Rectification-based Knowledge Retention for Task Incremental Learning

Pratik Mazumder*, Pravendra Singh*, Piyush Rai, Vinay P. Namboodiri

Abstract—In the task incremental learning problem, deep learning models suffer from catastrophic forgetting of previously seen classes/tasks as they are trained on new classes/tasks. This problem becomes even harder when some of the test classes do not belong to the training class set, i.e., the task incremental generalized zero-shot learning problem. We propose a novel approach to address the task incremental learning problem for both the non zero-shot and zero-shot settings. Our proposed approach, called Rectification-based Knowledge Retention (RKR), applies weight rectifications and affine transformations for adapting the model to any task. During testing, our approach can use the task label information (task-aware) to quickly adapt the network to that task. We also extend our approach to make it task-agnostic so that it can work even when the task label information is not available during testing. Specifically, given a continuum of test data, our approach predicts the task and quickly adapts the network to the predicted task. We experimentally show that our proposed approach achieves state-of-the-art results on several benchmark datasets for both non zero-shot and zero-shot task incremental learning.

Index Terms—Task Incremental Learning, Continual Learning, Image Classification, Generalized Zero-Shot Classification, Deep Learning.

1 Introduction

Deep learning models have surpassed human performance for a number of tasks and are being increasingly used to solve real-world problems. However, such models still suffer from a shortcoming, i.e., they require the entire training data to be available when they start training the model. If the training data becomes available sequentially, then these models suffer from catastrophic forgetting [1] of the previous classes or tasks, and their performance degrades for these classes/tasks. This is in stark contrast to humans, who can incrementally learn new tasks or categories of data without forgetting the knowledge gained previously. This is known as the lifelong/continual/incremental learning problem [2]. Another requirement for deep learning models is that all categories of data that the model is supposed to work with should be present in the training data. Instead, if there are classes in the test set that were not present in the training data, the performance of deep learning models suffers from significant degradation [3]. Researchers have proposed zero-shot learning to solve this problem. If the incremental learning problem is coupled with the zero-shot learning problem, then the overall problem becomes even harder. In this work, we solve for the task incremental learning problem in the zero-shot and non zero-shot settings and demonstrate that our approach achieves state-of-the-art performance in these settings.

In the task incremental learning problem, training data becomes available in the form of one task at a time. Each task contains training examples from a set of classes, and we assume that the classes in each task do not overlap. Further, the data from all the previous tasks become inaccessible to the model when the training on a new task begins. These constraints provide challenges during training. If we apply the general training process to train the model only on the data from the new task, the model will forget the knowledge gained from the previous tasks. As a result, the objective of any approach in this setting is to prevent the catastrophic forgetting of knowledge gained from the previous tasks when training the model on the new task. Similarly, the task incremental generalized zero-shot learning problem also involves training the model on tasks that arrive sequentially. However, each task contains a set of seen classes and unseen classes (not part of the training data). In this case, the objective of the model is to retain its capability to classify the seen and unseen classes of all the previous tasks when training on a new task.

In this paper, we propose a novel approach called Rectification-based Knowledge Retention (RKR) for the task incremental learning problem in the non zero-shot and zero-shot settings. Our proposed RKR approach learns the rectifications needed to adapt the network to a new task. During testing, it can then quickly adapt the network to any given task, without any fine-tuning, by simply applying these weight rectifications to the weights of the network layers. RKR uses a parameter-efficient technique to learn these weight rectifications in order to limit the model size. Additionally, RKR also learns the affine transformations (scaling factors) needed to better adapt all the intermediate outputs of the network to the given task. In our previous work [4], we assumed that the model has access to the task label of the test sample during test time (task-aware),

* *The first two authors have contributed equally to this work.*

- *Part of this work has been accepted in preliminary form in the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-2021). Our CVPR paper "Rectification-based Knowledge Retention for Continual Learning" can be found on this link.*
- *Pratik Mazumder is with the Department of Computer Science and Engineering, IIT Jodhpur, Rajasthan, 342030, India. E-mail: pratikm@iitj.ac.in.*
- *Pravendra Singh is with the Department of Computer Science and Engineering, IIT Roorkee, Uttarakhand, 247667, India. E-mail: pravendra.singh@cs.iitr.ac.in.*
- *Piyush Rai is with the Department of Computer Science and Engineering, IIT Kanpur, Uttar Pradesh, 208016, India. E-mail: piyush@cse.iitk.ac.in.*
- *Vinay P. Namboodiri is with the Department of Computer Science, University of Bath, United Kingdom. E-mail: vpn22@bath.ac.uk.*

