Knowledge Consolidation based Class Incremental
Online Learning with Limited Data

Mohammed Asad Karim\textsuperscript{1*}, Vinay Kumar Verma\textsuperscript{2}, Pravendra Singh\textsuperscript{3},
Vinay Namboodiri\textsuperscript{1,4} and Piyush Rai\textsuperscript{1}
\textsuperscript{1}Indian Institute of Technology Kanpur, India
\textsuperscript{2}Duke University, United States
\textsuperscript{3}Indian Institute of Technology Roorkee, India
\textsuperscript{4}University of Bath, United Kingdom
asadkarim0938@gmail.com, vinaykumar.verma@duke.edu, pravendra.singh@cs.iitr.ac.in, vpn22@bath.ac.uk, piyush@cse.iitk.ac.in

Abstract

We propose a novel approach for class incremental online learning in a limited data setting. This problem setting is challenging because of the following constraints: (1) Classes are given incrementally, which necessitates a class incremental learning approach; (2) Data for each class is given in an online fashion, i.e., each training example is seen only once during training; (3) Each class has very few training examples; and (4) We do not use or assume access to any replay/memory to store data from previous classes. Therefore, in this setting, we have to handle twofold problems of catastrophic forgetting and overfitting. In our approach, we learn robust representations that are generalizable across tasks without suffering from the problems of catastrophic forgetting and overfitting to accommodate future classes with limited samples. Our proposed method leverages the meta-learning framework with knowledge consolidation. The meta-learning framework helps the model for rapid learning when samples appear in an online fashion. Simultaneously, knowledge consolidation helps to learn a robust representation against forgetting under online updates to facilitate future learning. Our approach significantly outperforms other methods on several benchmarks.

1 Introduction

Deep neural networks have achieved promising results on various tasks. However, these models suffer from the problem of catastrophic forgetting [Kirkpatrick et al., 2017]. The most prominent reason for catastrophic forgetting is that the model is not trained to also remember the previous knowledge when acquiring new knowledge. In general, the model is trained to optimize its performance on the current task with no consideration of how the updated model will perform on earlier tasks. This greedy update overwrites the parameter values that may have been optimal for previous tasks. Continual-learning (CL) (also sometimes referred to as lifelong/incremental learning) is a learning paradigm to address this issue in deep neural networks and has been gaining significant attention in recent work [Paris et al., 2019].

We present a novel approach for class incremental online learning problem in a limited data setting. This problem setting is more challenging than standard class incremental learning [Javed and White, 2019] due to additional constraints: (1) Data in each class appears in the online fashion, i.e., the model sees every training example exactly once; (2) The number of training examples in each class is very small; and (3) We do not use any replay/memory to store the training examples from previous classes. This is the most general setting for class incremental learning and various practical usage scenario can be obtained through this or a relaxed setting. For instance, in face recognition, it is common to have few examples per class but usually not in an online learning fashion, whereas for a robot navigating in an environment, the setting would also be online. We empirically show that learning a robust representation that can accommodate future tasks may be a potential solution to handle the problem mentioned above. Our proposed approach achieves this by leveraging the meta-learning [Finn et al., 2017] framework with knowledge consolidation.

Meta-learning [Finn et al., 2017] has proven to be an effective approach for learning generic feature representations that can be rapidly adapted to new tasks by fine-tuning using very few examples (and in some cases, even without fine-tuning [Vinyals et al., 2016; Raghu et al., 2020]). While such use of meta-learning might seem appealing and does indeed show some promising results in continual learning settings [Javed and White, 2019], in practice, this approach is still prone to the problem of catastrophic forgetting. One of the reasons for this is the overparameterized nature of deep neural networks, in which only a few neurons are activated/fired for all samples. As a result, the network is reliant only on a small set of parameters. Although this may not be a problem when learning only a single task, it can potentially be an issue in continual learning where we are required to learn a sequence of tasks and, while learning a new task, any changes to these parameters can drastically affect the performance on the older tasks.

*Contact Author
our approach Knowledge Consolidation based Class Incremental Online Learning (KCCIOL) is better than memory-based rehearsal methods. Our approach can also be integrated with existing continual learning approaches such as MER, EWC, ER-Reservoir as shown in Section 7.

