentations does not work well with a pre-trained model.

We insert an adapter module at each transformer layer to exploit the pre-trained transformer network in continual learning. During training, the adapter module and the layer norm are trained while the transformer parameters are unchanged to prevent forgetting in the pre-trained network. Existing continual learning methods in natural language processing commonly use pre-trained transformers and adapter-like modules (Ke et al., 2021). Therefore, we also leverage it for computer vision tasks.

### 4.2 Training Details

For all experiments, we use the same backbone architecture DeiT-S/16 (Touvron et al., 2021) with 2-layers adapter (Houlsby et al., 2019) at each transformer layer, and the same class order for both baselines and our method. The first fully-connected layer in adapter maps from dimension 384 to bottleneck. The second fully-connected layer following ReLU activation function maps from bottleneck to 384. The the bottleneck dimension is the same for all adapters in a model. For our method, we use SGD with momentum value 0.9. The back-update method in Sec. 3.2 is also a hyper-parameter choice. If we apply it, we train each classifier for 10 epochs by SGD with learning rate 0.01, batch size 16, and momentum value 0.9. We choose 500 for $s$ in Eq. 8, 0.75 for $\lambda$ in Eq. 12 as recommended in Serra et al. (2018), and 20 for $c$ in distance based coefficient in Eq. 4 of Sec. 3.3. We use 10% of training data as the validation set to find a good set of learning rates and number of epochs. We follow (Chaudhry et al., 2019) and save an equal number of random samples per class in the replay memory. Following the experiment settings in (Rebuffi et al., 2017; Zhu et al., 2021), we fix the size of memory buffer and reduce the saved samples to accommodate a new set of samples after a new task is learned. We use the class order protocol in (Rebuffi et al., 2017; Buzzega et al., 2020) by generating random class orders for the experiments. The baselines and our method use the same class ordering for fairness. We also report the size of memory required for each experiment in Appendix C.

For CIFAR-10, we split 10 classes into 5 tasks (2 classes per task). The bottleneck size in each adapter is 64. Following (Buzzega et al., 2020), we use the memory size 200, and train for 20 epochs with learning rate 0.005, and apply the back-update method in Sec. 3.2.

For CIFAR-100, we conduct two experiments: 10 tasks and 20 tasks. We split 100 classes into 10 and 20 tasks, where each task has 10 classes and 5 classes, respectively. We double the bottleneck size of the adapter to learn more classes. We use the memory size 2000 following (Rebuffi et al., 2017) and train for 40 epochs with learning rate 0.001 and 0.005 for 10 tasks and 20 tasks, respectively, and apply the back-update method in Sec. 3.2.

For Tiny-ImageNet, two experiments are conducted. We split 200 classes into 5 and 10 tasks, where each task has 40 classes and 20 classes per task, respectively. We use the bottleneck size 128, and save 2000 samples in memory. We train with learning rate 0.005 for 15 and 10 epochs for 5 tasks and 10 tasks, respectively. There is no need to use the back-update method as the earlier tasks already have diverse OOD classes.

### 4.3 Results and Comparisons

We evaluate our method with the standard metrics: *average classification accuracy* (ACA), *average incremental accuracy* (AIA), and *average performance reduction rate* (which is commonly called the *average forgetting rate* (Liu et al., 2020a)) after the final task. The average classification accuracy $\mathcal{A}^k$ after task $k$ is the accuracy of all seen classes after task $k$. We report ACA after the final task. The average incremental accuracy (Rebuffi et al., 2017) $\mathcal{A}$ after the final task $t$ is defined as $\mathcal{A} = \sum_{k=1}^{t} \mathcal{A}^k/t$. The average performance reduction rate is $\mathcal{R}^t = \sum_{k=1}^{t-1}(\mathcal{A}_k^{\text{init}} - \mathcal{A}_k^t)/(t-1)$, where $\mathcal{A}_k^{\text{init}}$ is the classification accuracy on samples of task $k$ right after learning the task $k$. We do not consider the task $t$ as it is the last task. Note that we do not use the term *average forgetting rate* as in other papers (Liu et al., 2020a) because there are actually two factors that cause the performance degradation in continual learning: *forgetting* and *more classes* (as the system learns more tasks/classes, the classification accuracy will naturally decrease). That is why we use the term *average performance reduction rate* instead. All the reported results are the averages of 5 runs.

