Online Continual Learning through Mutual Information Maximization

Yidou Guo [1 2] **Bing Liu** [3] **Dongyan Zhao** [1 2]

Abstract

This paper proposes a new online continual learning technique called OCM based on *mutual information maximization*. It achieves two objectives that are critical in dealing with *catastrophic forgetting* (CF). (1) It reduces feature bias caused by cross entropy (CE) as CE learns only discriminative features for each task, but these features may not be discriminative for another task. To learn a new task well, the network parameters learned before have to be modified, which causes CF. The new approach encourages the learning of each task to make use of *holistic representations* or the full features of the task training data. (2) It encourages preservation of the previously learned knowledge when training a new batch of incrementally arriving data. Empirical evaluation shows that OCM substantially outperforms the online CL baselines. For example, for CIFAR10, OCM improves the accuracy of the best baseline by 13.1% from 64.1% (baseline) to 77.2% (OCM). The code is publicly available at https://github.com/gydpku/OCM.

1. Introduction

Continual learning (CL) incrementally learns a sequence of tasks $< 1, 2, ..., t, ... >$. Each task consists of a set of classes to be learned. Learning can be done in the *batch* mode and the *online* mode. In batch CL, when a new task t arrives, all its training data is available and training may go through the data any number of epochs. In online CL, the data for each task comes gradually in a stream and learning is done whenever a small batch of training examples is accumulated. Due to the fast speed of the data stream, for each data batch, the learning algorithm can see it only once like one training iteration. It is well-known that CL suffers from *catastrophic forgetting* (CF) (McCloskey & Cohen, 1989), i.e., the training of a new task needs to update the existing network parameters, which may cause accuracy degradation of the tasks learned earlier. There are also three CL settings. This paper works in the challenging setting of *class incremental learning* (CIL), in which only one model is incrementally learned for all classes of the tasks seen so far.[1] In testing, the model can classify a test instance from any class without the task information. *This paper focuses on dealing with CF in the CIL setting of the online CL mode.*

One popular approach for dealing with CF is *experience replay* or simply *replay*. In this approach, a small subset of the previous training data is saved in a *memory buffer*. In learning a new task (batch CL) or a small batch (online CL), the system trains on both the new data and the saved data to re-adjust the features and decision boundaries between both the old and the new classes. We will discuss the difference between replay approaches in batch and online CL in Sec. 2. Almost all online CL methods are based on replay. This work uses the replay approach as well. However, we propose a more principled solution based on *mutual information* (MI) *maximization*. The proposed approach is called OCM (*Online Continual learning based on Mutual information maximization*), which deals with CF in three ways.

(1). It prevents information loss in feature learning. In learning a task, *cross entropy* (CE) loss learns only *discriminative* (or *biased*) features that can separate the classes of the task. However, these discriminative features may not be sufficiently discriminative for a new task. To learn the new task well, the model parameters have to be modified, which causes CF for previous tasks. To deal with this, the system should learn ***holistic representations*** of the input data. The proposed solution precisely encourages learning all features (i.e., holistic representations) of the input data for each task. Then features that may not be discriminative for the current task but are useful for some future tasks are also learned.

(2). It encourages preservation of the previously learned knowledge when learning a new batch of data X^{new} from an existing task or a new task. This is also important for preventing CF. We note that without achieving (1), (2) will

[1]Wangxuan Institute of Computer Technology, Peking University. [2]Artificial Intelligence Institute, Peking University. [3]Department of Computer Science, University of Illinois at Chicago. Correspondence to: Bing Liu <liub@uic.edu>, Dongyan Zhao <zhaody@pku.edu.cn>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

[1]See (Ke et al., 2021; van de Ven & Tolias, 2019) for *task incremental learning* and *domain incremental learning* settings.

not fully realize its benefit because ***preserving the biased past*** may result in sub-optimal solutions.

(3). Theoretical analysis suggests a new training strategy on how to use the replay data (denoted by X^{buf}) and the new data batch X^{new} in training. The analysis also shows that data augmentation can be applied to help learning. A new augmentation called ***local rotation*** is proposed as well.

To our knowledge, ***MI maximization*** has not been used as a solution method to overcome CF in either batch or online CL. The paper also introduces the concept of ***holistic representation*** and shows why cross entropy loss causes CF due to the biased features that it learns. It then demonstrates that (i) OCM based on MI maximization is highly effective for online CL, and (ii) MI maximization can also be used to improve existing online CL methods.

Empirical evaluation using benchmark datasets MNIST, CIFAR10, CIFAR100 and TinyImageNet shows that OCM outperforms the state-of-the-art online CL systems markedly. For example, with the memory buffer size of $1k$, OCM outperforms the best baseline by 13.1% on CIFAR10.

2. Related Work

Batch continual learning (batch CL). Many approaches have been proposed for batch CL. Using *regularizations* to mitigate CF by penalizing changes to important parameters learned in the past is a popular approach (Kirkpatrick et al., 2017; Ritter et al., 2018; Ahn et al., 2019; Yu et al., 2020; Zhang et al., 2020). *Replay* is another popular approach (Chaudhry et al., 2020; Castro et al., 2018; Rebuffi et al., 2017; de Masson d'Autume et al., 2019; Wu et al., 2019; Hou et al., 2019; Zhao et al., 2021; Korycki & Krawczyk, 2021; Sokar et al., 2021; Yan et al., 2021; Hu et al., 2021; Wang et al., 2022), which uses a memory buffer to store some previous data and replays it in training a new task. Data generators have also been learned to generate pseudo-replay data (Shin et al., 2017; Hu et al., 2019). This approach is not suitable for online CL as it needs the full data to learn data generators. Many recent approaches are based on *dynamic architectures* (Ostapenko et al., 2019; von Oswald et al., 2020; Serrà et al., 2018; Rajasegaran et al., 2020; Abati et al., 2020; Saha et al., 2021; Yoon et al., 2018; Li et al., 2019; Hung et al., 2019; Rajasegaran et al., 2019; Farajtabar et al., 2020), which expand the network or isolate parameters to prevent forgetting. As an extension, **CAT** (Ke et al., 2020) uses a parameter isolation approach to deal with both CF and knowledge transfer across tasks. **CLOM** (Kim et al., 2022) combines parameter isolation and out-of-distribution detection for continual learning. Another approach is orthogonal projection (Zeng et al., 2019; Guo et al., 2022). **AOP** (Guo et al., 2022) can perform both batch CL and online CL. **IL2A** (Zhu et al., 2021) investigates feature representations based on spectral analysis and presents a data augmentation called ClassAug to learn more transferable features.

Online continual learning (online CL). Online CL methods are mainly based on replay except **AOP** (Guo et al., 2022), which uses a pre-trained model. **ER** (Chaudhry et al., 2020) randomly samples the replay data. **MIR** (Aljundi et al., 2019a) chooses replay samples whose losses increase most. **GSS** (Aljundi et al., 2019b) diversifies the gradients of the samples in the memory buffer and allows the model to learn more information from the memory buffer. **ASER** (Shim et al., 2021) uses the Shapley value theory. **DER++** (Buzzega et al., 2020) utilizes the dark knowledge (Hinton et al., 2014) and knowledge distillation. **SCR** (Mai et al., 2021) uses supervised contrastive loss for representation learning and the nearest class mean for classification. **GDumb** (Prabhu et al., 2020) greedily samples the data and ensures the classes are balanced. It uses all the samples in the memory to train the learner. **NCCL** (Yin et al., 2021) calibrates the network to mitigate CF. It is designed for task incremental learning. Yan et al. (2021) proposed an EM method for online CL in semantic segmentation. Wang et al. (2021) proposed to use online CL in object detection. The proposed **OCM** focuses on online CL for classification. It is also based on replay, but very different from existing approaches as its solution approach is MI maximization.

Differences between replay methods of batch CL and online CL: Before discussing the differences, we first define two terms, *inter-task CF* and *intra-task CF*. Inter-task CF refers to the commonly studied CF, i.e., learning a new task causing forgetting of the knowledge learned from previous tasks. Intra-task CF refers to forgetting of the knowledge learned from early batches by later batches within a task. Intra-task CF occurs only in online CL because online CL sees the data only once or trains in one epoch. The main differences between replay methods of batch CL and online CL are: (1) batch CL has only inter-task CF but online CL has both because batch CL trains each task many epochs. (2) To deal with inter- and intra-task CF, online CL usually samples replay data continuously from both previous tasks and the current task, but batch CL samples and saves some samples from only previous tasks. (3) In batch CL, when a task arrives, all its training data is available, but for online CL, the data comes gradually. Due to these differences, online CL needs different replay methods.

Mutual information (MI): MI has many machine learning (ML) applications. For example, Boudiaf et al. (2020) used MI to bridge cross-entropy and pairwise losses. They also proposed a method for few-shot learning by maximizing the MI between queries and their label predictions. However, MI is hard to estimate. They designed an estimator from the Donsker-Varadhan representation and the f-divergence

representation. Oord et al. (2018) tried to solve the problem by optimizing an InfoNCE-type lower bound of MI. To our knowledge, MI has not been used in CL, which needs unique formulations to guide the training to overcome CF.

3. Problem Description and Preliminaries

In online CL, we learn a sequence of tasks incrementally. Each task t has its dataset $D_t = \{(x_i, y_{x_i})\}_{i=1}^{n_t}$, where x_i is an input sample and y_{x_i} is its class label ($y_{x_i} \in Y_t$, the set of all labels of task t) and n_t is the number of training samples. The training data for each task t comes gradually in a stream. Following existing online CL works (Aljundi et al., 2019a), whenever a small batch of data (denoted by X^{new} with N samples) from task t is accumulated from the data stream, it is trained in one iteration. After all the data of a task are seen, the next task starts. In a replay method, a mini-batch used in training consists of X^{new} and X^{buf}, where X^{buf} of size N_b is sampled from the memory buffer $\mathcal{M}$. $\mathcal{M}$ saves a small set of training samples of seen tasks. Note that, before seeing all the data of the current task t, $\mathcal{M}$ has already saved some data from task t sampled from the data stream of task t seen so far.

Our neural model has two parts: the feature extractor f_θ and the classifier σ. For an input x, we get its logits from: $o(x) = \sigma(f_\theta(x))$ to calculate the loss or to predict the label.

Cross Entropy (CE). For supervised learning, CE is the commonly used loss function. However, as we state earlier, CE has a major shortcoming for CIL (see the second remark below about the generalisation), which can be stated with the following two propositions.