2 Problem Formulation and Evaluation Protocol for Novel Class Set Testing
Let \{\tau_1, \tau_2, \ldots, \tau_k \ldots, \tau_i \ldots\} denote a stream of learning-trajectories and \(\forall i, \tau_i \sim P_{\text{test}}(\tau)\) where \(P_{\text{test}}(\tau)\) denotes the trajectory distribution during testing from the novel class set. Each learning-trajectory \(\tau_i\) is further split into two sets – train and validation, i.e., \(\tau_i = \{\tau_{tr}, \tau_{val}\}\) where \(\tau_{tr}\) and \(\tau_{val}\) are the labeled samples, \(\{k, s\} \in \mathbb{N}\). Here, \(\forall n, (x_n, y_n) \in (X_{\text{test}}, Y_{\text{test}})\) denote the input and label pairs and the trajectory distribution \(P_{\text{test}}(\tau)\) is defined over \((X_{\text{test}}, Y_{\text{test}})\) which is disjoint from base training class set \((X_{\text{train}}, Y_{\text{train}})\) i.e. \(Y_{\text{train}} \cap Y_{\text{test}} = \emptyset\). Moreover, we assume \(\text{class}(\tau_{tr}) = \text{class}(\tau_{val})\) i.e. classes of \(\tau_{tr}\) are the same as classes of \(\tau_{val}\). The goal of continual learning is to minimize the loss on the unseen examples of classes learnt earlier in an incremental fashion, and can be written as: 
\[
\mathbb{E}_{\tau \sim P_{\text{test}}(\tau)}[\mathcal{L}(f_{\tau_{val}}, \theta, W)]
\]
where model evaluation on \(\tau_{val}\). Our evaluation protocol (Algorithm 4) is similar to the class-incremental setting [Javed and White, 2019], but samples within each class also arrive in an online manner. In particular, here are the key differences: 1) We assume availability of very few samples per class; 2) For a particular class, each sample is seen exactly once; and 3) We do not use any replay mechanism. These differences make our problem setting considerably more challenging than the standard class-incremental setting.

3 Training on Base Class Set using Meta Learning Approach
Following a similar notation as in Sec. 2, let \{\tau_1, \tau_2, \ldots, \tau_k \ldots, \tau_i \ldots\} denote a stream of learning-trajectories and \(\forall i, \tau_i \sim P_{\text{train}}(\tau)\) where \(P_{\text{train}}(\tau)\) denotes the learning trajectory distribution during training. Following the model-agnostic meta learning (MAML) set-up [Finn et al., 2017], we assume that the \(i^{th}\) learning-trajectory’s data \(\tau_i\) is further split into two sets, meta-train and meta-val, i.e. \(\tau_i = \{\tau_{tr}, \tau_{val}\}\) where \(\tau_{tr} = \{x_n, y_n\}_{n=1}^{k}\) and
\(\tau_{\text{val}} = \{x_n, y_n\}_{n=k+1}^{k+s}\) are the labeled samples, \(\{k, s\} \in \mathbb{N}\). Here, \(\forall n, (x_n, y_n) \in (X_{\text{train}}, Y_{\text{train}})\) denote the input and label pairs and the learning trajectory distribution \(P_{\text{train}}(\tau)\) is defined over \((X_{\text{train}}, Y_{\text{train}})\). Moreover, we assume \(\text{class}(\tau_{\text{tr}}) \subseteq \text{class}(\tau_{\text{val}})\) i.e. classes of \(\tau_{\text{tr}}\) are a proper subset of \(\tau_{\text{val}}\); therefore \(\tau_{\text{val}}\) contains all the classes of \(\tau_{\text{tr}}\) and some additional classes. Note that \(\tau_{\text{val}}\) and \(\tau_{\text{tr}}\) are the random trajectories of length \(k\) and \(s\), respectively. In \(\tau_{\text{tr}}\), samples of each class occur together, i.e samples from class 1 occur after and before samples from class 1 and class 3, respectively. We sample a learning trajectory multiple times for training on a base class set. We randomly sample a learning trajectory by selecting a subset (proper) of classes (randomly) from the base class set. Therefore, every learning trajectory has a different class order.

We follow the MAML setting of continual learning [Javed and White, 2019]. In the inner loop, each class arrives in a sequential manner, and the boundary of each class is available in advance. In the outer-loop, the aim is to minimize the empirical risk over the unseen data, provided the model is optimized on the seen data in an online fashion with the corresponding true labels. \(\text{tr}\) denotes the loss on the model \(f\) for the validation trajectory of \(\tau\). For notational brevity (and slight abuse of notation), we use \(\tau_{\text{val}}\) to refer to all the inputs of validation trajectory of task \(\tau\) and \(\tau_{\text{val}}^y\) to refer to the corresponding true labels.