**Average Accuracy after last task and Average Incremental Accuracy.** Table 1 shows that our method MORE consistently outperforms the baselines. We compare with the replay-based methods first. The best replay-based method by average ACA and AIA over all the datasets is DER++. Our method achieves 71.59 and 80.77 in ACA and AIA, respectively, much better than 66.89 and 78.46 of DER++. This demonstrates that the existing replay-based methods utilizing the samples to prevent forgetting are inferior to our MORE using samples for OOD learning. The best baseline is the generative method PASS. Its average ACA and AIA over all the datasets are 68.25 and 77.11, respectively. Our method achieves much better performances of 71.59 and 80.77 on ACA and AIA, respectively. The performances of the multi-head method HAT (Serra et al., 2018) using task-id prediction are 62.68 and 73.84 for ACA and AIA, respectively. These numbers are lower than many baselines. Its performance is particularly low in experiments where the number of classes per task is small. For instance, its ACA and AIA on CIFAR100-20T are 56.72 and 69.12, respectively, much

Table 1: Average accuracy (ACA) and average incremental accuracy (AIA) after the final task. '-XT' means X number of tasks. Our system MORE and all baselines used the pre-trained network. The last two columns show the average ACA and AIA of each method over all datasets and experiments. We highlight the best results in each column in bold.

| Method | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | T-ImageNet-5T | | T-ImageNet-10T | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA |
| OWM | 41.69±6.34 | 59.07±3.31 | 21.39±3.18 | 39.71±1.35 | 16.98±4.44 | 32.18±1.51 | 24.55±2.48 | 45.65±1.15 | 17.52±3.45 | 35.57±1.83 | 24.43 | 41.99 |
| iCaRL | 87.55±0.99 | 92.75±1.08 | 68.90±0.47 | 77.82±1.28 | 69.15±0.99 | 77.74±1.82 | 53.13±1.04 | 63.35±2.02 | 51.88±2.36 | 64.62±0.97 | 66.12 | 75.26 |
| A-GEM | 56.33±7.77 | 71.22±1.42 | 25.21±4.00 | 43.39±0.88 | 21.99±4.01 | 35.56±0.95 | 30.53±3.99 | 50.37±2.15 | 21.90±5.52 | 39.79±3.28 | 31.20 | 47.74 |
| EEIL | 82.34±3.13 | 90.50±0.72 | 68.08±0.51 | 81.09±0.37 | 63.79±0.66 | 79.54±0.69 | 53.34±0.54 | 66.63±0.40 | 50.38±0.97 | 66.54±0.61 | 63.59 | 76.86 |
| GD | **89.16**±0.53 | 94.22±0.75 | 64.36±0.57 | 80.51±0.57 | 60.10±0.74 | 78.43±0.76 | 53.01±0.97 | 67.51±0.38 | 42.48±2.53 | 63.91±0.40 | 61.82 | 76.92 |
| DER++ | 84.63±2.91 | 91.81±0.65 | 69.73±0.99 | **81.71**±0.67 | 70.03±1.46 | **82.24**±0.79 | 55.84±2.21 | 68.47±0.73 | 54.20±3.28 | 68.06±1.04 | 66.89 | 78.46 |
| HAL | 84.38±2.70 | 90.41±1.04 | 67.17±1.50 | 78.62±0.45 | 67.37±1.45 | 78.43±0.61 | 52.80±2.37 | 67.52±0.93 | 55.25±3.60 | 67.89±2.32 | 65.39 | 76.57 |
| PASS | 86.21±1.10 | 91.78±1.12 | 68.90±0.94 | 78.27±0.81 | 66.77±1.18 | 77.01±1.13 | 61.03±0.38 | 70.02±0.56 | 58.34±0.42 | 68.45±1.20 | 68.25 | 77.11 |
| HAT | 83.30±1.54 | 91.06±0.36 | 62.34±0.93 | 73.99±0.86 | 56.72±0.44 | 69.12±1.06 | 57.91±0.72 | 69.38±1.14 | 53.12±0.94 | 65.63±1.64 | 62.68 | 73.84 |
| MORE | **89.16**±0.96 | **94.23**±0.82 | **70.23**±2.27 | 81.24±1.24 | **70.53**±1.09 | 81.59±0.98 | **64.97**±1.28 | **74.03**±1.61 | **63.06**±1.26 | **72.74**±1.04 | **71.59** | **80.77** |

Table 2: Performance of the baselines and our method MORE with smaller memory sizes. We reduce the size of the memory buffer by half. The new sizes are 100, 1000, 1000 for CIFAR10, CIFAR100, and Tiny-ImageNet. Numbers in bold are the best results in each column.