Proposition 1. Minimal CE does not imply that all possible features of a class/task are learned in the feature extractor.

Let us assume that minimal CE implies that all features of a class are learned. We use an example to show that it is not true. Let us say that we want to learn to recognize two kinds of animals: cat and dog. Assume in the training data all the dogs are black and all the cats are white. We further assume that by chance the feature extractor learned only two significant features in the feature vector, one for black and one for white. When a black object is presented to the feature extractor, the first feature has the value of 1 and all other features have the value of 0. When a white object is presented to the feature extractor, the second feature has the value of 1 and the rest of the features have the value of 0. In this case, with proper weights in the classification/linear layer, CE loss can easily achieve 0 (minimal). The learned model is sufficient for classifying black dogs and white cats. However, cats and dogs clearly have many other features.

Proposition 2. Features not learned may cause CF in continual learning.

Without loss of generality, we can assume that only a subset of features is learned from task 1. But some of those features that are not learned may be necessary for distinguishing the classes of task 2. To learn these features, the feature extractor learned from task 1 has to be updated, which causes CF for classes in task 1. Following the example above, if task 2 is to learn to recognize chicken and pig, and they come in many colors. The two color based features above will not be sufficient. The existing feature extractor needs to be updated for task 2, which results in CF.[2]

Remarks. We make three remarks about the above analysis.

(1) It is difficult, if not impossible, to define or to learn *the full set of features* of a class/task. It is also difficult to define the *degree of holisticness*. This paper argues that if more features are learned in the representation, the amount of forgetting will be reduced. This is because with more features, the learning of future tasks will need to update less of the learned feature extractor. One approximate way to compare the holisticness of two different representation learning methods is through computing the eigenvalues and eigenvectors of the resulting representations, which we discuss in Section 6.5. The proposed method uses MI maximization to learn more features, i.e., to learn more holistic representations,[3] and also to preserve the model learned previously.

(2). Although the analysis above is based on cross entropy, it is also applicable to other existing loss functions for supervised learning because the current supervised learning paradigm does not require all the information of the dataset.

(3). The above analysis is not meant for *task incremental learning* (TIL) because in TIL each task may build a separate model and the parameters of the model may be protected by masks in a parameter isolation approach (Serrà et al., 2018; Wortsman et al., 2020), which can effectively eliminate CF. In testing, the task-id is provided for each test instance to identify the right model to use.

4. MI Maximization for Continual Learning

This section presents the proposed technique OCM based on *mutual information* (MI) *maximization*, which gives us a novel training objective and training strategy.

The popular loss function used in classification is the cross-entropy loss $\mathcal{L}_{ce}$. But as we discussed above, $\mathcal{L}_{ce}$ learns only discriminative and biased features to separate the classes of a task. The other features that may be transferable and

[2] Although the replay data can help the first task model to adapt, since the amount of replay data is limited, it is often not sufficient to address the CF issue.

[3] Another way to achieve more holistic representations is through pre-training using a large number of prediction classes, which is what most pre-training techniques have been doing.

useful in classifying classes in possible future tasks are ignored, which causes CF when learning the future tasks. The proposed MI based solution aims to deal with this problem.

4.1. Maximization of MI between Input X and $f_\theta(X)$

One way to solve the problem is to ensure that the learned feature representation $f_\theta(X)$ is *holistic* in the sense that it preserves the features of input X as much as possible to reduce feature bias (see Sec. 6.5 for the *measurement of holisticsness*). We propose to maximize the MI between X and $f_\theta(X)$: $I(X; f_\theta(X))$, as mutual information can detect general, possibly non-linear, relationships. We use an additional head Φ (e.g., a linear layer) to calculate $\Phi(f_\theta(X))$ and maximize $I(X; \Phi(f_\theta(X)))$ rather than to maximize $I(X; f_\theta(X))$ directly. $\Phi(f_\theta(x))$ creates a new parameter space for representation learning that has a much fewer dimensions than $f_\theta(x)$. Due to the Markov chain $X \rightarrow f_\theta(X) \rightarrow \Phi(f_\theta(X))$ and the data processing inequality (Beaudry & Renner, 2011), we have

$$I(X; f_\theta(X)) \geq I(X; \Phi(f_\theta(X))) \tag{1}$$

So what we optimize directly is a lower bound for $I(X; f_\theta(X))$. To simplify the presentation, we use $F(X)$ to denote $normalization(\Phi(f_\theta(X)))$, which projects the representations of all inputs to an unit sphere.

In the online CL context, each iteration with data $X^{new} \bigcup X^{buf}$ should maximize

$$I(X^{new} \bigcup X^{buf}; F(X^{new} \bigcup X^{buf})) \tag{2}$$

However, if the class labels Y are considered, this optimization can be revised, which we discuss next.

Maximization of MI between $F(X)$ and Label Y. Since we work in the supervised learning setting, making the best use of the label information is vital for model performance. Because if the learned representation $F(X)$ includes enough supervised label information, then clear class boundaries in the feature space would be naturally generated. However, MI $I(F(X); Y)$ is hard to compute as they have very different dimensions. Here we only analyse it and use the analysis to guide the design of our training strategy and loss to solve the problem. We rewrite $I(F(X); Y)$ as:

$$I(F(X); Y) = H(Y) - H(Y|F(X)) \tag{3}$$

where $H(X)$ is the entropy of the label variable Y. To maximize this MI, entropy $H(Y)$ should be maximized (the first term) and the label should be easily identified from the feature representation (the second term, which is naturally minimized in classification training). Maximizing $H(Y)$ means that variable Y should be balanced or follow uniform distribution, which gives the maximum value for $H(Y)$. This has two *implications*: **(1)** We should not maximize Eq. 2 directly as it may lead to class imbalance of Y (the classes in X^{new} may have a higher proportion). Since the classes in X^{new} or X^{buf} are approximately balanced[4] , we thus use $I(X^{new}; F(X^{new})) + I(X^{buf}; F(X^{buf}))$ as the surrogate to maximize the MI between the input and the feature representation, Eq. 2. **(2)** We should calculate only the cross-entropy loss of X^{buf} and use it to update the classifier. This is because the data randomly sampled from the task-free (i.e., no task-id used) buffer can be viewed as the samples from an approximately uniform distribution of all seen classes. Note that since the replay buffer is incrementally updated, X^{buf} includes some data from the new/current task. Sampling the current task's data from the buffer also mitigates inter-task CF. Our method updates the replay buffer as the data incrementally arrives.

The MI between $F(X)$ and label Y can also be written as

$$I(F(X); Y) = H(F(X)) - H(F(X)|Y) \tag{4}$$

From this view, the features extracted from F should spread uniformly in the feature space to maximize $H(F(X))$. And for minimizing $H(F(X)|Y)$, we should make samples sharing the same class to be close/near to each other. That is, the ideal optimization result is that samples from the same class should be close to each other and samples of different classes should be far away from each other. These will guide the loss function design in Section 5.

4.2. Maximization of MI between Current and Past

After learning the first $t-1$ tasks, the model has acquired the knowledge about classes $\bigcup_{i=1}^{t-1} Y_i$. However, in learning the new task t, some knowledge or information about the previous classes may be forgotten. To reduce forgetting, we want F in learning t and the previous $F^{1:t-1}$ to share the knowledge about the classes of the previous tasks because it is crucial (1) to establish good decision boundaries between the previous classes and the new classes and (2) to maintain the decision boundaries among previous classes themselves. We try to mitigate forgetting at the feature level. Specifically, for the input x from X^{buf}, we want to maximize $I(F^{1:t-1}(x); F(x))$ when training the new task t. Note that we do not maximize the MI at the logits-level (Buzzega et al., 2020), that is, $\max_{\theta,\sigma} I(\sigma^{1:t-1}(f_\theta^{1:t-1}(x)); \sigma(f_\theta(x)))$. Because the model should allow some concept drift of previous classes as it needs to establish new decision boundaries between the previous classes and the current classes. The classifier (the logits) plays a vital role in classification as it chooses important features for classification. So we allow it to adjust its "view" for the past classes rather than making it completely consistent with the past classifier.

[4] If the data streaming is imbalanced and temporally correlated, we follow (Chrysakis & Moens, 2020) and use class-balancing reservoir sampling (CBRS) to guarantee the class balance in X^{buf}.

4.3. OCM Model

Based on the analysis above, we are ready to present the proposed model OCM. Figure 1 shows its architecture.

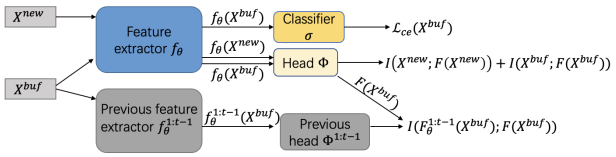

Figure 1. Architecture of OCM. $\mathcal{L}_{ce}$: cross-entropy loss.

In Figure 1, maximizing **(1)** $\mathcal{L}_{ce}(X^{buf})$ enables the model to learn a class-balanced classifier. Maximizing **(2)** $I(X^{new};F(X^{new}))+I(X^{buf};F(X^{buf}))$ helps the model learn holistic or comprehensive feature representations for inputs. After training $t-1$ $(t>1)$ tasks, we copy the trained model and freeze the copy as the previous model. In learning the new task, we maximize **(3)** $I(F^{1:t-1}(X^{buf});F(X^{buf}))$ to reduce forgetting of the learned model for previous classes. Our final optimization goal is to optimize the sum of those three objectives, i.e., **(1)+(2)+(3)**.

In summary, the proposed method OCM aims to learn comprehensive or holistic feature representations from the input data when training a new task. It also protects the learned knowledge for the previous tasks in learning subsequent tasks. The classifier is trained with class-balance inputs to maximize $H(Y)$.

