Knowledge Consolidation based Class Incremental
Online Learning with Limited Data

Mohammed Asad Karim$^{1*}$, Vinay Kumar Verma$^{2}$, Pravendra Singh$^{3}$, Vinay Namboodiri$^{1,4}$ and Piyush Rai$^{1}$

$^{1}$Indian Institute of Technology Kanpur, India
$^{2}$Duke University, United States
$^{3}$Indian Institute of Technology Roorkee, India
$^{4}$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 [Parisi 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.
To address this, we present a knowledge consolidation based meta-learning approach. In our approach, during training, we identify and split the network parameters into two groups, called important and unimportant. The network’s existing knowledge is squeezed into the set of important parameters, and the unimportant/dead parameters are freed, thereby expanding the network’s capacity to accommodate the future learning trajectories. Briefly, a learning trajectory is a sequence of examples where examples from a particular class occur together in the sequence (we discuss this in detail later). The proposed strategy ensures that the knowledge from the old learning trajectories is preserved in a compressed form within a small set of important network parameters, which we identify and isolate, and then move on to adopt new learning trajectories. The extra knowledge obtained via learning from the new trajectories updates the unimportant parameters, and they also become important. In knowledge consolidation, we rejuvenate the dead neurons in the model and consolidate the knowledge of the previously preserved parameters and the new rejuvenated parameters. This helps to learn a robust representation. Therefore, the model capacity is fully utilized, and small changes in a few parameters strongly resist the catastrophic forgetting. The knowledge consolidation process overview is shown in Fig. 1.

Note that we use knowledge consolidation and meta-learning in the training phase. Training is done on a base class set using multiple learning trajectories. We strictly follow class incremental online learning setting within a learning trajectory. However, across learning trajectories, the class incremental learning setting is not used (since multiple learning trajectories can have the same set of classes). Therefore, we do not follow the incremental learning setting during training on the base class set (since training is done on multiple learning trajectories). We follow this to make the model’s representation robust and facilitate future continual learning during evaluation time (Section 2), where we perform class incremental online learning with limited data on a novel class set. We use entirely different (disjoint) classes in the novel class set than the base class set, and during evaluation on the novel class set, we only use meta-learning for quick adaptation.

Our approach significantly outperforms other incremental learning methods by a significant margin. We show that a basic online updating strategy on representations learned by 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 \( \forall i, \tau_i \sim P_{test}(\tau) \) where \( P_{test}(\tau) \) 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 \( \tau_{tr} = \{x_n, y_n\}_{n=1}^{k+s} \) and \( \tau_{val} = \{x_n, y_n\}_{n=k+s+1}^{N} \) are the labeled samples, \( \{k, s\} \in \mathbb{N} \). Here, \( \forall n, (x_n, y_n) \in (X_{test}, Y_{test}) \) denote the input and label pairs and the trajectory distribution \( P_{test}(\tau) \) is defined over \( (X_{test}, Y_{test}) \) which is disjoint from base training class set \( (X_{train}, Y_{train}) \) i.e. \( Y_{train} \cap Y_{test} = \emptyset \). Moreover, we assume \( class(\tau_{tr}) = class(\tau_{val}) \) 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: \( \mathbb{E}_{\tau \sim P_{test}(\tau)}[\mathcal{L}(f_{\tau_{val}}(\theta, W), y_{\tau_{val}})] \) i.e model is evaluated 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 \( \forall i, \tau_i \sim P_{train}(\tau) \) where \( P_{train}(\tau) \) 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 \( \tau_{tr} = \{x_n, y_n\}_{n=1}^{k+s} \) and

![Diagram](image-url)
We sample a learning trajectory multiple times for training on \( \theta \) which learns \( g \) most recent parameter \( W \) alternating fashion using \( \tau \) data but during training we perform an online update over \( s \). The optimization problems solved by the inner loop and the outer loop are given by:

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

In Eq. 2, for notational simplicity, we use the entire training data but during training we perform an online update over \( \tau_{tr} \).

The above two optimization problems are solved in an alternating fashion using \( \tau_{tr} \) and \( \tau_{val} \), 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 relearned 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:

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

One way to define the weight-constrained regularization \( R(A_I) \) 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.