| Method | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | T-ImageNet-5T | | T-ImageNet-10T | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA |
| OWM | 41.69±6.34 | 59.07±3.31 | 21.39±3.18 | 39.71±1.35 | 16.98±4.44 | 32.18±1.51 | 24.55±2.48 | 45.65±1.15 | 17.52±3.45 | 35.57±1.83 | 24.43 | 41.99 |
| iCaRL | 86.08±1.19 | 92.18±1.24 | 66.96±2.08 | 76.70±0.98 | 68.16±0.71 | 77.78±1.29 | 47.27±3.22 | 59.95±3.20 | 49.51±1.87 | 62.02±3.33 | 63.60 | 73.72 |
| A-GEM | 56.64±4.29 | 68.33±1.73 | 23.18±2.54 | 44.29±1.01 | 20.76±2.88 | 36.63±0.61 | 31.44±3.84 | 50.83±2.76 | 23.73±6.27 | 40.53±3.72 | 31.15 | 48.12 |
| EEIL | 77.44±3.04 | 86.53±1.20 | 62.95±0.68 | 78.66±0.53 | 57.86±0.74 | 76.39±0.59 | 48.36±1.38 | 63.53±0.85 | 44.59±1.72 | 62.98±0.39 | 58.24 | 73.62 |
| GD | 85.96±1.64 | 92.75±0.58 | 57.17±1.06 | 76.57±0.28 | 50.30±0.58 | 73.36±0.73 | 46.09±1.77 | 64.18±0.71 | 32.41±2.75 | 57.68±0.32 | 54.39 | 72.91 |
| DER++ | 80.09±3.00 | 89.42±0.95 | 64.89±2.48 | 78.90±0.26 | 65.84±1.46 | 79.58±0.62 | 50.74±2.41 | 65.19±0.76 | 49.24±5.01 | 63.90±0.67 | 62.16 | 75.40 |
| HAL | 79.16±4.56 | 88.31±1.06 | 62.65±0.83 | 76.33±0.67 | 63.96±1.49 | 76.75±0.84 | 48.17±2.94 | 63.95±1.64 | 47.11±6.00 | 62.43±1.74 | 60.21 | 73.55 |
| PASS | 86.21±1.10 | 91.78±1.12 | 68.90±0.94 | 78.27±0.81 | 66.77±1.18 | 77.01±1.13 | 61.03±0.38 | 70.02±0.56 | 58.34±0.42 | 68.45±1.20 | 68.25 | 77.11 |
| HAT | 83.30±1.54 | 91.06±0.36 | 62.34±0.93 | 73.99±0.86 | 56.72±0.44 | 69.12±1.06 | 57.91±0.72 | 69.38±1.14 | 53.12±0.94 | 65.63±1.64 | 62.68 | 73.84 |
| MORE | **88.13**±1.16 | **93.82**±0.28 | **71.69**±0.11 | **79.83**±0.49 | **71.29**±0.55 | **80.44**±1.42 | **64.17**±0.77 | **72.48**±0.25 | **61.90**±0.90 | **71.46**±0.56 | **71.44** | **79.61** |

lower than our method of 70.53 and 81.59 trained based on OOD detection. Thus, introducing the OOD term in the regularization in MORE improves it drastically.

**Classification Accuracy with Smaller Memory Sizes.** For all the datasets, we run additional experiments with half of the original memory size and show that our method is even stronger with a smaller memory. The new memory sizes are 100, 1000, and 1000 for CIFAR-10, CIFAR-100, and Tiny-ImageNet, respectively. Table 2 shows that MORE has experienced almost no performance loss with reduced memory size while the memory-based baselines suffer from performance reduction. The ACA and AIA values of the best memory-based baseline (DER++) have decreased from 66.89 to 62.16 and 78.46 to 75.40, respectively, while MORE only decreases from 71.59 to 71.44 and 80.77 to 79.61. The little reduction in performance is because a small number of OOD samples is enough to enable the system to produce a more robust network against samples beyond the current task.

**Average Performance Reduction Rate (Backward Transfer)**

We compare the performance reduction rate of our method against the baselines using CIFAR10-5T, CIFAR100-10T, and CIFAR100-20T. Figure 2 shows that the performance of our method drops relatively lower than many baselines. A-GEM and HAT achieve lower reduction rate than our method in all the datasets. However, they are not able to adapt to new tasks well as ACA of A-GEM on the datasets after the final task are 56.33, 25.21, and 21.99, respectively, and those of HAT are 83.30, 62.34, and 56.72. The accuracy of our method on the same datasets are 89.16, 70.23, and 70.53. PASS experiences smaller drop in performance on CIFAR100-10T and CIFAR100-20T than our method, but its ACA of 68.90 and 66.77 are significantly lower than 70.23 and 70.53 of our MORE. For forward and backward transfer, refer to Appendix D.

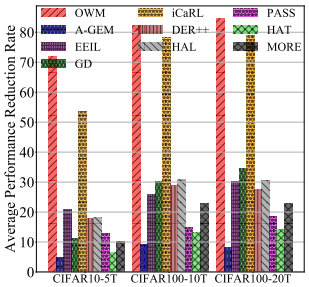


Figure 2: Average performance reduction rate (%). The lower the rate, the better the method is.

As we explained in the introduction section, since our method MORE is based on OOD detection in building each task model, our method can naturally be used to detect test samples that are out-of-distribution for all the classes or tasks learned thus far. We are not aware of any existing continual learning system that has done such an evaluation. We evaluate the performance of the baselines and our system in this out-of distribution scenario, which is also called the open set setting.