5. Proxy for Mutual Information (MI)

Calculating MI is notoriously difficult. To overcome this problem, inferring the lower bound and then maximizing the tractable objective becomes a popular choice. Oord et al. (2018) proposed the InfoNCE loss as a proxy objective to maximize the MI. We follow their framework. Let $X^{'}$ be a different view of input variable X created via *data augmentation*. Each input image x is first copied and then applied with *horizontal-flip*, *random-resized-crop* and *random-gray-scale* operations to create $x^{'}$. We have

$$\max_{\theta,\Phi} I(X;F(X)) \geq \max_{\theta,\Phi} I(F(X);F(X^{'})) \tag{5}$$

The proof is in Appendix 1. According to (Oord et al., 2018), the term on the right has a lower bound:

$$\max_{\theta,\Phi} I(F(X);F(X^{'})) \geq \log B + \text{InfoNCE}(\{x_i\}_{i=1}^{B};g) \tag{6}$$

$$\text{InfoNCE}(\{x_i\}_{i=1}^{B};g) = \frac{1}{B}\sum_{i=1}^{B}\log\frac{g(x_i,x_i^{'})}{\sum_{j=1}^{B} g(x_i,x_j^{'})} \tag{7}$$

where $g(x_i,x_j^{'}) = e^{\frac{F(x_i)^T F(x_j^{'})}{r}}$ can be regarded as calculating the similarity of x_i and $x_j^{'}$ and r is the temperature. $\{x_i\}_{i=1}^{B}$ are samples from variable X and B is the batch size. When $B\to\infty$, we have $\log B + \text{InfoNCE}(\{x_i\}_{i=1}^{B};g) \to I(F(X);F(X^{'}))$ (Sordoni et al., 2021). So the inequality is tighter when B is larger. Now we also introduce the supervised label into InfoNCE and rewrite it as:

$$\text{InfoNCE}(\{x_i,y_{x_i}\}_{i=1}^{B};g) = \alpha\text{InfoNCE}(\{x_i\}_{i=1}^{B};g) + \beta\sum_{i=1}^{B}\frac{\sum_{k:y_{x_k}=y_{x_i}}\log\frac{g(x_i,x_k)g(x_i,x_k^{'})g(x_i^{'},x_k)}{(\sum_{j=1}^{B} g(x_i,x_j)+g(x_i,x_j^{'})+g(x_i^{'},x_j))^3}}{3B\sum_{s=1}^{B}1(y_{x_s}=y_{x_i})} \tag{8}$$

where 1 is the indicator function, α and β are hyperparameters.

Eq. 8 is the lower bound of $I(F(X);Y)$**+**$I(X;F(X))$: The first term on the right of Eq. 8 is the same as Eq. 7. It is the lower bound of $I(X;F(X))$. The second term on the right of Eq. 8 considers the supervised label Y. It's the lower bound of $I(F(X);Y)$. The proof is as follows: For a batch of augmented data $\{x_i,y_{x_i}\}_{i=1}^{B}$, we use the mean of the hidden representation $(F(x))$ of the samples whose labels are y to represent the class label y. Following the same inequality in Eq. 6 or in (Oord et al., 2018), we get the second term on the right of Eq. 8 as the lower bound of $I(F(X);Y)$.

Rotation Augmentation for a Tighter Bound. In the above analysis, we know that MI maximization would benefit from an increased batch size (Eq. 7). However, in online CL, the batch size is usually small. For example, in ASER (Shim et al., 2021), the new data batch size for X^{new} is 10, and in some recent online CL methods, the size of X^{buf} sampled from the replay buffer is also 10. We can increase the buffer batch size for X^{buf} to a larger number, but increasing the size of X^{new} can be problematic because that means we need to wait for a longer time to start a new training iteration. A naive solution is to duplicate the original data to increase the batch size. But the numerator will appear in the denominator multiple times, which can lead to model collapse. We provide an experiment of this in Appendix 8.

In this work, we propose to increase the batch size by rotating each image sample *globally* and *locally* to create pseudo samples. Like in (Tack et al., 2020), we also give pseudo classes to pseudo samples created by different rotations, which produce better features. For each image x_i with class label y_{x_i} (i.e., (x_i,y_{x_i})) from a set of images X (e.g., X may be X^{new} or X^{buf}), the data and class transformation process for (x_i,y_{x_i}) goes through the following three steps:

(1). Local Rotation: This method is new. For each image x_i, we divide it vertically into two halves and then rotate each half by 0 or 180 degrees to create four locally rotated versions of x_i. We denote these by $\{x_{i,(left,right)}\}_{left,right\in\{0,1\}}$, where *left* $=1/0$ means the left half of x_i is or is not rotated by 180 degrees, and *right* $=1/0$ means similarly.

(2). Global Rotation: We further rotate each image in $\{x_{i,(\textit{left,right})}\}_{\textit{left,right}\in\{0,1\}}$ *globally* by 0°, 90°, 180° and 270° to create four globally rotated versions (0 degree rotation is the original x_i). We denote these images by $\{x_{i,(\textit{left,right},r)}\}_{\textit{left,right}\in\{0,1\},r\in\{0°,90°,180°,270°\}}$, where the value of r is the rotation degree.

The combination of the global and local rotations give us 16 rotated images for each x_i. In Experiments, we describe 3 other transformations to each image to improve the results.

(3). Assigning 16 Classes: Each combination of global and local rotations is assigned a separate new class c. Altogether the 16 rotation combinations create 16 classes. Now each image has a pair of classes: (y_{x_i}, c), where y_{x_i} is the original class label of the input image x_i and c is one of the 16 augmented class labels. Different values of c represent different combinations of global and local rotations $(\textit{left}, \textit{right}, r)$.

In this way, the batch sizes for $X^{\textit{new}}$ and $X^{\textit{buf}}$ are increased when calculating the proxy of MI, which enables the system to achieve the benefit of the tighter lower bound. We provide the visualization of feature embeddings (representations) of training samples in Figure 3 in Appendix 2. In the figure, we will see that the embeddings of the original data are surrounded by the pseudo data embeddings created from them. We also investigate the influence of the number of pseudo classes in Section 6.3 and compare with creating more samples by random augmentations in Appendix 8.

5.1. Connecting $\Phi(f_\theta(x))$ and $f_\theta(x)$

We can directly optimize $I(X; \Phi(f_\theta(x)))$ via the MI proxy of InfoNCE loss for feature learning. However, it is possible to do better. Since Φ is the deepest layer for learning feature representations, it learns more global features than the earlier layers. Those global features may be important for distinguishing different images. However, those features in Φ are not used in classification. We propose a method to mitigate this problem by drawing $f_\theta(X)$ into the calculation of the similarity function g.

Let the dimension of $F(x)$ (or Φ) be d_1 ($F(x) \in R^{1\times d_1}$) and of $f_\theta(x)$ be d_2 ($f_\theta(x) \in R^{1\times d_2}$). Since $d_2 \geq d_1$, we cannot multiply $F(x)$ and $f_\theta(x)$ directly. We choose a random contiguous fragment of length d_1 from $f_\theta(x)$ to represent $f_\theta(x)$ in each training iteration. First, for a batch, we randomly sample an integer z from the interval $[0, d_2 - d_1]$. Then we extract the local features $f_\theta(x)[z, z+d_1]$ from the original $f_\theta(x)$, where $[z, z+d_1]$ means a fragment from z to $z + d_1$. Second, we change $g(x_i, x_j)$ for data in this batch to

$$g^*(x_i, x_j) = e^{\frac{\lambda F(x_i)^T F(x_j) + (1-\lambda) f_\theta(x_i)[z,z+d_1]^T F(x_j)}{r}} \quad (9)$$

where λ is a hyperparameter. Note that we want to reduce the semantic gap between $F(X)$ and $f_\theta(X)$. As d_1 is usually smaller than d_2, we cannot multiply $F(x)$ and $f_\theta(x)$ to calculate the *cosine similarity/distance* ($d_{cos} = 1 - cosine$). We instead calculate the average d_{cos} between $F(x)$ and continuous fragments $f_\theta(x)_{[a,b]} \in R^{1\times d_1}$ to achieve direction consistency between $F(X)$ and $f_\theta(X)$ by minimizing (1) $\mathbb{E}_{x\sim X} \frac{\sum_{i=1}^{d_2-d_1} d_{cos}(F(x), f_\theta(x)_{[i,i+d_1]})}{d_2-d_1}$. But as $d_2 - d_1$ can be huge, it causes high compute. So we randomly sample a contiguous fragment of length d_1 from $f_\theta(x)$ to compute (2) $\mathbb{E}_{(x,u)\sim(X,U)} \sum_{i=1}^{d_2-d_1} d_{cos}(F(x), f_\theta(x)_{[i,i+d_1]}) * u_i$, where U is the discrete uniform distribution over the integer set $\{z|0 \leq z \leq d_2 - d_1\}$ and its sample, an integer, is presented as one-hot vector $u \in R^{d_2-d_1\times 1}$. As U and X are independent, minimizing (2) is equivalent to minimizing (1) in expectation. Results in ablation study show that this operation improves the performance of OCM.

5.2. Putting Everything Together

When training a task t, for a new batch of training data consisting of $X^{\textit{new}} = \{x_i, y_{x_i}\}_{i=1}^N$ and $X^{\textit{buf}} = \{x_i^b, y_{x_i^b}\}_{i=1}^{N_b}$, we use the InfoNCE loss as a proxy for MI and the new data augmentation to enlarge the batch size. After completing the training of the task, we copy the trained model and freeze the copy as the previous model. The optimization objective for our method is calculated as (also see Section 4.3),

$$\begin{aligned}
&\max_{\theta,\sigma,\Phi} -\mathcal{L}_{ce}(X^{\textit{buf}}) + \{I(X^{\textit{new}}; F(X^{\textit{new}})) + I(X^{\textit{buf}}; F(X^{\textit{buf}})) + \\
&I(F^{1:t-1}(X^{\textit{buf}}); F(X^{\textit{buf}}))\} \\
&\approx \max_{\theta,\sigma}\{\mathcal{L}_{ce}(\{x_i^b, y_{x_i^b}\}_{i=1}^{N_b})\} + \max_{\theta,\Phi}\{3\log(16) + 2\log(N_b) + \\
&\log(N) + \text{InfoNCE}(\{\{x_{i,c}, y_{x_{i,c}}\}_{c=1}^{16}\}_{i=1}^{N}; g^*) + \text{InfoNCE}(\{\{x_{i,c}^b \\
&, y_{x_{i,c}^b}\}_{c=1}^{16}\}_{i=1}^{N^b}; g^*) + \text{InfoNCE}(\{\{x_{i,c}^b, y_{x_{i,c}^b}\}_{c=1}^{16}\}_{i=1}^{N^b}; g^{'})
\end{aligned} \quad (10)$$

where $x_{i,c}^b, x_{s,r}^b \in \{\{x_{i,c}^b, y_{x_{i,c}^b}\}_{c=1}^{16}\}_{i=1}^{N^b}$ and

$$g^{'}(x_{i,c}^b, x_{s,r}^b) = e^{\frac{F(x_{i,c}^b)^T F^{1:t-1}(x_{s,r}^b)}{r}} \quad (11)$$

When a new batch of data incrementally arrives, to protect the learned knowledge, the last term of the objective maximizes the MI between the current model and the frozen previous model using $X^{\textit{buf}}$. It brings the representations obtained from the current model F of the input samples in $X^{\textit{buf}}$ close to the representations obtained from the previous model $F^{1:t-1}$ of the input samples from the same class, and pushes away the representations obtained from the previous model $F^{1:t-1}$ of different classes.