Manuscript received 30 April 2021.
(Corresponding author: Pravendra Singh.)

and we use this information to quickly adapt the network to the corresponding task by applying the weight rectifications and affine transformations of that task. Whereas in this work, we extend our approach to also support the test setting where the model does not have access to the task labels during testing (task-agnostic). We extend our approach to predict the task label from the output logits of the test images in a data continuum. It then quickly adapts the network to the predicted task and performs classification on that task. Therefore, our approach alleviates the need for task labels during the testing process for both the zero-shot and non zero-shot settings, making it task-agnostic [5]. RKR can quickly adapt to any task during testing because it only involves adding the corresponding weight rectifications to the network weights and applying the corresponding scaling factors to the intermediate network outputs.

We perform various experiments on multiple benchmark datasets for the task incremental learning problem in both the zero-shot and non zero-shot settings in order to show the effectiveness of our approach. We also perform various ablation experiments in order to validate the components of our approach. Our contributions can be summarized as follows:

- We propose a novel approach for the task incremental learning problem in the zero-shot and non zero-shot settings that learns weight rectifications and scaling factors in order to adapt the network to the respective tasks.
- Our proposed approach RKR introduces very few parameters during training for learning the weight rectifications and scaling factors. The model size growth in our method is significantly low as compared to other dynamic network-based task incremental learning methods.
- We experimentally show that our method Rectification-based Knowledge Retention (RKR) significantly outperforms the existing state-of-the-art methods for the task incremental learning problem in both the zero-shot and non zero-shot settings when the task label is available during testing (task-aware).
- Our approach works even when the task label information is missing during testing (task-agnostic). Our approach can automatically predict the task and classes of any test data continuum. Our approach achieves state-of-the-art performance for the task-agnostic non zero-shot incremental learning setting.
- We also show experimentally that for the task-agnostic generalized zero-shot incremental learning setting, our approach achieves state-of-the-art performance and significantly outperforms even the existing method that requires task labels during testing.

2 Related Works

2.1 Incremental Learning

Incremental learning [2] is a setting where we have to train the model on tasks that arrive incrementally. The model has to retain the knowledge gained from the older task while learning the new tasks [6], [7], [8]. We can categorize incremental learning methods into: replay-based, regularization-based, and dynamic network-based methods.

Replay-based methods store an exemplar set of data from the previously seen tasks and use this data along with the data from the new task to fine-tune the network such that the network performs well on the new task and the previous tasks. The authors in [6] propose to use an exemplar-based prototype rehearsal technique along with distillation. The methods proposed in [9], [10] use a custom architecture to produce pseudo samples for the older tasks to be used for rehearsal. Gradient of episodic memory (GEM) [11], promotes positive knowledge transfer to the older tasks while preventing catastrophic forgetting. The work in [5] proposes a task-agnostic meta-learning approach iTAML for incremental learning with a new meta-update rule to avoid catastrophic forgetting. During testing, iTAML predicts the task of the data continuum, and it stores an exemplar set to quickly adapt the network to the predicted task using meta-learning.

Another approach of preventing catastrophic forgetting of the older tasks/classes is to use regularization techniques to prevent any significant change in the network outputs while training on a new task. Such methods are known as regularization-based methods for incremental learning. The work in [12] proposes to use knowledge distillation as the regularization technique. In [6], [13], the authors propose to use modified classification techniques suited to continual learning in addition to the distillation loss. Deep model consolidation (DMC) [14] combines separate networks trained on disjoint classes using a distillation process. Memory Aware Synapses (MAS) [15] reduces catastrophic forgetting of older tasks by preventing important network parameters from changing significantly. Elastic Weight Consolidation (EWC) [16] selectively slows down the learning of some of the network weights that are vital to preserving the knowledge from the previous tasks. The authors in [17] propose Synaptic Intelligence (SI) that uses intelligent synapses to accumulate task-relevant information and utilizes this information to learn new tasks rapidly without forgetting the knowledge gained from the previous tasks. The authors in [18] propose an improved version of the EWC method called EWC++ and also propose the Riemannian Walk (RWalk) method, which is a generalized version of EWC++, and Path Integral [17]. P&C [19] trains a knowledge base to accumulate previous task knowledge and an active column to learn the new task. After learning the new task, the knowledge gained by the active column is merged into the knowledge base using distillation.