Table 3: AUC and incremental AUC (IAUC) after the second last task. The last task is not considered as there is no task after it and thus no OOD data after that. Numbers in bold are the best results in each column.

| Method | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | T-ImageNet-5T | | T-ImageNet-10T | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AUC | IAUC | AUC | IAUC | AUC | IAUC | AUC | IAUC | AUC | IAUC | AUC | IAUC |
| OWM | 58.26±17.38 | 70.02±3.59 | 50.87±2.86 | 63.17±1.06 | 55.43±10.25 | 59.42±1.26 | 58.20±2.51 | 67.24±0.92 | 56.17±4.26 | 62.17±0.35 | 55.79 | 64.41 |
| iCaRL | 78.54±9.59 | 82.12±5.38 | 72.10±2.66 | 77.42±0.45 | 69.79±5.75 | 76.91±1.30 | 66.05±1.73 | 71.86±1.57 | 66.62±1.77 | 74.24±1.66 | 70.19 | 76.06 |
| A-GEM | 63.71±15.18 | 74.92±5.62 | 52.18±2.60 | 64.19±0.86 | 54.78±13.40 | 60.23±0.95 | 58.97±2.52 | 67.88±1.28 | 56.33±4.14 | 63.08±1.12 | 57.19 | 66.06 |
| EEIL | 81.56±10.62 | 87.19±2.31 | 67.39±3.44 | 78.89±1.32 | 64.83±8.01 | 77.69±1.40 | 67.22±2.16 | 74.82±0.79 | 62.36±6.14 | 73.45±1.33 | 68.58 | 78.39 |
| GD | **85.02**±9.88 | **89.71**±1.85 | 64.22±2.47 | 77.31±1.03 | 61.95±9.02 | 75.19±0.87 | 68.35±2.97 | 75.36±0.78 | 58.79±3.10 | 70.90±1.75 | 67.67 | 77.69 |
| DER++ | 79.25±4.74 | 84.61±2.64 | 70.36±1.81 | 78.42±0.64 | 69.74±2.02 | 78.37±0.42 | 68.67±3.83 | 74.80±1.72 | 67.81±0.23 | 74.86±1.93 | 70.93 | 78.09 |
| HAL | 77.97±9.76 | 84.09±3.30 | 69.55±0.83 | 77.37±0.55 | 71.58±3.54 | 77.66±0.31 | 67.58±3.71 | 74.52±1.93 | 67.27±1.86 | 75.47±2.35 | 70.79 | 77.82 |
| PASS | 77.69±4.01 | 84.57±1.54 | 71.80±2.41 | 77.74±1.40 | 66.62±5.78 | 77.42±1.44 | 71.61±1.10 | 77.07±2.14 | 68.51±4.49 | 74.79±2.36 | 71.24 | 78.32 |
| HAT | 83.89±4.10 | 87.83±2.44 | 71.26±1.93 | 79.57±0.29 | 65.52±3.43 | 77.20±0.74 | 75.08±1.07 | 79.78±1.59 | 72.02±1.35 | 78.25±1.68 | 73.55 | 80.53 |
| MORE | 80.83±8.82 | 88.06±1.84 | **73.32**±2.80 | **81.67**±1.27 | **72.28**±4.81 | **80.97**±0.80 | **75.74**±2.66 | **80.72**±3.38 | **72.78**±1.08 | **79.73**±2.97 | **74.99** | **82.23** |

**Out-of-Distribution Detection Results.** This OOD detection ability is highly desirable for a continual learning system because in a real-life environment, the system can be exposed to not only seen classes, but also unseen classes. When the test sample is from one of seen classes, the system should be able to predict its class. If the sample does not belong to any of the training classes seen so far (i.e., the sample is out-of-distribution), the system should detect it.

We formulate the performance of OOD detection of a continual learning system as the following. A continual learning system accepts and classifies a test sample after training task $k$ if the test sample is from one of the classes in tasks $1, \cdots, k$. If it is from one of the classes of the future tasks $k+1, \cdots, t$, it should be rejected as OOD (where $t$ is the last task in each evaluation dataset).

We use maximum softmax probability (MSP) (Hendrycks & Gimpel, 2016) as the OOD score of a test sample for the baselines and use maximum output with coefficient in Eq. 7 for our method. We employ Area Under the ROC Curve (AUC) to measure the performance of OOD detection as AUC is the standard metric used in OOD detection papers (Yang et al., 2021). We report the AUC after the second last task. We do not consider the AUC after the last task because there is no more continual learning task (and thus, no OOD) after the final task. We also report incremental AUC (IAUC) which is the average of AUC over all the tasks.

Table 3 shows that our method consistently outperforms the baselines in all the datasets. The best baselines based on ACA and AIA are DER++ and PASS. Their average AUC and IAUC over the experiments are 70.93 and 78.09 for DER++ and 71.24 and 78.32 for PASS. Our method MORE achieves 74.99 and 82.23. The performance gain in OOD detection problem is not because of better performance in classification. For example, on CIFAR100-10T and CIFAR100-20T, the AIA of DER++ is slightly better than MORE in Table 1 with memory size 2000. However, in IAUC, DER++ achieves 78.42 and 78.37 for CIFAR100-10T and 20T, respectively, while MORE obtains 81.67 and 80.97. The improvement is due to OOD detection in training.