The function \(f : \mathcal{X} \rightarrow \mathcal{Y}\) is defined as \(f(\tau, \theta, W) = g(h(\tau||\theta)|W)\) where \(h : \mathcal{X} \rightarrow \mathbb{R}^d\) is defined by parameter \(\theta\) (representation learning parameters) and \(g : \mathbb{R}^d \rightarrow \mathcal{Y}\) is defined by \(W\). The classifier parameters \(W\) are learned using meta-train set \((\tau_{\text{tr}})\), and representation learning parameters \(\theta\) and \(W\) are jointly learned using the meta-val set \((\tau_{\text{val}})\).

In the inner-loop of the meta-learner, which learns \(W\), the model is trained on the meta-train data \(\tau_{\text{tr}}\). The outer loop, which learns \(\theta\) and \(W\) is trained using meta-val data \(\tau_{\text{val}}\). In the inner loop, the model’s loss is also computed on novel classes not seen during the inner-loop training since the classes in \(\tau_{\text{tr}}\) are a proper subset of classes in \(\tau_{\text{val}}\). Evaluation on both sets \(\tau_{\text{tr}}\) and \(\tau_{\text{val}}\) makes the model perform well on both current and previously learned classes. The optimization problems solved by the inner loop and the outer loop are given by:

\[
W = \arg\min_{W} l_{\text{tr}}(\theta, W) = \mathcal{L}(f(\tau_{\text{tr}}, \theta, W), \tau_{\text{tr}}^y)
\]

\[
(\theta, W) = \arg\min_{\theta, W} l_{\text{val}}(\theta, W) = \mathcal{L}(f(\tau_{\text{val}}, \theta, W), \tau_{\text{val}}^y)
\]

In Eq. 2, for notational simplicity, we use the entire training data but during training we perform an online update over \(\tau_{\text{tr}}\). The above two optimization problems are solved in an alternating fashion using \(\tau_{\text{tr}}\) and \(\tau_{\text{val}}\), respectively, with the most recent parameter \(W\) obtained from Eq. 2 used in Eq. 3.

### 4 Knowledge Consolidation based Meta-Learning for Training on Base Class Set

While the meta-learning based approach for continual learning described in the above section is promising, it is not particularly effective for our problem setting. One of the reasons for this is the overparametrized nature of deep neural networks, in which only a few neurons are activated/fired for all samples. As a result, the network is reliant only on a small set of parameters. Although this may not be a problem for a single task learning setting, it can potentially be an issue in continual learning where we are required to learn a sequence of tasks and, while learning a new task, any changes to these parameters can drastically affect the performance on the older tasks. We overcome this by using a knowledge consolidation based meta-learning approach. Our proposed approach identifies the important and unimportant/dead parameters, rejuvenates the dead parameters, and consolidates the knowledge of the important and reborn parameters (Fig. 1). Therefore, the model capacity is fully utilized, and small changes in a few parameters strongly resist the catastrophic forgetting.

A simple way to assess the importance of a parameter is to use its absolute value [Han et al., 2015], often used in deep model compression. We can discard weights/parameters having small absolute values without sacrificing upon the model’s performance [Han et al., 2015]. We leverage this simple idea to identify the important parameters in the model effectively. The proposed approach modifies the meta-learning framework by introducing knowledge consolidation. We define \(A = [\theta, W]\) as the joint set of parameters of the complete model. We partition model parameters \(A\) into two disjoint sets. The important parameters are denoted by \(A_I\) and the “less important” ones are denoted by \(A_L\), s.t., \(A = \{A_I, A_L\}\) and \(A_I \cap A_L = \emptyset\).

Most of the model’s knowledge is contained in \(A_I\). Our goal is to preserve the knowledge present in \(A_I\). We apply a weight-constrained regularization on \(A_I\) to ensure minimal changes when a new trajectory is learned. On the other hand, we let \(A_L\) be free to change in order to accommodate new trajectories. Therefore, while learning a new set of trajectories, the following regularized loss function is optimized:

\[
\sum_{\tau \sim P_{\text{train}}(\tau)} \mathcal{L}(f(\tau_{\text{val}}^{I\tau}, \theta, W), \tau_{\text{val}}^{y\tau}) + \mathcal{R}(A_I)\]

One way to define the weight-constrained regularization \(\mathcal{R}(A_I)\) would be \(\lambda||A_I^{t+1} - A_I^{t}||_F\), where \(A_I^{t}\) is the important weights after \(t^{th}\) step. The large value of \(\lambda\) ensures minimal changes in the important weights.