The pseudo-code of our algorithm is given in **Algorithms 1 and 2** in Appendix 3.

6. Experiments

Evaluation data. We use 4 image classification datasets. **MNIST** (LeCun et al., 1998) has 10 classes with 60,000

examples for training and 10,000 examples for testing. It is split into 5 disjoint tasks with 2 classes per task. **CIFAR10** (Krizhevsky & Hinton, 2009) has 10 classes with 50,000 for training and 10,000 for testing. It is split into 5 disjoint tasks with 2 classes per task. **CIFAR100** (Krizhevsky & Hinton, 2009), which has 100 classes with 50,000 for training and 10,000 for testing. It is split into 10 disjoint tasks with 10 classes per task. **TinyImageNet** (Le & Yang, 2015) has 200 classes. It is split into 100 disjoint tasks with 2 classes per task. Each class has 500 training examples and 50 test examples. We train all methods with full-size datasets. We train all methods with two classes per task to investigate OCM's performance with a large number of tasks. In Appendix 4, we also show the results of 50 tasks for CIFAR100 (two classes per task) and OCM also outperforms the baselines.

Compared Baselines. We use 10 baselines. 7 are recent online CL baselines: **ER** (Chaudhry et al., 2020), **AGEM** (Chaudhry et al., 2018), **MIR** (Aljundi et al., 2019a), **ASER** (Shim et al., 2021), **GSS** (Aljundi et al., 2019b), **GDumb** (Prabhu et al., 2020), and **SCR** (Mai et al., 2021). 3 are recent replay-based batch CL baselines that use the knowledge distillation loss by running them in one epoch: **DER++** (Buzzega et al., 2020), **IL2A** (Zhu et al., 2021), **Co^{2}L** (Cha et al., 2021). Our **OCM**follows the memory retrieval/update strategy of **ER** as **ER** is a basic method.

6.1. Architectures, Data Augmentations, Training Details and Evaluation Protocol

Architecture. For MNIST, we employ a fully-connected network with two hidden layers as the feature extractor f_θ, each one comprising of 400 ReLU units. We use a linear layer of size [400, 10] as the classifier σ and a linear layer of size [400, 128] as the projection head Φ. For CIFAR10, CIFAR100, and TinyImageNet, we follow (Buzzega et al., 2020) and use the full ResNet18 (not pre-trained) as the feature extractor f_θ with model size 23MB. Denoting C_{num} as the number of all classes, we employ a linear layer (size $[dim_h, C_{num}]$) as the classifier σ and a linear layer of size $[dim_h, 128]$ as the feature projection head Φ. For baselines, we use the same full ResNet18 for fair comparisons.

Data augmentation. As we will see, data augmentation via *horizontal-flip*, *random-resized-crop* and *random-gray-scale* can improve accuracy. We also apply three transformations for all baselines except SCR, Co^{2}L and DER++ in calculating $\mathcal{L}_{ce}$. SCR does not need to calculate $\mathcal{L}_{ce}$ and SCR, Co^{2}L have their augmentations for calculating supervised contrastive loss. DER++ has its own data augmentations for calculating $\mathcal{L}_{ce}$ as well. Those augmentations improve ER, MIR, GDumb, AGEM, GSS, IL2A, and ASER by 2 to 4%.

Training and hyperparameter settings. For all datasets, OCM is trained with the Adam optimizer. We set the learning rate as 0.001 and fix the weight decay as 0.0001. Following (Shim et al., 2021), we set each data increment size N to 10 (the size of X^{new}) for all systems. For the memory buffer batch (X^{buf}) size N_b, in OCM, we initialize N_b as zero and increase it by seven slots when the system meets a new class. We set the max N_b allowed as 64. Again for fair comparisons, for baselines, we also set their memory buffer batch (X^{buf}) size as 64 and do not change it with tasks. Setting the maximal value of N_b as 64 improves the performance but doesn't increase the training time much. We provide an analysis of memory buffer batch size in Appendix 5. We use the official code and default hyper-parameters of the baselines. All experiments follow their settings. The hyper-parameters and official codes are listed in Appendix 5. *We run all methods with one epoch for each task*.

Evaluation protocol. Accuracy is used as the evaluation metric. We first learn from the data stream of all tasks for each dataset, and then test the final model using the test data of all tasks. We report the average accuracy of all tasks from 15 random runs for each dataset. We report the training time in Appendix 6, which shows OCM is an efficient method.

6.2. Results and Analysis

Accuracy performance. Table 1 shows the accuracy results of our method OCM and all baselines with various memory sizes on the four datasets. For all datasets and buffer sizes, OCM outperforms all baselines by obvious margins. For example, with the memory size 1000 (M=1k), OCM outperforms the best baselines by 13.1% on CIFAR10, 2.1% on MNIST. And OCM outperforms the best baseline by 5.9% on CIFAR100 when the memory size is 5k. The improvements over baselines on TinyImageNet are equally substantial. The results also show that increased memory size results in increased accuracy. We also provide the performance of OCM without maximizing the MI between current and past (i.e., (3) in Section 4.3) in the method "OCM (no past)". The performance is poorer than that of OCM but it still outperforms baselines. "OCM (no local rotation)" is also better than baselines. The results of OCM without maximizing the MI between X and $f_\theta(X)$ are not listed as they are poor, e.g., 58.9% for CIFAR10, which indicates that the mechanism (i.e., (2) in Section 4.3) for learning holistic features is very important.

Forgetting rate. Table 2 reports the average forgetting rate for all methods. The detailed forgetting rate computation (Chaudhry et al., 2020) is given in Appendix 7. OCM has substantially lower forgetting rates than baselines except GDumb and SCR for TinyImageNet, but both GDumb and SCR's accuracy values are substantially lower than OCM (see Table 1). This is because GDumb uses only the replay data to train the model, but not the new batch and SCR uses a nearest class mean classifier without optimizing the cross-entropy loss. They do not produce very accurate models.

Table 1. Accuracy on MNIST (5 tasks), CIFAR10 (5 tasks), CIFAR100 (10 tasks) and TinyImageNet (100 tasks) datasets with different memory buffer sizes M. All values are the averages of 15 runs

Method	MNIST			CIFAR10			CIFAR100			TinyImageNet		
M	M=0.1k	M=0.5k	M=1k	M=0.2k	M=0.5k	M=1k	M=1k	M=2k	M=5k	M=2k	M=4k	M=10k
AGEM (Chaudhry et al., 2018)	56.9±5.2	57.7±8.8	61.6±3.2	22.7±1.8	22.7±1.9	22.6±0.7	5.8±0.2	5.8±0.3	6.5±0.2	0.9±0.1	2.1±0.1	3.9±0.2
GSS (Aljundi et al., 2019b)	70.4±1.5	80.7±5.8	87.5±5.9	26.9±1.2	30.7±1.3	40.1±1.4	11.1±0.2	13.3±0.5	17.4±0.1	3.3±0.5	10.0±0.2	10.5±0.2
ER (Chaudhry et al., 2020)	78.7±0.4	88.0±0.2	90.3±0.1	29.7±1.0	35.2±0.3	44.3±0.4	11.7±0.3	15.0±0.9	14.4±0.9	5.6±0.5	10.1±0.7	11.7±0.2
MIR (Aljundi et al., 2019a)	79.0±0.5	88.3±0.1	91.3±1.9	37.3±0.3	40.0±0.6	41.0±0.6	15.7±0.2	19.1±0.1	24.1±0.2	6.1±0.5	11.7±0.2	13.5±0.2
ASER (Shim et al., 2021)	61.6±2.1	71.0±0.6	82.1±5.9	27.8±1.0	36.2±1.2	44.7±1.2	16.4±0.3	12.2±1.9	27.1±0.3	5.3±0.3	8.2±0.2	10.3±0.4
GDumb (Prabhu et al., 2020)	81.2±0.5	91.0±0.2	94.5±0.1	35.9±1.1	50.7±0.7	63.5±0.5	14.1±0.3	20.1±0.2	36.0±0.5	12.6±0.1	12.7±0.3	15.7±0.2
SCR (Mai et al., 2021)	86.2±0.5	92.8±0.3	94.6±0.1	47.2±1.7	58.2±0.5	64.1±1.2	26.5±0.2	31.6±0.5	36.5±0.2	10.6±1.1	17.2±0.1	20.4±1.1
DER++ (Buzzega et al., 2020)	74.4±1.1	91.5±0.2	92.1±0.2	44.2±1.1	47.9±1.5	54.7±2.2	15.3±0.2	19.7±1.5	27.0±0.7	4.5±0.3	10.1±0.3	17.6±0.5
IL2A (Zhu et al., 2021)	90.2±0.1	92.7±0.1	93.9±0.1	54.7±0.5	56.0±0.4	58.2±1.2	18.2±1.2	19.7±0.5	22.4±0.2	5.5±0.7	8.1±1.2	11.6±0.4
Co^{2}L (Cha et al., 2021)	83.1±0.1	91.5±0.1	94.7±0.1	42.1±1.2	51.0±0.7	58.8±0.4	17.1±0.4	24.2±0.2	32.2±0.5	10.1±0.2	15.8±0.4	22.5±1.2
OCM (no local rotation)	88.3±0.2	95.3±0.1	97.1±0.1	55.3±0.5	63.1±0.4	70.7±0.3	26.7±0.1	33.5±0.2	39.6±0.1	13.5±0.2	20.5±0.2	26.4±0.3
OCM (no past)	89.5±0.1	95.0±0.1	96.0±0.1	56.2±0.4	63.2±0.2	73.1±0.2	27.0±0.4	34.0±0.1	41.0±0.3	15.0±0.4	21.0±0.3	26.0±0.2
OCM	**90.7**±0.1	**95.7**±0.3	**96.7**±0.1	**59.4**±0.2	**70.0**±1.3	**77.2**±0.5	**28.1**±0.3	**35.0**±0.4	**42.4**±0.5	**15.7**±0.2	**21.2**±0.4	**27.0**±0.3

Table 2. Average forgetting rate. All numbers are the averages of 15 runs.