Naively 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 \( A_I^{*} \) (important parameters set) such that the remaining part \( A_L^{*} \) (unimportant parameters set) contains negligible information.
The weight-constrained regularization can now be imposed on $A_t^*$, while the set $A_t^*$ is free to be adapted for the new trajectories. To transfer/distill the model’s knowledge to a subset of the parameters, we finetune the complete model with the following $\ell_1$ regularized objective:

$$\sum_{\tau \sim p_{\text{train}}(\tau)} L(f(\tau_{\text{val}}^{\tau}|\theta, W), \tau_{\text{val}}^{\nu}) + \gamma||A||_1$$ (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 $A_t^*$. Rest of the model parameters $A_t^*$ contain negligible information and therefore are free to change. Now, the set $A$ can be split into important parameters $A_t^*$ and unimportant parameters set $A_t^t$, i.e., $A = \{A_t^*, A_t^t\}$ and $A_t^* \cap A_t^t = \emptyset$. 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)

$$(\theta, W) = \arg \min_{\theta, W} l_{\text{val}}(\theta, W) \overset{\text{def}}{=} L(f(\tau_{\text{val}}^{\nu}|\theta, W), \tau_{\text{val}}^{\nu})$$

$$+ \lambda||A_t^{t+1} - A_t^{t+1}|\|_F$$ (6)

$$A_t^{t+1} - A_t^t \approx \nabla_{A_t^t} L(f(\tau_{\text{val}}^{\nu}|\theta, W), \tau_{\text{val}}^{\nu})$$ (7)

To preserve the knowledge contained in $A_t^*$, we apply weight-constrained regularization on $A_t^*$ as above, which ensures that $A_t^*$ do not change drastically when new learning-trajectories are encountered. Rest of the parameters ($A_t^t$) are free to change. Therefore, we “rejuvenate” the parameters in $A_t^t$. Representing these rejuvenated set of parameters as $A_t^{tr}$, now the consolidated knowledge from both $A_t^{tr}$ and $A_t^*$ 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 train-data and knowledge consolidation significantly enhance the model predictions are not reliant on a small set of parameters. 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 [Buzaglo 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 ($\sum_{A_{tr}}^{A_t^*}$)

**Require:** $\{\alpha_t, \beta_t\}^3_{t=1}$: Inner and outer loop lr (learning rate)

**Require:** $\{\text{steps}^1\}^3_{t=1}$: No of steps for KCCIOL algorithm

1: Randomly Initialize model parameters $\Theta$
2: $\text{mask} = \emptyset$
3: $\Theta = \text{KCCIOL}(\alpha_t, \beta_t, 0, 0, \text{mask}, \Theta, \text{steps}^1)$
4: $\Theta = \text{KCCIOL}(\alpha_t, \beta_t, 0, \gamma, \text{mask}, \Theta, \text{steps}^2)$
5: $\text{mask} = \text{GetMask}(\Theta, \delta)$
6: $\Theta = \text{KCCIOL}(\alpha_t, \beta_t, \lambda, 0, \text{mask}, \Theta, \text{steps}^3)$

### 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 $j=1, 2, \ldots, \text{steps do}$
7: $(X_j, Y_j) = (\tau_{tr}[j], \tau_{val}[j])$
8: $W_j = W_{j-1} - \alpha \nabla W_{j-1}[L(f(X_j|\theta, W_{j-1}), Y_j)]$
9: $L_{\text{meta}} = L(f(\tau_{val}^{\nu}|\theta, W_k), \tau_{val}^{\nu})$
10: $L_{\text{constraint}} = ||\text{mask} \nabla_{\theta, \omega} L_{\text{meta}}||_2$  
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 ($\sum_{A_{tr}}^{A_t^*}$)

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 $\text{index} \leq \text{len}(\Theta)$ do
6: if $\text{mask}[\text{index}]$ $\geq$ 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 $j=1, \ldots, \text{steps do}$
5: $(X_j, Y_j) = (\tau_{tr}[j], \tau_{val}[j])$
6: $W_j = W_{j-1} - \alpha \nabla W_{j-1}[L(f(X_j|\theta, W_{j-1}), Y_j)]$
7: return $\text{Accuracy}(f(\tau_{val}^{\nu}|\theta, W_k), \tau_{val}^{\nu})$
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 SRN 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, $\tau_{tr}$ consists of 10 samples from a class randomly sampled from the training set. $\tau_{val}$ 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 $\tau_{tr/val}$ during evaluation. We use Adam optimizer, and the first six layers are used for learning representation.