Naïvely partitioning the model into \(\{A_I, A_L\}\) (based on absolute value) often does not show any significant improvement since various techniques like dropout and batch normalization force the model’s knowledge to be shared across all model parameters, which causes \(A_L\) to contain non-negligible knowledge. Ideally, the value of unimportant weights should be zero. However, in reality, this is not the case and the set \(A_L\) usually contains the non-negligible information. Therefore, we first distill the model’s knowledge into a subset of parameters \(A_I^{j}\) (important parameters set) such that the remaining part \(A_L^{j}\) (unimportant parameters set) contains negligible information.
The weight-constrained regularization can now be imposed on \( A^n_j \), while the set \( A^s_j \) is free to be adapted for the new trajectories. To transfer/distill the model’s knowledge to a subset of the parameters, we fine-tune the complete model with the following \( \ell_1 \) regularized objective:

\[
\sum_{r \sim \text{train}^i(\tau)} \mathcal{L}(f(\tau^r_{\text{val}}; \theta, W), \tau^y_{\text{val}}) + \gamma ||A||_1 \tag{5}
\]

We can maintain model performance by using an appropriate hyperparameter \( \gamma \). The \( \ell_1 \) regularizer forces the model knowledge to be squeezed in a subset of model parameters \( A^s_j \). Rest of the model parameters \( A^n_j \) contain negligible information and therefore are free to change. Now, the set \( A \) can be split into important parameters \( A^s_j \) and unimportant parameters set \( A^n_j \), i.e., \( A = \{ A^s_j, A^n_j \} \) and \( A^s_j \cap A^n_j = \emptyset \). Given this updated set of important and unimportant/dead set of parameters, the outer-loop optimization of the meta-learner is given by (akin to Eq-4)

\[
(\theta, W) = \arg \min_{\theta, W} l_{\text{val}}(\theta, W) \text{def} \mathcal{L}(f(\tau^r_{\text{val}}; \theta, W), \tau^y_{\text{val}}) + \lambda ||A^{s+1}_j - A^s_j||_F \tag{6}
\]

\[
A^{s+1}_j - A^s_j = \nabla A^{s+1}_j(\mathcal{L}(f(\tau^r_{\text{val}}; \theta, W), \tau^y_{\text{val}})) \tag{7}
\]

To preserve the knowledge contained in \( A^s_j \), we apply weight-constrained regularization on \( A^s_j \) as above, which ensures that \( A^s_j \) do not change drastically when new learning-trajectories are encountered. Rest of the parameters (\( A^n_j \)) are free to change. Therefore, we “rejuvenate” the parameters in \( A^n_j \). Representing these rejuvenated set of parameters as \( A^n_{jk} \), now the consolidated knowledge from both \( A^s_j \) and \( A^n_{jk} \) provides a robust representation for our problem setting. Therefore, small changes in a few parameters strongly resist catastrophic forgetting because the model capacity is fully utilized, and model predictions are not reliant on a small set of parameters. As demonstrated by our experiments, such a parameter rejuvenation and knowledge consolidation significantly enhance the performance of a meta-learner based model for class incremental online learning. For a summarized algorithmic description of our approach, please refer to the Algorithms 1, 2, 3, 4.

### 5 Related Work

**Incremental Learning Methods:** Rehearsal based incremental learning methods [Rebuffi et al., 2017; Shin et al., 2017; Chaudhry et al., 2019; Isele and Cosgun, 2018; Rolnick et al., 2019; Chaudhry et al., 2021] store a part of training data and re-train on this stored data while training on new tasks. Regularization based incremental learning methods [Lopez-Paz and Ranzato, 2017; Kirkpatrick et al., 2017; Chaudhry et al., 2019] add an additional regularization term in the loss function, which prevents the weights from changing drastically when moving from one task to the next task. DER [Buzzeza et al., 2020] uses both rehearsal and regularization. PODNet [Douillard et al., 2020] uses a spatial distillation-loss along with a representation made of proxy vectors from each class. The dynamic network methods [Singh et al., 2020; Singh et al., 2021] are also proposed for incremental learning.