Method	MNIST			CIFAR10			CIFAR100			TinyImageNet		
M	M=0.1k	M=0.5k	M=1k	M=0.2k	M=0.5k	M=1k	M=1k	M=2k	M=5k	M=2k	M=4k	M=10k
AGEM (Chaudhry et al., 2018)	32.5±5.9	30.1±4.2	32.0±2.9	36.1±3.8	43.2±4.3	48.1±3.4	43.3±0.2	45.7±0.3	43.9±0.2	73.9±0.2	78.9±0.2	74.1±0.3
GSS (Aljundi et al., 2019b)	26.1±2.2	17.8±5.22	10.5±6.7	75.5±1.5	65.9±1.6	54.9±2.0	30.8±0.2	30.7±0.5	26.4±0.3	72.8±1.2	72.6±0.4	71.5±0.2
ER (Chaudhry et al., 2020)	22.7±0.5	9.7±0.4	6.7±0.5	42.0±0.3	26.7±0.7	20.7±0.7	34.2±0.2	31.7±0.9	35.3±0.9	68.2±2.8	66.2±0.8	67.2±0.2
MIR (Aljundi et al., 2019a)	22.3±0.5	9.0±0.5	5.7±0.9	40.0±1.6	25.9±0.7	24.5±0.5	24.5±0.3	21.4±0.3	21.0±0.1	61.1±3.2	60.9±0.3	59.5±0.3
ASER (Shim et al., 2021)	33.8±1.1	24.8±0.5	13.8±0.4	71.1±1.8	59.1±1.5	50.4±1.5	25.0±0.2	12.2±1.9	13.2±0.1	65.7±0.7	64.2±0.2	62.2±0.1
GDumb (Prabhu et al., 2020)	10.3±0.1	6.2±0.1	4.8±0.2	26.5±0.5	24.5±0.2	18.9±0.4	16.7±0.5	17.6±0.2	16.8±0.4	15.9±0.5	14.6±0.3	11.7±0.2
SCR (Mai et al., 2021)	10.7±0.1	4.7±0.1	4.0±0.2	41.3±0.1	31.5±0.2	24.7±0.4	17.5±0.2	11.6±0.5	5.6±0.4	19.4±0.3	15.4±0.3	14.9±0.7
DER++ (Buzzega et al., 2020)	25.0±0.3	7.3±0.3	6.6±1.2	30.1±0.8	31.8±2.5	18.7±3.4	43.4±0.2	44.0±1.9	25.8±3.5	67.2±1.7	63.6±0.3	55.2±0.7
IL2A (Zhu et al., 2021)	8.7±0.1	7.2±0.1	4.1±0.1	36.0±0.2	32.1±0.4	29.1±0.4	24.6±0.6	12.5±0.7	20.0±0.5	65.5±0.7	60.1±0.5	57.6±1.1
Co^{2}L (Cha et al., 2021)	14.7±0.2	7.1±0.1	3.1±0.1	32.0±0.1	21.0±0.3	16.9±0.2	16.9±0.4	16.6±0.6	9.9±0.7	60.5±0.5	52.5±0.9	42.5±0.8
OCM (no local rotation)	7.2±0.1	2.6±0.1	1.4±0.1	26.7±0.3	15.7±0.1	11.8±0.2	22.7±0.2	11.7±0.1	6.7±0.1	27.8±0.2	21.8±0.2	25.4±0.3
OCM (no past)	5.0±0.1	2.2±0.1	2.0±0.1	23.2±0.5	15.0±0.1	12.5±0.2	26.5±0.8	13.5±0.1	9.7±0.2	24.8±0.2	22.5±0.2	19.4±0.3
OCM	4.7±0.1	1.8±0.1	1.3±0.1	23.0±0.2	14.0±0.7	12.0±1.1	12.2±0.3	8.5±0.3	4.5±0.3	23.5±1.9	21.0±0.3	18.6±0.5

Table 3. Ablation accuracy - average of 5 runs. M is the memory buffer size and (no t) means that X^{buf} has no samples from the current task t in the memory buffer.

Dataset (M=1k)	$\mathcal{L}_{ce}$ in all	$\mathcal{L}_{ce}$ in all (no t)	MI union	unsupervised InfoNCE	gray scale	horizontal flip	resized crop	$\lambda = 1$ for $g^*(x_i, x_j)$	$\lambda = 0$ for $g^*(x_i, x_j)$
MNIST	94.8±0.1	96.1±0.1	95.8±0.1	93.6±0.1	-	-	93.9±0.1	96.2±0.1	95.3±0.1
CIFAR10	68.1±1.2	69.5±0.5	70.92±0.4	60.8±1.2	70.8±0.5	70.4±0.4	58.4±0.9	71.2±0.5	65.0±0.7

Table 4. Average accuracy (average forgetting rate) of MIR, ASER, DER++, and SCR without and with adding MI (+MI).

MIR	MIR+MI	ASER	ASER+MI	DER++	(DER++)+MI	SCR	SCR +MI	OCM
37.5(36.3)	46.3(18.7)	32.1(51.4)	38.7(44.5)	39.01(41.1)	44.0(37.7)	48.6(18.6)	50.9(17.5)	54.9(12.9)

6.3. Ablation Experiments

We conduct ablation experiments to analyze the contribution of various components and choices made in OCM with 1k (M=1k) memory. The results are given in Table 3.

(1). *Ablation study for MI between label and feature representation.* In experiment "$\mathcal{L}_{ce}$ to all," we apply cross-entropy to $X^{new} \cup X^{buf}$ with the task-free buffer rather than only to X^{buf} in OCM. In experiment "$\mathcal{L}_{ce}$ to all (no t)", we apply cross-entropy loss to $X^{new} \cup X^{buf}$ without sampling data from the current task t in the memory buffer. In experiment "MI union", we maximize the MI "$I(X^{buf} \cup X^{new}; F(X^{buf} \cup X^{new}))$" directly instead of separately as in OCM. In experiment "unsupervised InfoNCE," we set α as 1 and β as 0 for (2) in the OCM model. From Table 3, we see that their performances are all poorer than OCM (Table 1). Another interesting observation is that experiment "$\mathcal{L}_{ce}$ to all (no t)" can be viewed as training the ER baseline by maximizing our MI objective. It improves ER's performance drastically. For example, the performance improves by 27.0% on CIFAR10.

(2). *Ablation study for data augmentation and connecting* Φ *with* $f_\theta(x)$. Results in column 6-8 in Table 3 show that without using random-gray-scale or horizontal-flip does not affect the results much (random-gray-scale and horizontal-flip are not applied to MNIST), but random-resized-crop is extremely important. Without it, the accuracy drops drastically, e.g., from 77.2% (Table 1) to 58.4% (Table 3) for CIFAR10. This is because random-resized-crop randomly crops some pixels and resizes the image, so it has a higher

influence on the representation learning. Rotation ablations are given in Appendix 8.

About connecting $F(X)$ and $f_\theta(X)$ (Eq. 9), $\lambda = 1$ means without connection (column 9). Its results are poorer than OCM with connection (Table 1). $\lambda = 0$ means that we only consider $f_\theta(x_i)[z, z + d_1]^T F(x_j)$ in Eq. 9, which is even poorer (column 10).

(3). *Varying the number of pseudo classes*. Results in Figure 2(a) show positive correlation between the classification performance and the number of pseudo classes. The increased number of pseudo classes means tighter bound. It empirically verifies that CL benefits from MI maximization.

6.4. Improving Baselines by Adding MI Maximization

We add (1)+(2) of OCM (see Section 4.3) to the loss of 4 baselines MIR, ASER, DER++, and SCR. Table 4 shows the average accuracy and forgetting rate (in brackets) over all four datasets and memory sizes in Table 1. The detailed results are given in Appendix 9. We see MI maximization improves these baselines markedly.

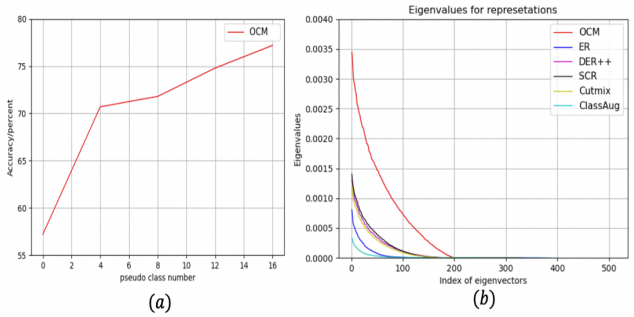

Figure 2. (a) Accuracy performance of OCM with different number of pseudo classes, tested on CIFAR10. Let the number of pseudo classes be $n \leq 16$ (16 rotation classes). For each experiment of the plot, we randomly choose n different pseudo classes for model building. After the training of five tasks, we test the trained model using the test data of the five tasks. We report the average accuracy. All the numbers are the averages of 5 random runs. (b) Eigenvalue distribution of ER, SCR, DER++, Cutmix (Yun et al., 2019), ClassAug (Zhu et al., 2021) and our OCM on CIFAR100. Note that Cutmix and ClassAug are not continual learning systems but are only data augmentation systems.

6.5. Holistic Degree of Representation

Zhu et al. (2021) observed that representation $f_\theta(x)$ with larger eigenvalues transfer better and suffer less forgetting. To improve the transferrability and diversity of the learned representation, it is important to enlarge the eigenvalues and increase the number of eigenvectors with significant variance. This is essentially an indicator of holistic degree of the learned representations or how holistic/comprehensive the learned representations are in capturing the features of the input images. We follow the experiment setting of (Zhu et al., 2021) and display (in Figure 2(b)) the eigenvalues of eigenvectors of the representations learned with different methods from the first 50 classes of CIFAR-100 as a single supervised learning task. Figure 2(b) shows that our OCM clearly enhances eigenvalues and has more significant directions (eigenvectors) than other methods, which shows that the representation learned by OCM is more holistic, indicating OCM learns more comprehensive features.

7. Conclusions

This paper proposed to use mutual information (MI) as the theoretical framework to deal with CF in online CL. To our knowledge, this has not been done before. The proposed MI formulation not only learns more holistic/comprehensive features but also preserves the past model learned from previous tasks. Both help deal with CF. Experimental results showed that our OCM markedly outperforms the latest online CL baselines. This paper focused on online CL, we believe the MI maximization should also be applicable to batch CL. We will study it in our future work.