Hyperparameter Settings: We train our model using hyperparameters: $\beta_1 = 1e-4$, $\alpha_1 = 1e-2$, $steps_1 = 20000$, $\beta_2 = 1e-4$, $\alpha_2 = 1e-2$, $\gamma = 5e-5$, $steps_2 = 15000$, $\beta_3 = 1e-4$, $\alpha_3 = 1e-2$, $\lambda = 5e-4$, $steps_3 = 4000$, $\delta = 0.5$.

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 $(2, 1, 2, 1, 2, 2)$ 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.

| Classes | Scratch | Pretrained | SRNN | MRCL | MAML-Rep | OML | Ours |
|---------|---------|------------|------|------|----------|-----|------|
| 10      | 15.9 ± 3.5 | 42.6 ± 10 | 70.4 | 83.8 ± 6.2 | 86.1 | 92.6 ± 1.5 | 95.4 ± 3.4 |
| 50      | 2.4 ± 1.4 | 24.4 ± 4.3 | 53.9 | 66.5 ± 4.0 | 71.3 | 81.3 ± 2.4 | 85.5 ± 2.5 |
| 100     | 1.5 ± 0.3 | 15.5 ± 1.9 | 44.3 | 51.8 ± 2.6 | 70.0 | 76.1 ± 2.0 | 81.4 ± 3.3 |
| 150     | 1.2 ± 0.5 | 11.7 ± 1.1 | 27.0 | 42.8 ± 2.5 | 53.0 | 65.2 ± 2.2 | 77.1 ± 1.6 |
| 200     | 0.8 ± 0.5 | 8.0 ± 1.1 | 18.3 | 33.8 ± 1.8 | 35.7 | 59.3 ± 1.8 | 72.6 ± 1.5 |

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 they use our model as base network. Online, Approx
The difference between the Pre-Training and Standard setting
Table 3: KCCIOL combined with existing continual learning methods even when they are combined with OML. KCCIOL
line updates significantly outperforms other continual learning
From Tables-3, 4 we can observe that KCCIOL with just on-
W
is that all the parameters of the model are updated in the case
Parameters. In the
Pre-Training setting, we train the model independently on the
training data and again keep the θ
Previously trained
θ
parameters. In the
Online, Approx. IID
approximated by KCCIOL not only performs well on online updates
learning methods such as GSS [Aljundi
er et al.
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.
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
per class in the sampled trajectory.
Table 2: Classification accuracy (mean±std) on mini-imagenet 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.

| Classes | Scratch | Pretrained | SRNN | MRCL | OML | Ours |
|---------|---------|------------|------|------|-----|------|
| 6       | 19.3±4  | 16.9       | 34.2 | 41.8±9 | 37.0±6.8 | 49.4±6.5 |
| 8       | 12.9±1  | 13.0       | 29.2 | 36.8±7.1 | 34.6±4.2 | 45.3±4.9 |
| 10      | 10.1±0.9| 10.0       | 25.8 | 34.4±6.5 | 33.2±4.6 | 42.7±4.6 |
| 12      | 8.5±0.9 | 8.5        | 23.1 | 30.6±4.9 | 29.5±3.2 | 39.4±3.0 |
| 14      | 9.1±1.9 | 8.8        | 20.4 | 26.9±4.5 | 27.8±2.5 | 37.0±3.7 |
| 16      | 7.4±1.8 | 10.8       | 18.1 | 28.0±3.5 | 25.2±2.0 | 33.4±2.1 |
| 18      | 5.5±0.7 | 10.4       | 17.7 | 26.3±3.3 | 23.9±2.1 | 31.9±2.2 |
| 20      | 6.2±1.4 | 10.0       | 17.3 | 25.5±2.8 | 22.9±1.6 | 29.7±1.4 |

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

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 θ generated by these methods as the base of the model and do not update the θ parameters. In the Pre-Training setting, we train the model independently on the training data and again keep the θ 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 mem-