---

**Algorithm 1 Training Algorithm**

**Require:** \( \gamma \): Learning coefficient for L1 loss  
**Require:** \( \lambda \): Learning coefficient for constraint loss  
**Require:** \( \delta \): Fraction of model parameters \( \{ A^n_j \} \)  
**Require:** \( \{ \alpha_i, \beta_i \} \): Inner and outer loop lr (learning rate)  
**Require:** \( \{ \text{steps} \} \): No of steps for KCCIOL algorithm

1: Randomly Initialize model parameters \( \Theta \),  
2: \( \text{mask} = \emptyset \)  
3: \( \Theta = \text{KCCIOL}(\alpha_1, \beta_1, 0, 0, \text{mask}, \Theta, \text{steps1}) \)  
4: \( \Theta = \text{KCCIOL}(\alpha_2, \beta_2, 0, \gamma, \text{mask}, \Theta, \text{steps2}) \)  
5: \( \text{mask} = \text{GetMask}(\Theta, \delta) \)  
6: \( \Theta = \text{KCCIOL}(\alpha_3, \beta_3, \lambda, 0, \text{mask}, \Theta, \text{steps3}) \)

**Algorithm 2 KCCIOL**

**Require:** \( p_{\text{train}}(\tau) \): Distribution over learning trajectories  
**Require:** \( \text{mask} \): Index matrix  
**Require:** \( \Theta \): Model parameters

1: for \( i = 1, \ldots, \text{steps} \) do  
2: Sample learning trajectory \( \tau_i \sim p_{\text{train}}(\tau) \)  
3: \( \{ \tau_{tr}, \tau_{val} \} = \tau_i \)  
4: \( \Theta, W = \Theta \)  
5: \( W_0 = W \)  
6: for \( j=1, 2, \ldots, k \) do  
7: \( (X_j, Y_j) = (\tau_{tr}[j], \tau_{tr}[j]) \)  
8: \( W_j = W_{j-1} - \alpha \nabla W_{j-1} \mathcal{L}(f(X_j; \theta, W_{j-1}), Y_j) \)  
9: \( l_{\text{meta}} = \mathcal{L}(f(\tau^r_{\text{val}}; \theta, W_k), \tau^y_{\text{val}}) \)  
10: \( l_{\text{constraint}} = ||\text{mask} \times \nabla_{\theta, W} l_{\text{meta}}||_2 \)  
11: \( l_1 = ||\Theta||_1 \)  
12: Update \( \Theta \leftarrow \Theta - \beta \nabla_{\theta, W} (l_{\text{meta}} + \lambda l_{\text{constraint}} + \gamma l_1) \)  
13: return \( \Theta \)

**Algorithm 3 Mask Calculation**

**Require:** \( \Theta \): Model Parameters  
**Require:** \( \delta \): Fraction of model parameters \( \{ A^n_j \} \)

1: procedure GETMASK(\( \Theta \), \( \delta \))  
2: \( \text{threshold} = \text{percentile}(||\Theta||_1, 1 - \delta) \)  
3: \( \text{mask} = \text{zeros}(\text{len}(\Theta)) \)  
4: \( \text{index} = 0 \)  
5: while \( \text{index} <= \text{len}(\Theta) \) do  
6: if \( \text{mask}[\text{index}] >= \text{threshold} \) then  
7: \( \text{mask}[\text{index}] = 1 \)  
8: \( \text{index} = + \)  
9: return \( \text{mask} \)

**Algorithm 4 Evaluation Protocol**

**Require:** \( p_{\text{test}}(\tau) \): Distribution over learning trajectories  
**Require:** \( \theta \): Representation Learning Parameters

1: Randomly Initialize \( W \)  
2: Sample learning trajectory \( \tau_i \sim p_{\text{test}}(\tau) \)  
3: \( \{ \tau_{tr}, \tau_{val} \} = \tau_i \)  
4: for \( j=1, \ldots, k \) do  
5: \( (X_j, Y_j) = (\tau_{tr}[j], \tau_{tr}[j]) \)  
6: \( W_j = W_{j-1} - \alpha \nabla W_{j-1} \mathcal{L}(f(X_j; \theta, W_{j-1}), Y_j) \)  
7: return \( \text{Accuracy}(f(\tau^r_{\text{val}}; \theta, W_k), \tau^y_{\text{val}}) \)
Meta-Learning Methods: MER [Riemer et al., 2019] uses meta-learning to learn parameters that prevent interference and enable knowledge transfer based on future gradients. OML [Javed and White, 2019] is a meta-learning approach that focuses on learning a generic representation that prevents catastrophic forgetting during online learning. OSAKA [Caccia et al., 2020] is a general approach to continual learning where the agent must be able to solve previously unseen distribution tasks with minimal forgetting on previous tasks. OSAKA is different from the standard incremental learning setting as seen tasks can be revisited, and online average accuracy is reported at the end of the training instead of reporting accuracy on all seen tasks. Therefore, it would be unfair to compare our method with OSAKA.