Acknowledgement

The work of Yidou Guo and Dongyan Zhao was supported in part by the National Key Research and Development Program of China (No. 2020AAA0106600). We thank Changnan Xiao for some useful discussions.

References

Abati, D., Tomczak, J., Blankevoort, T., Calderara, S., Cucchiara, R., and Bejnordi, B. E. Conditional channel gated networks for task-aware continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3931–3940, 2020.

Ahn, H., Cha, S., Lee, D., and Moon, T. Uncertainty-based continual learning with adaptive regularization. In *NeurIPS*, 2019.

Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., and Tuytelaars, T. Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*, 2019a.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019b.

Beaudry, N. J. and Renner, R. An intuitive proof of the data processing inequality. *arXiv preprint arXiv:1107.0740*, 2011.

Boudiaf, M., Rony, J., Ziko, I. M., Granger, E., Pedersoli,

M., Piantanida, P., and Ayed, I. B. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. In *European Conference on Computer Vision*, pp. 548–564. Springer, 2020.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., and Alahari, K. End-to-end incremental learning. In *ECCV*, pp. 233–248, 2018.

Cha, H., Lee, J., and Shin, J. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9516–9525, 2021.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. Continual learning with tiny episodic memories. In *ICML-2019*.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Chaudhry, A., Khan, N., Dokania, P. K., and Torr, P. H. Continual learning in low-rank orthogonal subspaces. *arXiv preprint arXiv:2010.11635*, 2020.

Chrysakis, A. and Moens, M.-F. Online continual learning from imbalanced data. In *International Conference on Machine Learning*, pp. 1952–1961. PMLR, 2020.

de Masson d'Autume, C., Ruder, S., Kong, L., and Yogatama, D. Episodic memory in lifelong language learning. In *NeurIPS*, 2019.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773. PMLR, 2020.

Guo, Y., Hu, W., Zhao, D., and Liu, B. Adaptive orthogonal projection for batch and online continual learning. In *Proceedings of AAAI-2022*, 2022.

Hinton, G., Vinyals, O., and Dean, J. Dark knowledge. *Presented as the keynote in BayLearn*, 2, 2014.

Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 831–839, 2019.

Hu, W., Lin, Z., Liu, B., Tao, C., Tao, Z., Ma, J., Zhao, D., and Yan, R. Overcoming catastrophic forgetting for continual learning via model adaptation. In *ICLR*, 2019.

Hu, W., Qin, Q., Wang, M., Ma, J., and Liu, B. Continual learning by using information of each class holistically. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7797–7805, 2021.

Hung, S. C. Y., Tu, C.-H., Wu, C.-E., Chen, C.-H., Chan, Y.-M., and Chen, C.-S. Compacting, picking and growing for unforgetting continual learning. In *NeurIPS*, 2019.

Ke, Z., Liu, B., and Huang, X. Continual learning of a mixed sequence of similar and dissimilar tasks. In *NeurIPS*, 2020.

Ke, Z., Liu, B., Xu, H., and Shu, L. Classic: Continual and contrastive learning of aspect sentiment classification tasks. In *Proceedings of 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP-2021)*, 2021.

Kim, G., Esmaeilpour, S., Xiao, C., and Liu, B. Continual learning based on ood detection and task masking. *arXiv:2203.09450 [cs.CV]*, 2022.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., and Others. Overcoming catastrophic forgetting in neural networks. volume 114, pp. 3521–3526. National Acad Sciences, 2017.

Korycki, Ł. and Krawczyk, B. Class-incremental experience replay for continual learning under concept drift. *arXiv preprint arXiv:2104.11861*, 2021.

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

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015.

LeCun, Y., Cortes, C., and Burges, C. J. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

Li, X., Zhou, Y., Wu, T., Socher, R., and Xiong, C. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *ICML*, 2019.

Mai, Z., Li, R., Kim, H., and Sanner, S. Supervised contrastive replay: Revisiting the nearest class mean classifier in online class-incremental continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 3589–3599, 2021.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning & motiv.*, volume 24. 1989.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Ostapenko, O., Puscas, M., Klein, T., Jahnichen, P., and Nabi, M. Learning to remember: A synaptic plasticity driven framework for continual learning. In *CVPR*, pp. 11321–11329, 2019.

Prabhu, A., Torr, P. H., and Dokania, P. K. Gdumb: A simple approach that questions our progress in continual learning. In *EECV*, pp. 524–540, 2020.

Rajasegaran, J., Hayat, M., Khan, S., Shahbaz, F., and Shao, K. L. Random path selection for incremental learning. In *NeurIPS*, 2019.

Rajasegaran, J., Khan, S., Hayat, M., Khan, F. S., and Shah, M. itaml: An incremental task-agnostic meta-learning approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13588–13597, 2020.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *CVPR*, pp. 2001–2010, 2017.

Ritter, H., Botev, A., and Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. In *NIPS*, pp. 3738–3748, 2018.

Saha, G., Garg, I., and Roy, K. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*, 2021.

Serrà, J., Surís, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, 2018.

Shim, D., Mai, Z., Jeong, J., Sanner, S., Kim, H., and Jang, J. Online class-incremental continual learning with adversarial shapley value. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9630–9638, 2021.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *NIPS*, pp. 2994–3003, 2017.

Sokar, G., Mocanu, D. C., and Pechenizkiy, M. Learning invariant representation for continual learning. *arXiv preprint arXiv:2101.06162*, 2021.

Sordoni, A., Dziri, N., Schulz, H., Gordon, G., Bachman, P., and Des Combes, R. T. Decomposed mutual information estimation for contrastive representation learning. In *International Conference on Machine Learning*, pp. 9859–9869. PMLR, 2021.

Tack, J., Mo, S., Jeong, J., and Shin, J. Csi: Novelty detection via contrastive learning on distributionally shifted instances. In *Proceedings of 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

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

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. Continual learning with hypernetworks. *ICLR*, 2020.

Wang, J., Wang, X., Shang-Guan, Y., and Gupta, A. Wanderlust: Online continual object detection in the real world. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10829–10838, 2021.

Wang, L., Zhang, X., Yang, K., Yu, L., Li, C., Hong, L., Zhang, S., Li, Z., Zhong, Y., and Zhu, J. Memory replay with data compression for continual learning. In *ICLR-2022*, 2022.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. Supermasks in superposition. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *NeurIPS*, 2020.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., and Fu, Y. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 374–382, 2019.

Yan, S., Zhou, J., Xie, J., Zhang, S., and He, X. An em framework for online incremental learning of semantic segmentation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pp. 3052–3060, 2021.

Yin, H., Li, P., et al. Mitigating forgetting in online continual learning with neuron calibration. *Advances in Neural Information Processing Systems*, 34, 2021.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong Learning with Dynamically Expandable Networks. In *ICLR*, 2018.

Yu, L., Twardowski, B., Liu, X., Herranz, L., Wang, K., Cheng, Y., Jui, S., and Weijer, J. v. d. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6982–6991, 2020.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the*

IEEE/CVF International Conference on Computer Vision, pp. 6023–6032, 2019.

Zeng, G., Chen, Y., Cui, B., and Yu, S. Continuous learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 2019.

Zhang, J., Zhang, J., Ghosh, S., Li, D., Tasci, S., Heck, L., Zhang, H., and Kuo, C.-C. J. Class-incremental learning via deep model consolidation. In *CVPR*, 2020.

Zhao, H., Wang, H., Fu, Y., Wu, F., and Li, X. Memory efficient class-incremental learning for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Zhu, F., Cheng, Z., Zhang, X.-y., and Liu, C.-l. Class-incremental learning via dual augmentation. *Advances in Neural Information Processing Systems*, 34, 2021.

A. Appendix

A.1. Appendix 1: Proof of Eq. 5

$F(X) \leftarrow X \rightarrow F(X^{'})$ is *Markov equivalent* to $F(X) \rightarrow X \rightarrow F(X^{'})$ as both graphs meet the same conditional independence (*CI*) condition: $F(X^{'})$ and $F(X)$ are conditionally independent given X. Then we have $I(X;F(X)) = I(X, F(X^{'}); F(X))$ as $I(X, F(X^{'}); F(X)) = H(F(X)) - H(F(X)|X, F(X^{'})) = H(F(X)) - H(F(X)|X)$ ($\frac{P(F(X),F(X^{'}),X)}{P(F(X^{'}),X)} = \frac{P(F(X),F(X^{'})|X)}{P(F(X^{'})|X)}$ $\overset{(CI)}{=} P(F(X)|X)$). Also, we have $I(X, F(X^{'}); F(X)) \geq I(F(X^{'}); F(X))$ as $H(F(X)|X, F(X^{'})) \leq H(F(X)|F(X^{'}))$. So $I(X;F(X)) \geq I(F(X^{'}); F(X))$.

A.2. Appendix 2: Visualization of Rotation Augmentation

Figure 3 visualizes the data of each original class and the pseudo data created by rotations and their pseudo classes. We observe that each original class's data is surrounded by pseudo classes' data created from it (see the caption of Figure 3).

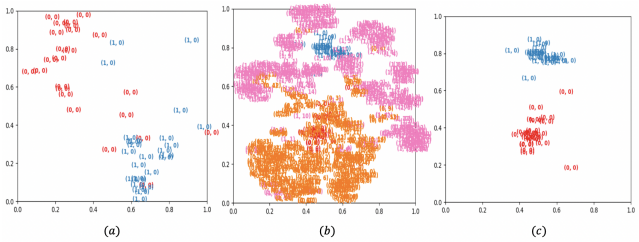

Figure 3. 2D t-SNE (Van der Maaten & Hinton, 2008) visualization of CIFAR-10 data feature embeddings (class 0 and class 1) and their class labels (different colors): samples from class 1 (blue), pseudo data created from samples belonging to class 1 (pink), samples from class 0 (red), and pseudo data created from samples belong to class 0 (orange). We use all the training data of class 0 and class 1 in the CIFAR10 dataset to create one task. After the training of this task, we randomly choose 50 samples from the task's training set and get their data and their pseudo data's feature embeddings from the trained feature extractor f_θ. Sub-figure (a) shows the visualization of samples from class 1 and class 0 with the trained f_θ without using our rotation augmentation. Sub-figure (c) shows the visualization with the trained f_θ using our rotation augmentation in the training process. On the basis of (c), sub-figure (b) adds the points of pseudo data created from the original samples.

A.3. Appendix 3: OCM Algorithm

The proposed OCM algorithm is given in Algorithm 1 and Algorithm 2.