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

| Method | Standard | Pre-Training | OML | Ours |
|--------|----------|--------------|-----|------|
| Online | 1.40±0.43| 11.80±1.92 | 55.32±2.25 | 80.10±1.71 |
| Approx. IID | 48.02±5.67 | 46.02±2.83 | 67.03±2.10 | 85.90±1.76 |
| MER    | 29.02±4.01| 42.05±3.71 | 62.05±2.19 | 83.42±1.67 |
| EWC    | 2.04±0.35| 10.03±1.53 | 56.03±3.20 | 82.90±1.27 |
| ER-Reservoir | 24.32±5.37 | 37.44±1.67 | 60.92±2.41 | 84.76±1.12 |

| Method | DER | DER++ | HAL | MERLIN | GSS | PODNet | Ours |
|--------|-----|-------|-----|--------|-----|--------|------|
| Online | 27.22±6.39 | 36.77±5.73 | 26.33±3.44 | 22.78 | 35.00 | 42.68 | 23.3 | 49.4 ±5.5 |
| 8      | 22.29±5.42 | 28.13±3.60 | 34.63±3.11 | 14.16 | 30.00 | 36.10 | 2.00 | 45.3 ±4.9 |
| 10     | 19.0±2.69  | 23.5±4.14  | 15.93±4.14 | 11.24 | 24.33 | 31.48 | 1.56 | 42.7 ±4.6 |
| 12     | 15.58±3.35 | 19.14±3.51 | 14.86±2.45 | 9.42  | 17.78 | 27.94 | 1.42 | 39.4 ±3.0 |
| 14     | 13.93±2.18 | 17.14±3.27 | 12.36±1.58 | 7.53  | 17.14 | 25.15 | 1.29 | 37.0 ±3.7 |
| 16     | 13.33±2.83 | 14.21±3.63 | 10.75±1.56 | 6.83  | 11.87 | 22.93 | 1.29 | 34.4 ±2.1 |
| 18     | 11.48±1.97 | 13.15±2.96 | 9.46±1.55  | 6.16  | 12.41 | 21.14 | 1.22 | 31.9 ±2.2 |
| 20     | 9.68±3.07  | 12.48±2.08 | 8.9±1.50  | 6.11  | 13.00 | 19.63 | 1.14 | 29.7 ±1.4 |

Table 5: Performance comparison with recent SOTA incremental learning methods on omniglot. We use 100 tasks with 5 classes per task. Classes column refers to the total number of classes in the sampled trajectory.

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 under incremental online learning setting. From Table 5, we can observe that recent incremental learning methods such as DER [Buzzega et al., 2020], PODNet [Douillard et al., 2020], HAL [Chaudhry et al., 2021] perform poorly in an incremental online setting. Even SOTA methods for online learning, such as GSS [Aljundi et al., 2019], MERLIN [K J and Balasubramanian, 2020] suffer from catastrophic forgetting when training data is limited. We only use thirty samples per class for experiments on the mini-imagenet dataset. We observe that KCCIOL outperforms the next best method by more than 10% when ten classes are learnt continually. Also, all of the above-mentioned methods are replay based approaches, while KCCIOL does not use any replay/memory.

9 Conclusion

We propose a novel approach to learn robust feature representations that are generalizable across future learning classes. Our approach uses a meta-learning framework with knowledge consolidation for learning generic feature representations that can be rapidly adapted for future classes without forgetting the previous classes under online updates to facilitate future learning. Our approach shows a significant improvement for class incremental online learning on several benchmark datasets.

Acknowledgements

PR thanks support from Qualcomm Innovation Fellowship and Visvesvaraya Young Faculty Fellowship.
References

[Aljundi et al., 2019] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In Advances in Neural Information Processing Systems, pages 11816–11825, 2019.

[Arjovsky et al., 2017] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.

[Buzzega et al., 2020] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. In Advances in Neural Information Processing Systems, 2020.

[Caccia et al., 2020] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexandre Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. In Advances in Neural Information Processing Systems, 2020.

[Chaudhry et al., 2019] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In International Conference on Learning Representations, 2019.

[Chaudhry et al., 2021] Arslan Chaudhry, Albert Gordo, Puneet K Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In Thirty-Fifth AAAI conference on artificial intelligence, 2021.

[Douillard et al., 2020] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In Proceedings of the IEEE European Conference on Computer Vision (ECCV), 2020.

[Finn et al., 2017] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1126–1135. JMLR. org, 2017.

[Ha et al., 2017] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In International Conference on Learning Representations, 2017.

[Han et al., 2015] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in neural information processing systems, pages 1133–1143, 2015.