Dynamic network-based methods use network expansions/modifications for training new tasks. The work in [20] proposes to create an extra network for each new task with lateral connections to the networks of the older tasks. The method proposed in [21] uses reinforcement learning to determine how many neurons to add for each new task. DEN [22] performs selective retraining and dynamically expands the network for each task with only the required amount of units. The method proposed in [23] uses a random path selection methodology for each task. The authors in [24] propose an order-robust approach APD, which uses task-shared and task-adaptive parameters. Deep Adaptation Module (DAM) [25] learns filters for the new task as a linear combination of existing filters. Recently the authors in [26] proposed CCLL that calibrates the feature maps of convolutional layer outputs to perform incremental learning. The superposition method proposed in [27] learns and stores layer weights for different tasks within a single set of parameters using superposition through context parameters. Therefore, when the weights are retrieved for a given task, the retrieved weights are likely to be noisy estimates of the actual task-specific layer weights (as also mentioned in [27]). Due to this issue, as the model in the superposition approach is trained on more tasks, the problem

of catastrophic forgetting gradually increases for the older tasks (which can be observed in Fig. 6 (b) in [27]). This is in contrast to our approach, where the base network remains frozen after being trained on the first task, and we only learn the weight rectifications and scaling factors for each new task. As a result, in our approach, we can retrieve the actual task-specific layer weights for adapting the network to the given task. Further, the authors in [27] specify that when the task label is not known, the number of models to be stored increases 100 times in the superposition method, which is not the case in RKR. We empirically show that RKR significantly outperforms this approach (see Fig. 3). Therefore, our proposed RKR is a more parameter-efficient and effective approach than [27]. The work in [28] iteratively finetunes and prunes the network parameters for incremental tasks. However, its scalability is limited by the network size, and we have empirically shown that our proposed approach significantly outperforms this approach (see Fig. 3).

Our method RKR follows the dynamic network-based approach, but it is the first work that learns rectifications for the layer weights and outputs to adapt the model to any task. Even though our method is dynamic network-based, it does not use the parameter isolation approach, which incrementally reserves a set of model parameters for new tasks. Therefore, our model will not run out of model capacity to accommodate future tasks. Our method introduces a significantly less number of parameters to learn the weight rectifications and scaling factors, e.g., for CIFAR-100 tasks using the ResNet-18 architecture, RKR introduces only 0.5% additional parameters per task. When the task label is available during testing (task-aware), RKR outperforms the state-of-the-art methods in this setting (see Fig. 3, Table 5). We also extend our approach to work when the task labels are not available during test time (task-agnostic) and empirically show that our approach significantly outperforms the existing methods in this setting (see Fig. 6, Table 5). Unlike iTAML, our approach also does not need to fine-tune the network during testing to adapt it to the predicted task.

2.2 Zero-Shot Learning

In the zero-shot learning (ZSL) problem, the objective of the network is to identify categories that are not part of the training data. Seen classes are those classes that are part of the training data. In contrast, unseen classes are the classes that are not seen during training. The generalized zero-shot learning problem is a harder and more practical setting, in which the test set contains samples from both the seen and unseen classes. Zero-shot learning methods utilize side information in the form of class embeddings/attributes that encode the semantic relationship between classes. The most popular approach for zero-shot learning is to learn an embedding space where the image data and the class embedding are close to each other [29]. Another popular approach is to generate images/features of unseen classes by using their class embeddings [30], [31]. The authors in [32] propose f-CLSWGAN, which uses conditional Wasserstein GANs, to generate features for unseen classes. Cycle-WGAN [30] improves upon f-CLSWGAN by using reconstruction regularization in order to preserve the discriminative features of classes.

CADA-VAE [33] uses variational auto encoders to model visual features and class attribute embeddings. It then uses the learned latent embeddings to train a zero-shot classifier. Recently, the authors in [34] proposed an approach LZSL for the task incremental generalized zero-shot learning problem. The LZSL approach performs selective parameter retraining and knowledge distillation to preserve old domain knowledge and prevent catastrophic forgetting in the image feature encoder. We extend our proposed method RKR to the task incremental generalized zero-shot learning setting [34]. In this setting, RKR "rectifies" the weights and outputs of the image features encoder network in order to perform continual learning and prevent catastrophic forgetting. Our approach works in this setting irrespective of whether the model has access to task/dataset labels (task-aware) or not (task-agnostic) during testing. We compare our approach with LZSL for both the task-aware and task-agnostic generalized zero-shot incremental learning settings. We also empirically show that our task-agnostic RKR even outperforms the task-aware LZSL method (see Table 10).