### 4.4 Ablation Study

We conduct an ablation study to measure the performance gain by each proposed technique: back-update in previous models in Sec. 3.2 and the distance-based coefficient in Sec. 3.3 using three experiments. The back-update by Eq. 3 is to improve the earlier task models as they are trained with less diverse OOD data than later models. The modified output with coefficient in Eq. 4 is to improve the classification accuracy by assembling or combining the softmax probability of the task networks and the inverse Mahalanobis distances.

Table 4 compares AIA obtained after applying each method. Both distance based coefficient and back-update show large improvements from the original method without any of the two techniques. Although the performance is already competitive with either technique, the performance improves further after applying them together.

As discussed in Sec. 3.2, the earlier tasks are trained with less diverse OOD samples than later tasks because the samples in memory are considered as OOD data in training. As a result, the models suffer from over-confidence problem in later tasks, which is a phenomenon that a neural network produces a high score for OOD instance. For example, in our experiment with CIFAR100-20T, we found that after training the last task, 73% of incorrectly classified test instances are classified as one of the classes in earlier tasks than the true tasks while only 27% of incorrectly classified samples are predicted as one of the classes

Table 4: Performance gains with the proposed techniques. The row Original indicates the method without the coefficient and back-update and the row Back means the back-update method.

| | CIFAR10-5T | CIFAR100-10T | CIFAR100-20T |
|---|---|---|---|
| Original | 91.01±2.48 | 76.93±1.58 | 75.76±2.35 |
| Coefficient (C) | 93.86±1.12 | 80.31±1.02 | 80.77±1.36 |
| Back (B) | 93.36±0.79 | 80.35±1.08 | 80.32±0.82 |
| C + B | 94.23±0.82 | 81.24±1.24 | 81.59±0.98 |

Table 5: Average classification accuracy (ACA) and average incremental accuracy (AIA) after the final task. '-XT' means X number of tasks. The results are based on a pre-trained model using 200 randomly selected classes from the 611 classes of ImageNet after removing 389 classes similar to the classes in CIFAR10/100 and Tiny-ImageNet. Our system MORE and all baselines used the pre-trained network. The last two columns show the average ACA and AIA of each method over all datasets and experiments. We highlight the best results in each column in bold.

| Method | CIFAR10-5T | | CIFAR100-10T | | CIFAR100-20T | | T-ImageNet-5T | | T-ImageNet-10T | | Average | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA | ACA | AIA |
| OWM | 30.04±9.98 | 49.88±3.67 | 11.51±5.63 | 30.78±1.75 | 17.68±4.66 | 24.28±0.95 | 8.57±3.85 | 33.92±1.70 | 10.24±5.18 | 26.47±4.47 | 15.61 | 33.07 |
| iCaRL | 77.51±0.77 | 85.87±1.13 | 59.83±0.64 | 69.79±1.24 | **58.27**±0.23 | 70.00±1.14 | 42.85±0.77 | 53.30±0.90 | 42.11±0.57 | 54.61±1.29 | 56.11 | 66.71 |
| A-GEM | 47.70±0.98 | 62.04±1.87 | 16.15±2.00 | 34.67±0.57 | 9.78±0.64 | 24.20±0.39 | 21.26±1.03 | 39.71±0.67 | 13.56±2.44 | 30.13±1.06 | 21.69 | 38.15 |
| EEIL | 67.50±4.46 | 81.42±0.60 | **59.41**±0.36 | **74.48**±0.72 | 55.17±0.83 | **72.83**±0.91 | 43.10±0.47 | 57.12±0.41 | 40.63±0.59 | 56.76±0.53 | 53.16 | 68.52 |
| GD | **80.44**±0.86 | **88.99**±0.85 | 55.88±0.71 | 73.89±0.60 | 50.42±0.52 | 70.41±0.76 | 43.01±0.90 | 58.45±0.56 | 32.47±1.78 | 54.28±0.54 | 52.44 | 69.21 |
| DER++ | 69.11±1.24 | 81.60±1.33 | 56.02±0.73 | 71.77±0.53 | 56.22±0.41 | 72.08±0.83 | 40.67±1.17 | 54.36±0.37 | 38.51±2.15 | 53.78±0.21 | 52.11 | 66.72 |
| HAL | 68.13±2.63 | 80.10±1.33 | 53.56±0.75 | 67.54±0.55 | 53.56±1.11 | 66.86±0.82 | 38.52±2.13 | 54.54±0.75 | 35.98±3.57 | 50.75±0.85 | 49.95 | 63.96 |
| PASS | 76.92±0.60 | 85.98±1.30 | 57.67±0.64 | 69.72±0.77 | 54.64±0.83 | 67.41±1.01 | 48.51±0.54 | 58.92±0.85 | 45.01±0.53 | 56.77±1.17 | 56.55 | 67.76 |
| HAT | 67.50±3.75 | 80.33±1.06 | 51.08±0.23 | 64.22±0.72 | 46.84±0.57 | 61.54±1.16 | 50.54±0.77 | 60.80±1.16 | 45.68±0.54 | 57.91±1.34 | 52.33 | 64.96 |
| MORE | 72.56±6.41 | 85.23±3.23 | 58.68±1.90 | 70.78±0.65 | 58.02±0.66 | 69.79±1.34 | **51.75**±0.32 | **61.86**±0.66 | **48.00**±0.89 | **58.96**±0.68 | **57.80** | **69.32** |

in later tasks. The average accuracy is 66.19. After updating the previous networks using the back-update method, equal proportion (49.72%) of incorrectly classified samples are predicted to either earlier or later tasks and the accuracy increases to 70.53. The back-update helps eliminate the asymmetric performance issue in the task models and improves the classification accuracy.