[He et al., 2020] Jiangpeng He, Runyu Mao, Zeman Shao, and Fengqing Zhu. Incremental learning in online scenario. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[Isele and Cosgun, 2018] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In Thirty-second AAAI conference on artificial intelligence, 2018.

[Javed and White, 2019] Khurram Javed and Martha White. Meta-learning representations for continual learning. In Advances in Neural Information Processing Systems, pages 1818–1828, 2019.

[K J and Balasubramanian, 2020] Joseph K J and Vineeth Balasubramanian. Meta-consolidation for continual learning. In Advances in Neural Information Processing Systems, 2020.

[Kirkpatrick et al., 2017] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, pages 3521–3526, 2017.

[Krizhevsky and others, 2009] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[Lake et al., 2015] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. Science, pages 1332–1338, 2015.

[LeCun and Cortes, 2010] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[Liu et al., 2019] Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 4384–4391, 2019.

[Lopez-Paz and Ranzato, 2017] David Lopez-Paz and Marc’Aurelio Ranzato. Gradient episodic memory for continual learning. In Advances in Neural Information Processing Systems, pages 6467–6476, 2017.

[Parisi et al., 2019] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanar, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. Neural Networks, 2019.

[Pham et al., 2021] Quang Pham, Chenghao Liu, Doyen Sahoo, and Steven CH Ho. Contextual transformation networks for online continual learning. In International Conference on Learning Representations, 2021.

[Raghu et al., 2020] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In International Conference on Learning Representations, 2020.

[Rebuffi et al., 2017] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pages 2001–2010, 2017.

[Riemer et al., 2019] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In International Conference on Learning Representations, 2019.

[Rolnick et al., 2019] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In Advances in Neural Information Processing Systems, pages 348–358, 2019.

[Shin et al., 2017] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In Advances in Neural Information Processing Systems, pages 2990–2999, 2017.

[Singh et al., 2020] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. Advances in Neural Information Processing Systems, 33, 2020.

[Singh et al., 2021] Pravendra Singh, Pratik Mazumder, Piyush Rai, and Vinay P Namboodiri. Rectification-based knowledge retention for continual learning. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[Vinyals et al., 2016] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In Advances in neural information processing systems, pages 3630–3638, 2016.
10 Supplementary Material

10.1 Experiments on CIFAR-100 Dataset

Implementation Details: CIFAR-100 [Krizhevsky and others, 2009] is a labeled subset of the 80 million tiny images dataset. There are a total of 100 classes and 600 images of 32 × 32 per class. The first 70 classes are used for training, and the last 30 classes are used as a test set. We use the same split for learning trajectories for training and testing as used in the case of imagenet.

Hyperparameter Settings: We train our mode using hyperparameters: \( \beta_1 = 1e^{-4}, \alpha_1 = 1e^{-3}, \text{steps}_1 = 50000, \beta_2 = 1e^{-4}, \alpha_2 = 1e^{-3}, \gamma = 1e^{-4}, \text{steps}_2 = 35000, \beta_3 = 1e^{-4}, \alpha_3 = 0.1, \lambda = 5e^{-4}, \text{steps}_3 = 21000, \delta = 0.5. \)

Model Architecture: We use five convolutional layers followed by two fully connected layers with 256 convolutional filters of 3 × 3 kernel size with strides [1, 2, 1, 2, 1].

We use the first 70 classes as a training set and the last 30 classes as the test set. From Table 6, we observe an absolute improvement of 23.9% over the close competitor MRCL, when 10 classes are learned continually.

| Classes | Scratch | MRCL | Ours |
|---------|---------|------|------|
| 10      | 10.3 ± 1.3 | 20.0 ± 4.3 | 43.9 ± 6.0 |
| 15      | 7.0 ± 0.8  | 16.6 ± 2.5 | 37.5 ± 4.0 |
| 20      | 5.1 ± 0.3  | 13.8 ± 2.4 | 32.7 ± 2.8 |
| 25      | 4.2 ± 0.8  | 13.4 ± 1.7 | 29.3 ± 1.5 |
| 30      | 3.4 ± 0.1  | 11.6 ± 2.3 | 26.0 ± 3.1 |

Table 6: Results on CIFAR-100 dataset for the different number of classes incrementally learned in an online fashion.