3 Problem Definition

3.1 Task Incremental Learning

In the task incremental learning setting, the network receives a sequence of tasks containing new sets of classes. When a new task becomes available, the previous task data are not accessible. The objective of task incremental learning is to obtain a model that performs well on the current task as well as the previous tasks.

3.2 Task Incremental Generalized Zero-Shot Learning

The task incremental generalized zero-shot learning setting also involves training the model on a sequence of tasks, but each task contains a set of seen and unseen classes, and the final model should perform well on the seen and unseen classes of the current task as well as the previous tasks. For this problem, we follow the setting defined in [34], where each task is a separate dataset. When a new task becomes available for training, the older tasks are no longer accessible for further training/fine-tuning.

4 Proposed Method

4.1 Rectification-based Knowledge Retention

We propose a task incremental learning approach called Rectification-based Knowledge Retention (RKR) that applies network weight rectifications and scaling transformations to adapt the network to different tasks.

Let us assume that we have a deep neural network with N layers, i.e., $\{L_1, L_2..., L_N\}$. Each layer can be a convolutional layer or a fully connected layer. Let Θ_l represent the parameter weights of layer L_l. If we train this network on a task containing a set of classes, the network will learn the parameter weights Θ_l for each layer $l \in \{1, 2, .., N\}$. However, if we then train the network on a new task (with a new set of classes), it will learn new parameter weights Θ_l^* to work for this task and will lose information regarding the previous tasks (catastrophic forgetting).

We propose to avoid this problem using the dynamic network-based approach. For each task, we learn the rectifications needed to adapt the layer weights of the network to work for that task. Let R_l^t refer to the weight rectification needed to adapt the l^{th} layer of the network to work for task t. We use a rectification generator (RG) for learning these rectifications. RG uses very few parameters to learn the weight rectifications as described in Sec. 4.2. The weight rectifications R_l^t are added to the layer weights Θ_l for each task t (Figs. 1, 2).

$$\Theta_l^t = \Theta_l \oplus R_l^t \tag{1}$$

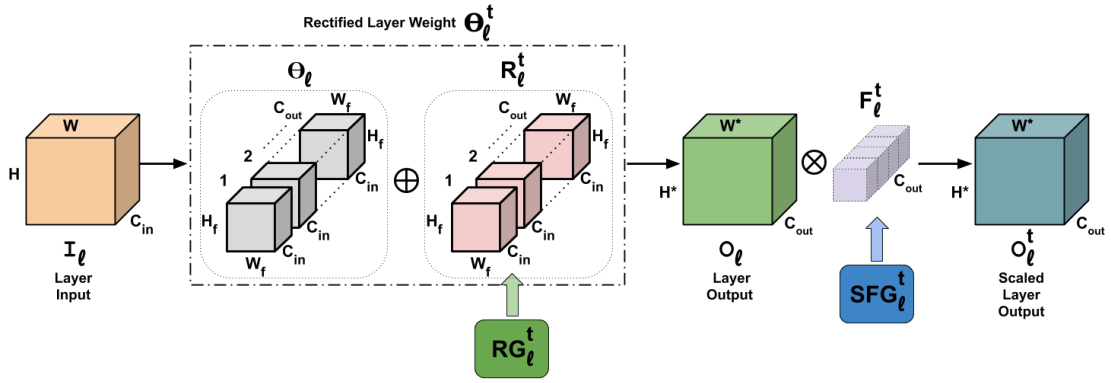


Fig. 1: RKR for a convolutional layer. The weight rectifications R_l^t produced by the rectification generator (RG_l^t) are added to the layer weights (Θ_l) of the convolutional layer l for task t. The task-adapted convolution layer weights (Θ_l^t) are applied to the input I_l to produce the layer output O_l. The scaling factor generator (SFG_l^t) produces scaling factors F_l^t that are applied to O_l to produce the scaled layer output (O_l^t).

where Θ_l refers to weights of layer l of the network, R_l^t refers to the rectifications to be learned for the weights of the layer l for task t, Θ_l^t refers to the rectified weights of the layer l for task t, $\oplus$ refers to element-wise addition. The layer weight Θ_l is trained only on the first task and is adapted using the weight rectifications R_l^t (that are learned for all tasks) to obtain Θ_l^t.