### 4.5 Additional Experiments

**Number of Pre-Training Classes.** We study the effect of the number of classes of the data used in pre-training the feature extractor. For the above experiments, we used 611 classes of ImageNet after removing 389 classes similar to the classes in CIFAR10/100 and Tiny-ImageNet to pre-train the feature extractor. Here, we randomly select 200 classes from the 611 classes and use the data to pre-train a feature extractor. The results are reported in Table 5. The average ACA and AIA (last column) over 5 experiments show that our method MORE still outperforms the baselines. Despite that some baselines outperform MORE in some cases, the differences are small. For the more challenging dataset, Tiny-ImageNet, MORE is markedly superior to the baselines.

**Performance of Pre-Trained Feature Extractor.** We provide the performance of the pre-trained feature extractor to measure how much value the proposed method add to the feature extractor. We fix the pre-trained feature extractor and construct a classifier on top of the feature representations of the training data. The considered classifiers are nearest class mean classifier (NCM) and cosine similarity classifier (CSC). We use the mean (i.e., prototype) of the class features to represent each class in the classifier. Given a test instance, we use Euclidean distance (or cosine similarity) for NCM (for CSC) between the test sample and the prototypes, and choose the class that gives the minimum distance (or maximum similarity). Since these classifiers incrementally add new prototypes without training the classifier or feature extractor, they are not continual learning methods, but are appropriate to evaluate the performance of a pre-trained feature extractor. We report the results of CIFAR10, CIFAR100, and Tiny-ImageNet by using all the samples in a single task. NCM achieves 85.80, 58.94, and 54.75 on the three datasets, respectively. CSC achieves 85.93, 58.97, 54.75. On the other hand, our method MORE achieves 89.16, 70.23/70.53, and 64.97/63.06 on CIFAR10-5T, CIFAR100-10T/20T, and T-ImageNet-5T/10T, respectively. Our MORE is considerably better, which shows that despite the strong feature extractor, one can achieve much better result by training a model with our MORE method than simply building a classifier on top of the pre-trained feature extractor.

## 5 Conclusion

This paper proposes a novel approach MORE for class incremental learning (CIL). Unlike the existing memory-based CIL methods that use a single-head model and leveraging the memory buffer to mitigate forgetting, the proposed method uses a multi-head model and a memory buffer for building a task specific OOD model for each task which can identify the class of a sample from the task and also detect OOD samples. Since the parameters are effectively protected by task identifiers in the multi-head setting through hard attentions, the network successfully adapts to new knowledge with little reduction in performance. The resulting network outperforms the state-of-the-art baselines in terms of standard continual learning metrics and also shows strong performance in OOD detection in the continual learning setting.

## ACKNOWLEDGMENTS

This work was supported in part by a research contract from KDDI, two National Science Foundation (NSF) grants (IIS-1910424 and IIS-1838770), a DARPA contract HR001120C0023, and a Northrop Grumman research gift.

## 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.

Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Caccia. Online continual learning with maximal interfered retrieval. In *NeurIPS*. 2019a.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019b.

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

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, 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, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

Arslan Chaudhry, Naeemullah Khan, Puneet Dokania, and Philip Torr. Continual learning in low-rank orthogonal subspaces. *Advances in Neural Information Processing Systems*, 33:9900–9911, 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.

Zhiyuan Chen and Bing Liu. *Lifelong machine learning*. Morgan & Claypool Publishers, 2018.

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

Yidou Guo, Bing Liu, and Dongyan Zhao. Online continual learning through mutual information maximization. In *ICML*, 2022.

Tyler L. Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pp. 466–483. Springer, 2020.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

Christian Henning, Maria Cervera, Francesco D'Angelo, Johannes Von Oswald, Regina Traber, Benjamin Ehret, Seijin Kobayashi, Benjamin F Grewe, and João Sacramento. Posterior meta-replay for continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

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.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 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.

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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Gyuhak Kim and Bing Liu. Continual learning via principal components projection, 2020. URL https://openreview.net/forum?id=SkxlElBYDS.

Gyuhak Kim, Sepideh Esmaeilpour, Changnan Xiao, and Bing Liu. Continual learning based on ood detection and task masking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3856–3866, June 2022.

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.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=UYneFzXSJWh.

Y. Le and X. Yang. Tiny imagenet visual recognition challenge, 2015.

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

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

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

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.

Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation. *arXiv preprint arXiv:2010.15277*, 2020.

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.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Fmg_fQYUejf.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11321–11329, 2019.