10.2 Experiments on Regression Tasks

Implementation Details: We follow the same setting as [Javed and White, 2019]. We use sine functions for our experiments. We create functions by randomly sampling amplitude, phase from [0,1] and [0, \pi] respectively. 400 randomly sampled sine functions are used as the training set, and 500 functions are used as the testing set. A learning trajectory is created by randomly selecting ten functions from the train/test set for training/evaluation. We use 1280 samples for \( \tau_{tr} \) from each of the ten functions. We use 32 samples for \( \tau_{val} \) from each of the 10 functions. Each sample in the learning trajectory could be denoted by \((X = (n,z),Y)\) where \(Y = \sin_{steps}(z), n \in [1,10] \). We perform batch-online updates over the training data of the learning trajectory with batch size 32. We use Adam optimizer, and the first six layers are used for learning representations.

Hyperparameter Settings: We train our mode using hyperparameters: \( \beta_1 = 1e^{-5}, \alpha_1 = 1e^{-5}, \text{steps}_1 = 300000, \beta_2 = 1e^{-5}, \alpha_2 = 1e^{-5}, \gamma = 1e^{-4}, \text{steps}_2 = 75000, \beta_3 = 2.7e^{-6}, \alpha_3 = 3e^{-3}, \lambda = 5e^{-4}, \text{steps}_3 = 23500, \delta = 0.5. \)

Model Architecture: The regression task’s architecture has nine fully connected layers with 300 neurons per layer, which is the same as used in [Javed and White, 2019]. ReLu activation function is used for non-linearity.

We use sinusoidal functions as regression tasks. We sample 400 functions for the training and 500 functions for the evaluation. As we can observe from Table 7, our method consistently outperforms all the baseline methods.

Table 7: MSE loss on 10 regression tasks learned in an incremental fashion averaged over 50 runs. The tasks column refers to loss calculated after having seen the specified number of tasks.

| Tasks | Pretrained | SRNN | MRCL | OML | Ours |
|-------|------------|------|------|-----|------|
| 1     | 0.03       | 0.01 ± 0.003 | 0.11 ± 0.10 | 0.08 | 0.08 ± 0.06 |
| 2     | 0.42       | 0.32 ± 0.27  | 0.23 ± 0.16 | 0.23 | 0.12 ± 0.13 |
| 3     | 0.73       | 0.41 ± 0.35  | 0.37 ± 0.18 | 0.26 | 0.09 ± 0.05 |
| 4     | 0.87       | 0.50 ± 0.34  | 0.28 ± 0.13 | 0.22 | 0.16 ± 0.13 |
| 5     | 1.03       | 0.64 ± 0.33  | 0.28 ± 0.12 | 0.26 | 0.16 ± 0.10 |
| 6     | 1.04       | 0.64 ± 0.40  | 0.34 ± 0.17 | 0.28 | 0.17 ± 0.08 |
| 7     | 1.40       | 0.87 ± 0.37  | 0.33 ± 0.19 | 0.29 | 0.24 ± 0.13 |
| 8     | 1.22       | 0.76 ± 0.36  | 0.30 ± 0.16 | 0.27 | 0.19 ± 0.10 |
| 9     | 1.33       | 0.80 ± 0.37  | 0.33 ± 0.18 | 0.32 | 0.22 ± 0.10 |
| 10    | 1.45       | 0.90 ± 0.45  | 0.40 ± 0.23 | 0.35 | 0.22 ± 0.09 |

10.3 Generative Incremental Learning Experiments on MNIST Dataset

Implementation Details: We use MNIST [LeCun and Cortes, 2010] dataset for this experiment. We follow the same algorithm for the generative setting also. We use a conditional GAN for our experiment. The classification loss is replaced by the discriminative and generative loss. Classes 5, 6, 7 are used for testing, and the rest of the classes are used as the training set. For the learning trajectory during training, \( \tau_{tr} \) constitutes of 10 samples of a class randomly sampled from the train set, and \( \tau_{val} \) consists of 15 samples where 10 samples are randomly sampled from the train set, and the remaining five samples belong to the class used in \( \tau_{tr} \). The evaluation protocol for the generative setting is as follows: We learn from classes 5, 6, and 7 sequentially. While learning from a particular class, we do batch updates with batch size 64 for 25 epochs before learning the next class. We have demonstrated the model’s ability to prevent catastrophic forgetting by generating samples for class 5 after learning classes 5, 6, 7 in a continual fashion.