A.4. Appendix 4: 50 Tasks of CIFAR100

In the original papers of SCR, MIR, and ASER, they train and test their methods on CIFAR100 using 10 classes per task. We also train OCM in this setting and test its performance. As Table 5 shows, the accuracy of our method OCM is the highest on the CIFAR100 dataset (10 classes per task). For example, when $M = 5k$, OCM outperforms the best baseline (SCR) by 5.4%.

Algorithm 1 - OCM Training Algorithm

Input: dataset $\{D_t\}_{t=1}^{T}$, feature extractor f_θ, linear classifier σ, feature head Φ, *augment* which is comprised of the Random-Resized-Crop, Horizontal-flip, and Random-Color-Gray transform.
Initialization: $\mathcal{M} \leftarrow \{\}$, // $\mathcal{M}$ is the replay/memory buffer
for $t = 1$ to T **do**
  for each new increment of data X^{new} **in** D_t's data stream **do**
    $X^{buffer} \leftarrow sample(\mathcal{M})$
    $\{\{x_{i,c}, y_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N} \leftarrow$ **Rotation**(X^{new}) // c replaces $(left, right, r)$ in **Algorithm** 2
    $\{\{x^{'}_{i,c}, y^{'}_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N} \leftarrow$ **Copy**$(\{\{x_{i,c}, y_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N})$
    $\{x^{'}_{i,c}, y^{'}_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N} \leftarrow$ **Augment**$(\{\{x^{'}_{i,c}, y^{'}_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N})$
    if $t = 1$ **then**
      $\theta \leftarrow \theta + \nabla_\theta[\frac{1}{N_b}\sum_{i=1}^{N_b} \mathcal{L}_{ce}(\sigma(f_\theta(x_i^b)), y_{x_i^b}) + \text{InfoNCE}(\{\{x_{i,c}, y_{x_{i,c}}\}_{c=1}^{16}\}_{i=1}^{N}; g^*)$
    end if
    if $t > 1$ **then**
      $\{\{x^b_{i,c}, y^b_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N_b} \leftarrow$ **Rotation**(X^{buffer})
      $\{\{x^{',b}_{i,c}, y^{',b}_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N_b} \leftarrow$ **Augment**(**Copy**$(\{\{x^b_{i,c}, y^b_{x_i,c}\}_{c=1}^{16}\}_{i=1}^{N_b})$)
      $\theta \leftarrow \theta + \nabla_\theta[\frac{1}{N_b}\sum_{i=1}^{N_b} \mathcal{L}_{ce}(\sigma(f_\theta(x_i^b)), y_{x_i^b}) + \text{InfoNCE}(\{\{x_{i,c}, y_{x_{i,c}}\}_{c=1}^{16}\}_{i=1}^{N}; g^*) + \text{InfoNCE}(\{\{x^b_{i,c}, y_{x^b_{i,c}}\}_{c=1}^{16}\}_{i=1}^{N^b}; g^*) + \text{InfoNCE}(\{\{x^b_{i,c}, y_{x^b_{i,c}}\}_{c=1}^{16}\}_{i=1}^{N^b}; g^{'})$
    end if
    $\mathcal{M} \leftarrow$ *reservoir* $(\mathcal{M}, \{x_i, y_{x_i}\}_{i=1}^{N})$
  end for
  $F^{1:t} \leftarrow F$
end for

Algorithm 2 - Rotation

Input: a set of data $\{x_i, y_{x_i}\}_{i=1}^{n}$
Output: $\mathcal{S}$
Initialization: $\mathcal{S} \leftarrow \emptyset$;
for $i = 1$ to n **do**
  $\{(x_{i,(left,right)}, y_{x_i,(left,right)})\}_{left,right\in\{0,1\}} \leftarrow$ **rotation_local**(x_i, y_{x_i})
  for r **in** $\{0°, 90°, 180°, 270°\}$ **do**
    $\{(x_{i,(left,right,r)}, y_{x_i,(left,right,r)})\}_{left,right\in\{0,1\}} \leftarrow$ **rotation_global**$(x_{i,r}, y_{x_i,r})$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(x_{i,(left,right,r)}, y_{x_i,(left,right,r)})\}_{left,right\in\{0,1\}}$
  end for
end for

Table 5. Accuracy and average forgetting rate (average of 15 runs) on the CIFAR100 (50 tasks) dataset with different memory buffer sizes M.

Method	CIFAR100			CIFAR100		
M	M=1k	M=2k	M=5k	M=1k	M=2k	M=5k
Metrics	Accuracy			Average forgetting rate		
AGEM (Chaudhry et al., 2018)	1.8±0.2	1.8±0.3	1.8±0.3	78.6±2.1	77.5±1.3	78.3±1.2
GSS (Aljundi et al., 2019b)	4.3±0.2	8.3±0.1	8.7±0.1	73.4±4.2	69.3±3.1	70.9±2.9
ER (Chaudhry et al.)	8.3±0.3	11.4±0.9	13.4±0.9	65.1±1.3	59.3±0.9	60.0±1.6
MIR (Aljundi et al., 2019a)	12.7±0.3	14.3±1.3	15.3±0.1	40.9±1.3	41.9±0.6	44.3±0.1
MIR (Aljundi et al., 2019a)+MI	20.1±0.2	25.1±0.4	33.1±0.2	22.5±0.2	20.0±0.5	18.0±0.6
ASER (Shim et al., 2021)	9.6±1.3	12.2±1.9	16.0±2.5	61.5±1.2	58.0±2.9	52.3±2.7
DER++ (Buzzega et al., 2020)	9.3±0.3	9.6±1.0	12.3±3.5	64.5±1.4	62.1±1.9	60.1±0.7
DER++ (Buzzega et al., 2020)+MI	20.0±0.3	27.4±0.3	31.6±0.2	41.4±0.2	40.0±1.9	22.8±3.5
GDumb (Prabhu et al., 2020)	18.1±0.3	30.1±0.2	36.0±0.5	18.9±0.4	19.2±0.4	20.9±0.7
SCR (Mai et al., 2021)	25.6±1.3	31.0±0.5	35.0±0.5	23.0±0.4	15.1±0.4	11.5±0.7
OCM	29.1±0.2	36.6±0.4	42.2±0.2	26.0±0.2	23.1±0.2	19.1±0.3

A.5. Appendix 5: Hyperparameters

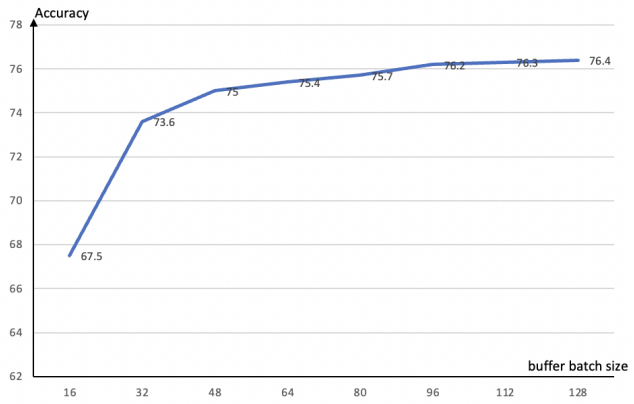

Figure 4. The performance of OCM with different buffer batch (X^{buffer}) sizes on CIFAR10 ($M = 1k$)

For OCM, the hyperparameters are given in the main paper. Here we add an experiment to show the effect of buffer batch size X^{buffer} (see Figure 4). From Figure 4, we observe that the performance of our method becomes better when the buffer batch (X^{buffer}) size increases. But the training time also grows with larger buffer batch sizes. To achieve a trade off, we set the buffer batch size to 64. Also, We set λ as 0.5 (Eq. 9 in main paper). Note that Eq. 8 in main paper is used as proxy for both (2) and (3) in the OCM Model section (the full objective is in Eq. 10 in main paper). We set α as 1 and β as 2 for (2) and α to 0 and β still as 2 for (3).

For the choice of random seed, we use the Numpy and set the seed as 0. For interested readers, we list the details of baselines' default hyperparameters given in their papers.

For AGEM, following the original paper, we use the SGD optimizer and set the learning rate as 0.1. We use the random method to update the buffer and to sample data.

For GSS, based on the original paper, we use the same optimizer and learning rate as above. The number of buffer batches randomly sampled from the memory to estimate the maximal gradients cosine similarity score is set to 10 and the random sampling buffer batch (X^{buffer}) size for calculating the score is 64.

For ASER, we again use the SGD optimizer and set the learning rate as 0.1 as in their paper. We use the mean value of Adversarial SV and Cooperative SV, and set the maximum number of samples per class for random sampling as 1.5. We

allow 3 nearest neighbors for KNN-SV computation. We use the same SV-based methods for both Memory-Update and Memory-Retrieval as given in the original paper.

For MIR/ER, we use the Adam optimizer and we set the learning rate as 0.001 and fix the weight decay as 0.0001. We set the sub-sample size as 128.

For DER++, we use the Adam optimizer, set the learning rate as 0.001, fix the weight decay as 0.0001 and the value of alpha (α) as 0.1, and fix the beta (β) as 0.5.

For GDumb, we use the Adam optimizer, set the learning rate as 0.001 and fix the weight decay as 0.0001. We use the CutMix as the regularization to overcome over-fitting. we follow the official code and set the number of epoch for training the whole buffer data as 256 for MINIST, CIFAR10, and CIFAR100 datasets, and 32 for the TinyImagenet dataset. We set the gradient clip as 10.

For SCR, we use the Adam optimizer and we set the learning rate as 0.001 and fix the weight decay as 0.0001. We set the temperature for contrastive loss as 0.07. We employ a linear layer with the size [dim_h,128] as the contrastive head. We follow the official code and use the horizontal-flip, random-resized crop, random-gray-scale,color-jitter as its data augmentations. The official codes are The code of ER and MIR: https://github.com/optimass/Maximally_Interfered_Retrieval.
The code of ASER and SCR: https://github.com/RaptorMai/online-continual-learning.
The code of GDumb: https://github.com/drimpossible/GDumb.
The code of DER++: https://github.com/aimagelab/mammoth.
The code for AGEM: https://github.com/facebookresearch/agem.
The code for GSS: https://github.com/rahafaljundi/Gradient-based-Sample-Selection.
The code for Co^{2}L: https://github.com/chaht01/Co2L.
The code for IL2A: https://github.com/Impression2805/IL2A.