Online Continual Learning Methods: MERLIN [K J and Balasubramanian, 2020] is a replay based method for online continual learning which learns a meta-distribution from which task specific parameters are sampled at the time of inference. [von Oswald et al., 2020] proposed a method similar to MERLIN where deterministic task specific weights are generated using hypernetworks [Ha et al., 2017]. GSS uses constrained optimization for sampling samples for replay instead of random sampling for online continual learning. Incremental learning in online scenario [He et al., 2020] tackles the problem of catastrophic forgetting in the online scenario under different setting than ours. [He et al., 2020] is a memory based method where future data consists of samples from new classes as well as unseen samples from old classes, whereas, in our approach, data from one class are seen together. CTN [Pham et al., 2021] is a bi-level optimization network that uses a context network to model task-specific features which address catastrophic forgetting and knowledge transfer. But the context network needs task-specific knowledge and semantic memory to function, whereas our method does use replay and is task-free.

6 Experiments
We evaluate our approach via extensive experiments across various datasets. We follow the evaluation protocol where the model is updated in an online fashion and later evaluated on the unseen data (Section 2). We compare the performance of our model (KCCIOL) against several baselines.

Baselines: In the Scratch baseline, we evaluate the performance of a randomly initialized network. In Pretrained baseline, the network is pretrained on the train set. The SRNN approach uses a Set-KL method proposed by [Liu et al., 2019] to learn a sparse representation using train set. MRCL is a recent approach originally proposed by [Javed and White, 2019] to train a model in a meta-learning setup for continual learning tasks. OML is a modified version of MRCL proposed by [Javed and White, 2019] where the classifier parameters are randomly re-initialized at each step of the training. MAML-Rep is also a MAML [Finn et al., 2017] based algorithm similar to OML and MRCL, where batch updates are performed in the inner-loop.

6.1 Experiments on Omniglot Dataset:
Implementation Details: The Omniglot dataset [Lake et al., 2015] contains 1623 classes of different handwritten characters from 50 different alphabets. Each class contains 20 samples with 15/5 as the train/test split. The first 963 classes constitute the \((X_{train}, Y_{train})\) and the remaining classes are used as \((X_{test}, Y_{test})\). For learning trajectory during training, the \(\tau_{tr}\) consists of 10 samples from a class randomly sampled from the training set. The \(\tau_{val}\) consists of 10+1 samples where ten samples are randomly sampled from the train set, and the 11th sample belongs to the class used in \(\tau_{tr}\). During the evaluation, each learning trajectory consists of an equal number of classes in both \(\tau_{tr}\) and \(\tau_{val}\) which are sampled from the test set. We use 15/5 samples per class for \(\tau_{tr}\) during evaluation. We use Adam optimizer, and the first six layers are used for learning representation.

Hyperparameter Settings: We train our model using hyperparameters: \(\beta_1 = 1e-4, \alpha_1 = 1e-2, steps_{tr} = 20000, \beta_2 = 1e-4, \alpha_2 = 1e-2, \gamma = 5e-5, steps_{val} = 15000, \beta_3 = 1e-4, \alpha_3 = 1e-2, \lambda = 5e-4, steps_{test} = 4000, \delta = 0.5\).

Model Architecture: We use six convolutional layers followed by two fully connected layers, and each convolutional layer contains 256 filters of \(3 \times 3\) kernel size with \(2, 1, 2, 1, 2\) strides (same as used in [Javed and White, 2019]). ReLU activation function is used for the non-linearity.

The proposed approach KCCIOL performs better by a significant margin on a wide range of classes as compared to the other baselines. We achieve a significant performance boost of 13.3%, when 200 classes are learnt continually as shown in Table 1.

6.2 Experiments on Mini-Imagenet Dataset
Implementation Details: Vinyals et al. [Vinyals et al., 2016] proposed the mini-imagenet dataset, which is a subset of the imagenet dataset. There are a total of 100 classes with 600 colored images of size \(84 \times 84\). We use 64 classes for training and 20 classes for testing. For learning trajectory during training, the \(\tau_{tr}\) consists of 10 samples from a class randomly sampled from the training set. The \(\tau_{val}\) consists of 15 samples where 10 samples are randomly sampled from the training set, and 5 samples belong to the class used in \(\tau_{tr}\). During the evaluation, we sample classes from the test set for creating a learning trajectory. We use 30 samples per class for \(\tau_{tr}\) and \(\tau_{val}\). We use Adam optimizer, and the first six layers are used for learning representations.