Jathushan Rajasegaran, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. itaml: An incremental task-agnostic meta-learning approach. In *CVPR*, 2020.

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

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

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

Ryne Roady, Tyler L. Hayes, Hitesh Vaidya, and Christopher Kanan. Stream-51: Streaming classification and novelty detection from videos. In *CVPR Workshops*, 2020.

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.

Gobinda Saha, Isha Garg, and Kaushik Roy. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*, 2021.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

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

James Smith, Jonathan Balloch, Yen-Chang Hsu, and Zsolt Kira. Memory-efficient semi-supervised continual learning: The world is its own replay buffer. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

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

Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):1–14, 2020.

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

Mitchell Wortsman, Vivek R., R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/file/ad1f8bb9b51f023cdc80cf94bb615aa9-Paper.pdf.

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

Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3014–3023, 2021.

Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021.

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.

Li Zhaoping and Zhaoping Li. *Understanding vision: theory, models, and data*. Oxford University Press, USA, 2014.

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 Pseudo-Code

For task $k$, Let $p(y|\boldsymbol{x}, k) = \text{softmax} f(h(\boldsymbol{x}, k; \theta, \boldsymbol{e}^k); \phi_k)$, where $\theta$ is the parameters for adapter, $\boldsymbol{e}^k$ is the trainable embedding for hard attentions, and $\phi_k$ is the set of parameters of the classification head of task $k$. Algorithm 1 and Algorithm 2 describe the training and testing processes, respectively. We add comments with the symbol "//".

**Algorithm 1** Training MORE

**Require:** Memory $\mathcal{M}$, learning rate $\lambda$, a sequence of tasks $\mathcal{D} = \{\mathcal{D}^k\}_{k=1}$, and parameters $\{\theta, \boldsymbol{e}, \phi\}$, where $\boldsymbol{e}$ and $\phi$ are collections of task embeddings $\boldsymbol{e}^k$ and task heads $\phi_k$

// CL starts

**for** each task data $\mathcal{D}^k \in \mathcal{D}$ **do**

  // Model training

  **for** a batch $(\boldsymbol{X}_i^k, \boldsymbol{y})$ in $\mathcal{D}^k$, until converge **do**

    $\boldsymbol{X}_s$ = $sample(\mathcal{M})$

    Compute $\mathcal{L}$ of Eq. 13 and gradients of parameters

    Modify the model parameters $\nabla\theta \leftarrow \nabla\theta'$ using Eq. 11

    Update parameters as $\theta \leftarrow \theta - \lambda\nabla\theta,\ \boldsymbol{e}^k \leftarrow \boldsymbol{e}^k - \lambda\partial\mathcal{L},\ \phi_k \leftarrow \phi_k - \lambda\partial\mathcal{L}$

  **end for**

  // Back-updating in Sec. 3.2

  Randomly select $\tilde{\mathcal{D}} \subset \mathcal{D}^k$, where $|\tilde{\mathcal{D}}| = |\mathcal{M}|$

  **for** each task $j$, until converge **do**

    minimize $\mathcal{L}(\phi_j)$ of Eq. 3

  **end for**

  // Obtain statistics in Sec. 3.3

  Compute $\boldsymbol{\mu}_j^k$ using Eq. 5 and $\boldsymbol{S}^k$ using Eq. 6

**end for**

**Algorithm 2** MORE Prediction

**Require:** Test instance $\boldsymbol{x}$ and parameters $\{\theta, \boldsymbol{e}, \phi\}$

**for** each task $k$ **do**

  Obtain $p(\mathcal{Y}^k|\boldsymbol{x}, k)$

  Obtain $s^k(\boldsymbol{x})$ using Eq. 4

**end for**

// Concatenate outputs for final prediction $y$ and OOD score $s$

$y = \arg\max \bigoplus_{1\leq k\leq t} p(\mathcal{Y}^k|\boldsymbol{x}, k)s^k(\boldsymbol{x})$ (i.e. Eq. 7)

$s = \max \bigoplus_{1\leq k\leq t} p(\mathcal{Y}^k|\boldsymbol{x}, k)s^k(\boldsymbol{x})$

Table 6: Total memory (in entries) required for each method without the replay memory buffer.

| Method | CIFAR10-5T | CIFAR100-10T | CIFAR100-20T | T-ImageNet-5T | T-ImageNet-10T |
|---|---|---|---|---|---|
| OWM | 26.6M | 28.1M | 28.1M | 28.2M | 28.2M |
| iCaRL | 22.9M | 24.1M | 24.1M | 24.1M | 24.1M |
| A-GEM | 26.5M | 31.4M | 31.4M | 31.5M | 31.5M |
| EEIL | 22.9M | 24.1M | 24.1M | 24.1M | 24.1M |
| GD | 22.9M | 24.1M | 24.1M | 24.1M | 24.1M |
| DER++ | 22.9M | 24.1M | 24.1M | 24.1M | 24.1M |
| HAL | 22.9M | 24.1M | 24.1M | 24.1M | 24.1M |
| PASS | 22.9M | 24.2M | 24.2M | 24.3M | 24.4M |
| HAT | 23.0M | 24.7M | 25.4M | 24.6M | 25.1M |
| MORE | 23.7M | 25.9M | 27.7M | 25.1M | 25.9M |

## B Large Distribution Shift from Pre-Training to CL datasets

This section evaluates model performance when the distribution between the dataset used for pre-training and the dataset used in continual learning is very different. Kumar et al. (2022) have showed that fine-tuning does not work well for OOD detection if this data distribution shift is large. We use street view house numbers (SVHN) (Netzer et al., 2011) to show the effect of a large distribution shift from the pre-training dataset. Our feature extractor is pre-trained with the ImageNet dataset, which does not contain any class of data similar to house numbers.