Apart from the weight rectifications, we also learn scaling factors to perform affine transformations on the intermediate outputs generated by each layer of the network. We use a scaling factor generator (SFG) to learn the scaling factors. In the case of a fully connected layer l, the scaling factors F_l^t have the same size as the layer output O_l, and we multiply them element-wise to each component of O_l (Fig. 2). In the case of a convolutional layer l, the scaling factors F_l^t have the same number of elements as the number of feature maps in O_l, and we multiply them to the corresponding feature maps of O_l (Fig. 1). These learned scaling factors represent the rectifications needed to adapt the intermediate network outputs to the corresponding task.

$$O_l^t = O_l \otimes F_l^t \tag{2}$$

where O_l refers to the output from the layer l of the network, F_l^t refers to the scaling factors learned for the output of the layer l of the network for task t, $\otimes$ refers to the scaling operation, and O_l^t denotes the scaled layer output for task t. Our approach applies the weight rectifications and scaling factors to adapt the network for any task t.

4.2 Reducing Parameters for Weight Rectification

4.2.1 Convolutional Layers

The weight rectifications for a convolutional layer is required to be of the same size as the convolutional layer weights. Let W_f,H_f,C_{in} be the width, height and number of channels of each filter of the convolutional layer L_l and C_{out} be the number of filters used in L_l. Therefore, the total size of the convolutional layer weights is $W_f \times H_f \times C_{in} \times C_{out}$. The weight rectification R_l^t for layer L_l has to be of the same size. In order to reduce the number of parameters needed to generate these weight rectifications, we use a rectification generator (RG). The rectification generator (RG) learns two matrices of smaller size i.e., LM_l^t of size $(W_f * C_{in}) \times K$ and RM_l^t of size $K \times (H_f * C_{out})$, where $*$ represents scalar multiplication. Here, $K \ll (W_f * C_{in})$ and $K \ll (H_f * C_{out})$. This process ensures that we introduce very few parameters to generate these weight rectifications. The product of these two matrices produces the weight rectifications which are reshaped to the size $W_f \times H_f \times C_{in} \times C_{out}$ and added to the convolutional layer weights element-wise. Therefore, RG computes the weight rectifications R_l^t for task t as:

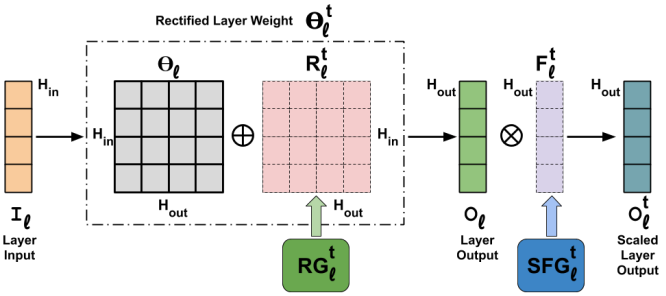


Fig. 2: RKR for a fully connected layer. The weight rectifications R_l^t produced by the rectification generator (RG_l^t) are added to the layer weights (Θ_l) of the fully connected layer l for task t. The task-adapted fully connected layer weights (Θ_l^t) are applied to the input I_l to produce the layer output O_l. The scaling factor generator (SFG_l^t) produces scaling factors F_l^t that are applied to O_l to produce the scaled layer output (O_l^t).

$$R_l^t = \texttt{MATMUL}(LM_l^t, RM_l^t) \tag{3}$$

where MATMUL refers to matrix multiplication.

We apply the adapted convolution layer weights (Θ_l^t) to the input (I_l) of size, say, $W \times H \times C_{in}$, to obtain an output of size $W' \times H' \times C_{out}$ (See Fig. 1). Here, W, W' refer to the width of the feature maps before and after applying the convolution. H, H' refer to the height of the feature maps before and after applying the convolution. We then apply scaling transformation to the output O_l. The scaling factor generator (SFG) learns a scaling parameter for each feature map. Therefore, SFG introduces a very insignificant number of parameters, i.e., C_{out}, which is equal to the number of feature maps in O_l.

$$EP_{conv} = \frac{K * (W_f * C_{in} + H_f * C_{out}) + C_{out}}{W_f * H_f * C_{in} * C_{out}} * 100 \tag{4}$$

where EP_{conv} refers to the percentage of extra parameters introduced by our approach for each convolutional layer.