Hyperparameter Settings: We train our model using hyperparameters: \(\beta_1 = 1e-4, \alpha_1 = 1e-1, steps_{tr} = 26000, \beta_2 = 1e-4, \alpha_2 = 1e-1, \gamma = 1e-4, steps_{val} = 26000, \beta_3 = 1e-4, \alpha_3 = 0.5, \lambda = 1e-4, steps_{test} = 22000, \delta = 0.5\).

Model Architecture: The model architecture for the Mini-Imagenet is the same as used in the Omniglot dataset.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Scratch</th>
<th>Pretrained</th>
<th>SRNN</th>
<th>MRCL</th>
<th>MAML-Rep</th>
<th>OML</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15.9 ± 3.5</td>
<td>42.6 ± 10</td>
<td>70.4</td>
<td>83.8 ± 6.2</td>
<td>86.1</td>
<td>92.6 ± 3.5</td>
<td>95.4 ± 3.4</td>
</tr>
<tr>
<td>50</td>
<td>2.4 ± 1.4</td>
<td>24.4 ± 4.3</td>
<td>53.9</td>
<td>66.5 ± 4.0</td>
<td>71.3</td>
<td>81.3 ± 2.4</td>
<td>85.8 ± 2.5</td>
</tr>
<tr>
<td>100</td>
<td>1.5 ± 0.3</td>
<td>15.5 ± 1.9</td>
<td>44.3</td>
<td>51.8 ± 2.6</td>
<td>70.0</td>
<td>76.1 ± 2.0</td>
<td>81.4 ± 2.3</td>
</tr>
<tr>
<td>150</td>
<td>1.2 ± 0.5</td>
<td>11.7 ± 1.1</td>
<td>27.0</td>
<td>42.8 ± 2.5</td>
<td>53.0</td>
<td>65.2 ± 2.2</td>
<td>77.1 ± 1.6</td>
</tr>
<tr>
<td>200</td>
<td>0.8 ± 0.5</td>
<td>8.0 ± 1.1</td>
<td>18.3</td>
<td>33.8 ± 1.8</td>
<td>35.7</td>
<td>59.3 ± 1.8</td>
<td>72.6 ± 1.5</td>
</tr>
</tbody>
</table>

Table 1: Classification accuracy (mean±std) on the omniglot dataset averaged across 50 test trajectories randomly sampled from the meta-test set. Classes column refers to the total number of classes in the sampled trajectory.
We have demonstrated the efficiency of KCCIOL when the difference between the Pre-Training and Standard setting is shown in Table 2. We get a performance boost up to 12.4% use 15/5 samples per class. Even without using any memory/replay, KCCIOL can outperform the replay based methods by a significant margin.

### 7 KCCIOL Complements Existing Continual Learning Methods

We have demonstrated the efficiency of KCCIOL when learning-trajectories are learnt in a continual online fashion. In this section, we will demonstrate that the representation generated by KCCIOL not only performs well on online updates but it also greatly improves the performance of other continual learning methods such as MER [Rolnick et al., 2019], EWC [Kirkpatrick et al., 2017], ER-Reservoir [Rolnick et al., 2019] when they use our model as base network. Online, Approx IID baselines have been taken from [Javed and White, 2019].

We evaluate the model’s performance under four different settings for each continual learning method, which are Standard, Pre-Training, OML, and KCCIOL. In OML and KCCIOL, we use the $\theta$ generated by these methods as the base of the model and do not update the $\theta$ parameters. In the Pre-Training setting, we train the model independently on the training data and again keep the $\theta$ fixed during the process. The difference between the Pre-Training and Standard setting is that all the parameters of the model are updated in the case of the Standard setting, while in the pre-training, only W are updated. In the standard setting, the model is pretrained independently on train data to prevent other settings from having an unfair advantage of being trained on the training set. From Tables-3, 4 we can observe that KCCIOL with just online updates significantly outperforms other continual learning methods even when they are combined with OML. KCCIOL can do so because it can learn a generic robust feature representation that is generalizable across all tasks, enabling the model to perform future learning. For the train/test split, we use 15/5 samples per class. Even without using any memory/replay, KCCIOL can outperform the replay based methods by a significant margin.

### 8 Performance Comparison with Recent SOTA Incremental Learning Methods

We also compare our method KCCIOL with the recent state-of-the-art incremental learning methods on the mini-imagenet dataset. Classification accuracy (mean±std) is averaged across 50 test trajectories randomly sampled from the meta-test set. Classes column refers to the total number of classes in the sampled trajectory.