Hyperparameter Settings: We train our mode using hyperparameters: \( \beta_1 = 1e^{-5}, \alpha_1 = 1e^{-5}, \text{steps}_1 = 300000, \beta_2 = 1e^{-5}, \alpha_3 = 1e^{-5}, \lambda = 1e^{-4}, \text{steps}_3 = 3500000, \delta = 1.0. \)

Model Architecture: We use conditional-gan with Wasserstein loss [Arjovsky et al., 2017] for generative setup. The Generator and Discriminator architecture are given as follows.

Generator: Generator is a five-layered fully connected neural network of size \((110 \times 128, 128 \times 256, 256 \times 512, 512 \times 1024, 1024 \times 1024)\). For the non-linearity, Leaky ReLU is used with 0.2 as the negative slope value.

Discriminator: Discriminator is a four-layered fully connected neural network of size \((1034 \times 512, 512 \times 512, 512 \times 512, 512 \times 1)\). For the non-linearity, Leaky ReLU is used with 0.2 as the negative slope value.

Figure 2: Comparison of our model against MRCL on generative continual learning tasks. Classes 5, 6, 7 were learnt sequentially. As we can observe the 5’s generated by MRCL have been deformed to 7’s as they had been learnt recently, but our model is still able to retain the shape of the 5’s which was learnt at the earliest stages.
We use three classes (say 5, 6, 7) as the test set and the remaining classes are used as the training set. For evaluation, we learn from 1600 examples each of the classes in an online fashion. As we can observe from Figure 2, the quality of images generated by KCCIOL is much better than MRCL. The images generated by MRCL are highly distorted, but images generated by our model are slightly distorted, as it is able to retain the knowledge about previous classes. Therefore the KCCIOL does not outperform the discriminative model only but the generative model also.

Figure 3: Comparison of performance of KCCIOL and KCCIOL-o (without knowledge transfer) on continual learning tasks. The x axis indicates the number of classes learnt continually averaged over 50 test tasks on a) CIFAR-100 dataset, b) Mini-Imagenet dataset.

Figure 4: (a) Histogram of 5th convolution layer with and without $l_1$ fine-tuning on Mini-Imagenet dataset. We can observe from the flat tail of the distribution that the knowledge is being transferred to a subset of parameters. (b) Comparison of performance of KCCIOL as a function of masking level for the omniglot dataset.

10.4 Ablation Study and Analysis

Parameter Importance: We plot the accuracy as a function of the number of classes learned continually on the CIFAR-100 and Mini-Imagenet dataset as shown in Figure 3 for KCCIOL and KCCIOL-o (without knowledge transfer). In the case of KCCIOL, only the important parameters are constrained. Whereas in KCCIOL-o, all the parameters are constrained. To apply a constraint loss on the basis of importance, we first perform a knowledge transfer step to transfer all the knowledge to the important parameters. We can infer from the flat tail in Figure 4 (a) that the knowledge is indeed transferred to a subset of the model parameters. We can observe from Figure 3 that constraining only important parameters (KCCIOL) instead of constraining all the parameters (KCCIOL-o) boosts the performance significantly on continual learning tasks. Therefore, weight constraint on important parameters substantially improves performance over a different number of classes.

Masking level: Masking level is defined as the fraction of model parameters constrained during training. We plot the accuracy as a function of the masking level on the Omniglot dataset. We can observe from Figure 4 (b) that we get the best performance at 50% masking level, i.e., we apply weight constraint over 50 percent of the model parameters (important parameters) based on the absolute value of weights. If we choose the masking level below 50% then we may lose accuracy because we are not applying weight constraint over all important parameters. If we choose a masking level above 50% then we may lose accuracy because we have not rejuvenated all the unimportant parameters.

Analysis of Representation Learned by KCCIOL: Figure 5 depicts the activation maps of the feature representation generated by the models. As we can observe for a random image instance of meta-test of the omniglot dataset, the activation map of both OML and KCCIOL are sparse. The strength of average activation is much better in KCCIOL as compared to OML. This indicates that the model produced by OML is not being used to its full capacity. In KCCIOL, we enhance the model’s performance as shown in Figure 5 where, on average, a significant proportion of activations is strong for KCCIOL as compared to OML. Columns-3, 4 in Figure 5 show that the representation learned by our method (KCCIOL) is better than OML.