4.2.2 Fully Connected Layers

The weight rectifications R_l^t for the fully connected layers require to be of the same size as the layer weights Θ_l. Let Θ_l be of size $H_{in} \times H_{out}$. Here H_{in} and H_{out} refer to the size of input and output of the fully connected layer, respectively. In order to reduce the number of parameters needed to generate the weight rectifications, the rectification generator (RG) learns two matrices of smaller size, i.e., LM_l^t of size $H_{in} \times K$ and RM_l^t of size $K \times H_{out}$. Here, $K < H_{in}$ and $K < H_{out}$. Therefore, the total parameters added will not be significant. The product of these two matrices will give the weight rectifications of size $H_{in} \times H_{out}$ which we add to the layer weights Θ_l element-wise to produce the adapted layer weight Θ_l^t. Therefore, RG computes the weight rectifications R_l^t for task t as follows:

$$R_l^t = \texttt{MATMUL}(LM_l^t, RM_l^t) \tag{5}$$

where MATMUL refers to matrix multiplication.

We apply the adapted fully connected layer weights (Θ_l^t) to the input (I_l) of size H_{in}, to obtain an output (O_l) of size H_{out} (See Fig. 2). We then apply scaling transformation to the output O_l. The scaling factor generator (SFG) learns a parameter for each component of O_l. Therefore, SFG introduces a very insignificant number of parameters, i.e., H_{out}.

$$EP_{fc} = \frac{K * (H_{in} + H_{out}) + H_{out}}{H_{in} * H_{out}} * 100 \tag{6}$$

where, EP_{fc} refers to the percentage of extra parameters introduced by our approach for each fully connected layer.

Therefore, our approach introduces very few parameters per task to learn weight rectifications and scaling factors in the incremental learning setting, e.g., for the ResNet-18 architecture RKR introduces only 0.5% additional parameters per ImageNet-1K task. Intuitively, this simulates separate networks for each task using very few parameters, e.g., our model for ImageNet-1K with 10 tasks has $(100 + 10 * 0.5)\%$ capacity. However, directly using separate networks will lead to an impractical model with $(100 * 10)\%$ capacity.

4.3 Testing Process with Access to Task Labels

In the task-aware test setting, the model has access to task labels during testing [22], [24], [26], and each test sample consists of the test image (x_i), the actual class label (y_i), and the task label (t_i). Given the trained deep neural network with layers $L_1, L_2, ..., L_N$, we use the weight rectifications, and scaling factors learned for each network layer for task t_i to quickly adapt the network. Specifically, we add the weight rectifications to the network layer weights using Eq. 1 and scale the intermediate outputs of the network using Eq. 2 to adapt the network to the given task. Finally, we perform classification on the test example using this adapted network to predict its class label.

4.4 Testing Process without Access to Task Labels

In the task-agnostic test setting, the model does not have access to task labels during testing and the test data is received in the form of a continuum with an unknown task t (similar to [5]). Let $DC = \{(x_i, y_i, t_i)\}_{i=1}^{N_{DC}}$ refers to a data continuum with N_{DC} test examples from the same task t, such that $t_i = t, \forall i \in \{1, 2, ..., N_{DC}\}$. However, the task label (t) is not known to the model.

Our proposed approach requires the task label to adapt the network to the given task. Therefore, in the task-agnostic test setting, we have to identify the task of the data continuum before carrying out the testing process. Let us consider a simple approach for identifying the task label of the data continuum. First, we adapt the network to any task t by applying the weight rectifications and scaling factors for that task. Next, we obtain the output logits for each example in DC for task t and store the maximum logit value for each example in DC for that task. Next, we compute the average maximum logit (AML) by taking the mean of the maximum logit values for all examples in DC for task t. We repeat this process for all the tasks. Therefore, we have the average maximum logit for each task for this data continuum. Ideally, the task for which the average maximum logit of the data continuum is the highest should be the correct task of the data continuum. However, in our approach, the model has not been jointly trained on all the tasks. Therefore, given a test data continuum from a particular task t_a, a model adapted to another task t_b ($a \neq b$) has never seen such type of data. However, neural networks can still produce high logits even for images of classes that the network has never been trained on ([35], [36], [37]). As a result, the above process may lead to high average maximum logit even for the incorrect tasks, i.e., AML for t_b may be higher than AML for t_a in the above example. Therefore, this is a problem that needs to be addressed. A naive solution can be to train the model jointly on the complete data from all the tasks. However, this is not possible in the incremental learning setting. Therefore, we propose a novel approach to address this issue. Specifically, we propose to jointly learn a single hyper-parameter per task that will be used to re-weight the average maximum logit of each task for the given data continuum (see Eqs. 7, 8). We refer to these hyper-parameters as the task weight (TW) hyper-parameters.