Table 5: Performance comparison with recent SOTA incremental learning methods on the mini-imagenet dataset. Classification accuracy (mean±std) is averaged across 50 test trajectories randomly sampled from the meta-test set. Classes column refers to the total number of classes in the sampled trajectory.

<table>
<thead>
<tr>
<th>Classes</th>
<th>DER</th>
<th>DER++</th>
<th>HAL</th>
<th>MERLIN</th>
<th>GSS</th>
<th>PODNet</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>27.22 ± 6.39</td>
<td>36.77 ± 5.73</td>
<td>26.33 ± 3.44</td>
<td>22.78</td>
<td>35.00</td>
<td>42.68 ± 2.33</td>
<td>49.4 ± 5.5</td>
</tr>
<tr>
<td>8</td>
<td>22.29 ± 5.42</td>
<td>28.13 ± 3.46</td>
<td>20.63 ± 3.11</td>
<td>14.16</td>
<td>20.00</td>
<td>38.10 ± 2.00</td>
<td>45.3 ± 4.9</td>
</tr>
<tr>
<td>10</td>
<td>19.0 ± 4.69</td>
<td>23.5 ± 4.14</td>
<td>15.93 ± 4.14</td>
<td>11.24</td>
<td>24.33</td>
<td>31.48 ± 1.56</td>
<td>42.7 ± 4.6</td>
</tr>
<tr>
<td>12</td>
<td>15.58 ± 3.35</td>
<td>19.14 ± 3.51</td>
<td>14.86 ± 2.45</td>
<td>9.42</td>
<td>17.78</td>
<td>27.94 ± 1.42</td>
<td>39.4 ± 3.0</td>
</tr>
<tr>
<td>14</td>
<td>13.93 ± 2.18</td>
<td>17.14 ± 3.27</td>
<td>12.36 ± 1.84</td>
<td>7.53</td>
<td>17.14</td>
<td>25.15 ± 1.29</td>
<td>37.0 ± 3.7</td>
</tr>
<tr>
<td>16</td>
<td>13.33 ± 2.83</td>
<td>14.21 ± 3.83</td>
<td>10.75 ± 1.56</td>
<td>6.83</td>
<td>11.87</td>
<td>22.93 ± 1.29</td>
<td>33.4 ± 2.1</td>
</tr>
<tr>
<td>18</td>
<td>11.48 ± 1.97</td>
<td>13.15 ± 2.96</td>
<td>9.46 ± 1.55</td>
<td>6.16</td>
<td>12.41</td>
<td>21.14 ± 1.22</td>
<td>31.9 ± 2.2</td>
</tr>
<tr>
<td>20</td>
<td>9.68 ± 3.07</td>
<td>12.48 ± 2.08</td>
<td>8.9 ± 1.50</td>
<td>6.11</td>
<td>13.00</td>
<td>19.63 ± 1.14</td>
<td>29.7 ± 1.4</td>
</tr>
</tbody>
</table>

Our method KCCIOL consistently outperforms others as shown in Table 2. We get a performance boost up to 12.4% over the OML.

<table>
<thead>
<tr>
<th>Method</th>
<th>Standard</th>
<th>Pre-Training</th>
<th>OML</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online</td>
<td>4.64 ± 2.61</td>
<td>21.16 ± 2.71</td>
<td>64.72 ± 2.57</td>
<td>85.68 ± 2.10</td>
</tr>
<tr>
<td>Approx. IID</td>
<td>53.95 ± 5.50</td>
<td>54.29 ± 3.48</td>
<td>75.12 ± 3.24</td>
<td>88.66 ± 2.10</td>
</tr>
<tr>
<td>MER</td>
<td>54.88 ± 4.12</td>
<td>62.76 ± 2.16</td>
<td>76.00 ± 2.07</td>
<td>91.28 ± 1.38</td>
</tr>
<tr>
<td>EWC</td>
<td>5.08 ± 2.47</td>
<td>18.72 ± 3.97</td>
<td>64.44 ± 3.13</td>
<td>87.10 ± 1.40</td>
</tr>
<tr>
<td>ER-Reservoir</td>
<td>52.56 ± 2.12</td>
<td>36.72 ± 3.06</td>
<td>68.16 ± 3.12</td>
<td>90.10 ± 1.35</td>
</tr>
</tbody>
</table>

Table 3: KCCIOL combined with existing continual learning methods on the omniglot dataset. We use 50 tasks with 1 class per task. The accuracies are averaged over 10 runs.

Table 4: KCCIOL combined with existing continual learning methods on omniglot. We use 100 tasks with 5 classes per task. The accuracies are averaged over 10 runs.
References