A.6. Appendix 6: Execution Time

Figure 5 and Figure 6 chart the training times of all systems on MNIST and CIFAR10, respectively. We observe from Figure 5 that OCM training is more efficient than SCR, MIR, GDumb, GSS for the MNIST dataset. OCM (no local rotation) is only slightly less efficient than ER, DER++, and AGEM, but our method outperforms them in accuracy by large margins (see the main paper). For the CIFAR10 dataset in Figure 6, OCM is still faster than DGumb and GSS. OCM (no local rotation), which is more efficient and also performs better than all baselines (see the main paper).

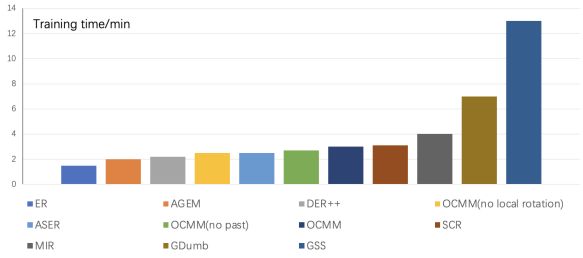

Figure 5. Training time of each method on the MNIST dataset.

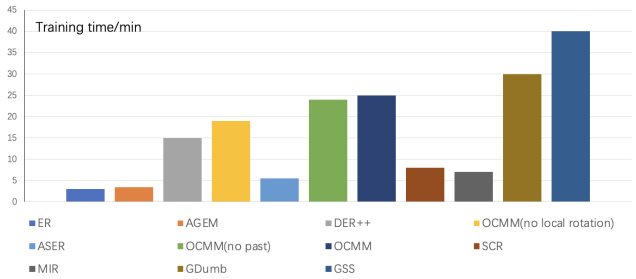

Figure 6. Training time of each method on the CIFAR10 dataset

A.7. Appendix 7: Computing the Average Forgetting Rate

After training the model from task 1 to task j, we denote $acc_{j,i}$ as the accuracy of trained model evaluated on the held-out test set of task $i \leq j$. The average forgetting rate FR_t at task t is computed (Mai et al., 2021) as:

$$FR_t = \frac{\sum_{i=1}^{t-1} f_i^t}{t-1}, \text{where} f_i^t = \max_{l \in \{1,2,\ldots,t-1\}} (acc_{l,i} - acc_{t,i}) \tag{12}$$

A.8. Appendix 8: Rotation Ablation and Duplicate Data

Rotation ablation. We use MNIST and CIFAR10 with the replay buffer size 1k to conduct these experiments. Table 6 shows the results of using "different view", "only local" rotation, "no rotation","rotation as one class". In experiment "different view", we first copy the data 16 times to enlarge the batch without rotations and then use random-resized crop and random-gray-scale randomly in the copy to

Table 6. Ablation accuracy (average of 5 runs) - different data augmentations. *M* is the memory buffer size. The last column also shows the results of duplicating data to increase batch size (see the Rotation Augmentation for a Tighter Bound section in the main paper).

Dataset (M=1k)	different view	only local	no rotation	rotations as one class	duplicate
MNIST	96.0±0.1	95.2±0.2	95.6±0.2	95.9±0.1	96.3±0.1
CIFAR10	67.4±0.2	67.7±0.2	67.1±0.1	60.1±0.1	67.1±1.1

change the duplicate versions. So each sample in the enlarged batch is different. No pseudo classes are created in this case. We use these data to calculate the loss. We can see all these results are markedly poorer than OCM (Table 1 in the main paper).

In experiment "only local," we apply only local rotation. We can see the results are also poorer than OCM which has global rotation (Table 1 in the main paper). Note that the results of using only global rotation (OCM (no local rotation)) are given in Table 1 of the main paper.

In experiment "rotation as one class," all pseudo data are assigned to one class. We named it "pseudo class" to differentiate it from the original classes. Table 6 shows that this operation hurts the performance. One reason is that this class creates imbalance between "pseudo class" and the original classes. Another reason is that pseudo data created by different rotations are semantically different. Treating them as one class ignores their differences, which hurts the classification ability of the method.

Varying the number of pseudo classes: We also conducted experiments by varying the number of pseudo classes. From Figure 2 in the main paper, we can see that the performance of our method benefits from the increased number of pseudo classes.

Duplicate data: In experiment "duplicate", we duplicate the new data (x and y_x) 16 times and use the final version to calculate the InfoNCE loss (see the Rotation Augmentation for a Tighter Bound section in the main paper). Although it has the same number of samples as OCM. its performance is poorer than that of OCM (see Table 1 in the main paper). The reason is the model collapse.

A.9. Appendix 9: Detailed Results of the Four Baselines with MI Maximization

Table 7. Accuracy (average of 15 runs) on MNIST (5 tasks), CIFAR10 (5 tasks), CIFAR100 (50 tasks) and TinyImageNet (100 tasks) datasets with different memory buffer sizes *M*. '+MI' means adding mutual information (MI) maximization.

Method	MNIST			CIFAR10			CIFAR100			TinyImageNet		
M	*M*=0.1*k*	*M*=0.5*k*	*M*=1*k*	*M*=0.2*k*	*M*=0.5*k*	*M*=1*k*	*M*=1*k*	*M*=2*k*	*M*=5*k*	*M*=2*k*	*M*=4*k*	*M*=10*k*
MIR (Aljundi et al., 2019a)	79.0±0.5	88.3±0.1	91.3±1.9	37.3±0.3	40.0±0.6	41.0±0.6	12.7±0.3	14.3±1.3	15.3±0.1	6.1±0.5	11.7±0.2	13.5±0.2
MIR (Aljundi et al., 2019a)+MI	87.0±0.1	92.0±0.1	94.0±0.1	51.0±0.4	57.0±0.4	62.1±0.3	20.7±0.2	24.3±0.3	30.1±0.2	8.0±0.3	14.0±0.3	15.5±0.4
ASER (Shim et al., 2021)	61.6±2.1	71.0±0.6	82.1±5.9	27.8±1.0	36.2±1.2	44.7±1.2	9.6±1.3	12.2±1.9	16.0±2.5	5.3±0.3	8.2±0.2	10.3±0.4
ASER (Shim et al., 2021)+MI	62.6±0.9	79.0±0.2	86.5±0.5	32.3±0.5	44.3±0.7	50.1±0.4	19.4±0.5	21.0±0.5	29.3±0.4	6.0±0.2	14.2±0.2	20.1±0.2
DER++ (Buzzega et al., 2020)	74.4±1.1	91.5±0.2	92.1±0.2	44.2±1.1	47.9±1.5	54.7±2.2	9.3±0.3	9.6±1.0	12.3±3.5	4.5±0.3	10.1±0.3	17.6±0.5
DER++ (Buzzega et al., 2020)+MI	81.3±1.1	93.3±0.3	94.3±0.1	59.6±0.5	60.3±1.4	67.5±0.9	10.4±0.5	11.5±0.6	14.3±2.4	5.6±0.3	12.0±0.3	18.0±0.5
SCR (Mai et al., 2021)	86.2±0.5	92.8±0.3	94.6±0.1	47.2±1.7	58.2±0.5	64.1±1.2	25.6±1.3	31.0±0.5	35.0±0.5	10.6±1.1	17.2±0.1	20.4±1.1
SCR (Mai et al., 2021)+MI	88.3±0.1	94.8±0.2	94.7±0.1	46.8±0.5	60.3±1.4	67.5±0.9	31.2±0.5	32.9±0.7	38.4±0.4	14.0±0.2	20.5±0.3	22.4±0.4

We report the performance of the four baselines (MIR, ASER, DER++, and SCR) with/without adding maximizing the mutual information (+MI) goal in Table 7 and Table 8. From the two tables, we can observe that maximizing the MI goal improves the average accuracy of all four baselines and reduces the average forgetting rate as well.

Table 8. Average forgetting rate with or without adding MI (+MI). All numbers are the average of 15 random runs.

Method	MNIST			CIFAR10			CIFAR100			TinyImageNet		
M	*M*=0.1*k*	*M*=0.5*k*	*M*=1*k*	*M*=0.2*k*	*M*=0.5*k*	*M*=1*k*	*M*=1*k*	*M*=2*k*	*M*=5*k*	*M*=2*k*	*M*=4*k*	*M*=10*k*
MIR (Aljundi et al., 2019a)	22.3±0.5	9.0±0.5	5.7±0.9	40.0±1.6	25.9±0.7	24.5±0.5	40.9±1.3	41.9±0.6	44.3±0.1	61.1±3.2	60.9±0.3	59.5±0.3
MIR (Aljundi et al., 2019a)+MI	16.0±0.2	8.0±0.2	5.0±0.2	34.0±0.3	19.2±0.3	18.0±0.3	24.9±0.2	24.0±0.4	14.0±0.2	22.1±0.6	20.1±0.1	19.5±0.2
ASER (Shim et al., 2021)	33.8±1.1	24.8±0.5	13.8±0.4	71.1±1.8	59.1±1.5	50.4±1.5	61.5±1.2	58.0±2.9	52.3±2.7	65.7±0.7	64.2±0.2	62.2±0.1
ASER (Shim et al., 2021)+MI	33.6±1.1	18.2±1.1	11.7±1.5	67.0±1.8	44.3±0.8	41.5±0.6	56.4±0.2	51.0±0.4	41.7±0.5	63.7±0.7	56.0±0.3	46.4±0.1
DER++ (Buzzega et al., 2020)	25.0±0.3	7.3±0.3	6.6±1.2	30.1±0.8	31.8±2.5	18.7±3.4	64.5±1.4	62.1±1.9	60.1±0.7	67.2±1.7	63.6±0.3	55.2±0.7
DER++ (Buzzega et al., 2020)+MI	17.7±0.1	5.2±0.2	4.2±1.2	27.8±0.3	25.1±0.3	17.7±0.5	61.0±0.3	60.1±0.3	57.1±0.2	65.2±1.7	61.6±0.3	50.2±0.7
SCR (Mai et al., 2021)	10.7±0.1	4.7±0.1	4.0±0.2	41.3±0.1	31.5±0.2	24.7±0.4	23.0±0.4	15.1±0.4	11.5±0.7	19.4±0.3	15.4±0.3	14.9±0.7
SCR (Mai et al., 2021)+MI	9.4±0.1	3.3±0.2	3.6±0.2	46.2±0.3	30.1±0.3	17.7±0.5	21.8±0.3	20.1±0.4	16.5±0.5	15.2±1.7	14.5±0.3	11.1±0.7