We split the SVHN dataset into 5 tasks, where each task consists of 2 classes. We train DER++ and PASS, the two best performing baselines in our experiment. The average classification accuracy values after the final task are 68.19, 76.09 for DER++ and PASS, respectively. Our method achieves 80.57. Despite the fact that the distribution of the pre-training dataset and the continual learning dataset are very different, our method still outperforms them.

## C Size of Memory Required

In this section, we report sizes of memory required by each method. The sizes include network size, replay buffer, all other parameters or example kept in memory simultaneously for a model to be functional.

We use an 'entry' to refer to a parameter or element in a vector or matrix to calculate the total memory required to train and test. The pre-trained backbone uses 21.6 million (M) entries (parameters). The adapter modules use 1.2M entries for CIFAR10 and 2.4M for other datasets. The baselines and our method use 22.8M and 24.0M entries for the model on CIFAR10 and other datasets, respectively. The unique technique of each method may add additional entries for training and test/inference.

The total memory required for each method without considering the replay memory buffer is reported in Table 6. Our method is competitive in memory consumption. Baselines such as OWM and A-GEM take considerably more memory than our system. iCaRL and DER++ take the least amount of memory, but the differences between our method and theirs are only 0.8M, 1.8M, 3.6M, 1.0M, and 1.8M for CIFAR10, CIFAR100-10T, CIFAR100-20T, T-ImageNet-5T and T-ImageNet-10T.

Many replay based methods (e.g., iCaRL, HAL) need to save the previous network for distillation during training. This requires additional 1.2M or 2.4M entries for CIFAR10 or other datasets. Our method does not save the previous model as we do not use distillation.

Note that a large memory consumption usually comes from the memory buffer as the raw data is of size 32*32*3 or 64*64*3 for CIFAR and T-ImageNet. For memory buffer of size 2000, a system needs 6.1M or 24.6M entries for CIFAR or T-ImageNet. Therefore, saving a smaller number of samples is important for reducing the memory consumption. As we demonstrated in Table 1 and Table 2 in the main paper, our method performs better than the baselines even with a smaller memory buffer. In Table 1, we use large memory sizes (e.g., 200 and 2000 for CIFAR10 and other datasets). In Table 2, we reduce the memory size by half. When we compare the accuracy of our method in Table 2 to those of the baselines in Table 1, our method still outperforms them on all datasets. Our method with a smaller memory buffer achieves average classification accuracy of 88.13, 71.69, 71.29, 64.17, 61.90 on CIFAR10-5T, CIFAR100-10T, CIFAR100-20T, T-ImageNet-5T, and T-ImageNet-10T. On the other hand, the best baselines achieve 88.98, 69.73, 70.03, 61.03, 58.34 on the same experiments with a larger memory buffer.

## D FORWARD AND BACKWARD TRANSFER

We follow (Ke et al., 2020) for forward transfer (FWT) and (Lopez-Paz & Ranzato, 2017) for backward transfer (BWT). FWT is basically the average performance difference on each task between a CL method when the task is first learned and a trained individual model from random initialization. By definition, the higher the FWT, the better the transfer is. For our experiments, we train the pre-trained model with adapters instead of a randomly initialized model. The FWTs of the three best baselines (iCaRL, DER++, PASS) and our method on CIFAR100-10T are -19.04, -1.4, -13.79, and -5.28. The FWTs on Tiny-ImageNet are -17.33, 2.96, -11.18, and -10.12. DER++ shows the best FWT, but its final accuracy values are lower than MORE due to bad backward transfer or forgetting (see below).

The BWT in (Lopez-Paz & Ranzato, 2017) is the same as the performance reduction rate that we report in the paper. We report the exact values here. The BWT of the three best baselines (iCaRL, DER++, PASS) and our method are -8.32, -26.51, -18.37, and -19.75 on CIFAR100-10T. The BWT on Tiny-ImageNet-5T are -10.69, -25.70, -7.14, -10.96. iCaRL and PASS show better BWT (lower reduction in performance from the initial accuracy) than MORE. However, MORE achieves better average classification accuracies than the baselines as discussed in Sec. 4.3. These baselines are good at preserving the past knowledge, but their learning methods are weak at learning each task, or vice versa. Our method shows good balance between FWT and BWT and achieve the best average accuracy.