$$AML_{t_i} = Avg(Max(M_{t_i}(DC))) \tag{7}$$

$$t_p = arg\ max_{t_i}(TW_{t_1} * AML_{t_1}, TW_{t_2} * AML_{t_2}, \cdots) \tag{8}$$

Where, AML_{t_i} refers to the AML of DC for task t_i. M_{t_i} refers to the model adapted to task t_i. $M_{t_i}(DC)$ refers to the M_{t_i} model being applied to each example in DC. Max refers to the function that that finds the maximum logit for each example in DC. Avg refers to the function that computes the average of the maximum logits of the examples in DC. TW_{t_i} refer to the TW hyper-parameter for task t_i. $TW_{t_i} * AML_{t_i}$ refers to the re-weighting of the AML for task t_i using the TW hyper-parameter for task t_i. $arg\ max_{t_i}$ refers to the function that finds the task for which the re-weighted AML is maximum. t_p refers to the predicted task.

In order to find the suitable values of the TW hyper-parameters, we use the validation data. The validation data consists of only one data continuum from each task. We perform a task prediction operation using the validation data and find the most suitable TW hyper-parameters for this operation. Please note that one validation data continuum will have a single predicted task. Therefore, one validation data continuum corresponds to a single data point for task prediction. Consequently, for t tasks, we will have only t data points in the validation set. The TW hyper-parameter values obtained using these few data points will not be generic enough to properly work for any given test data continuum. Therefore, we create multiple corrupted data points

from each validation data continuum by randomly replacing a few samples of a validation data continuum of a particular task with samples from the validation data continuum of the remaining tasks. Since we replace very few samples in the validation data continuum, the majority of the samples in the corrupted data continuum still belong to the original task, and therefore, we assume that the task label of the new corrupted data continuum is the same as the original validation data continuum. This process introduces a type of noise/corruption in each validation data continuum and helps the identified TW hyper-parameters to be more generic. For each new corrupted validation data continuum, we adapt the network to each seen task t one by one and obtain the AML for each task as described earlier. Next, we multiply the AML of each task with the TW hyper-parameter of the corresponding task. Finally, we compare the weighted AML of each task to obtain the task with the highest weighted AML for the corrupted validation data continuum. We use a cross-entropy loss on the predicted and actual task of the validation data continuum to optimize the TW hyper-parameters for very few iterations (2-5 iterations requiring a total of about 2-3 seconds on a GeForce GTX 1080 Ti graphics processing unit). Finally, during testing, the optimal TW hyper-parameters are used to re-weight the task-wise AML for each test data continuum before comparing the task-wise AML.

We also improve our approach for task prediction further by using data augmentation approaches during testing. We use different data augmentation approaches to create new views of a test data continuum. We use the same data augmentation approaches for this process that are used for training the model. This is because, in the case of an augmented view of the data continuum, the model adapted to the correct task has already seen similar data with this augmentation, and therefore, its AML will still remain high. However, a model adapted to any incorrect task has not seen similar data with this augmentation, and therefore, there is a low possibility that the same wrong task will still have a high AML for different augmented views of that data continuum. As a result, when we consider the task-wise mean of the AML across all the differently augmented views of the same test data continuum, there will be a higher possibility that the correct task will have the highest mean AML. We validate this finding for a data continuum in Table 9. This observation can be employed to further improve the task prediction accuracy of our approach.

Based on the above observation, we propose our final testing procedure for the task-agnostic setting as follows. First, we create multiple views of the given test data continuum DC by applying different data augmentation approaches. Next, for each augmented view of the data continuum, we obtain the task-wise average maximum logit using the process mentioned earlier. We re-weight the task-wise AMLs using the corresponding TWs obtained using the process described earlier. Finally, we take an average of the re-weighted AML per task across all the differently augmented views of the same test data continuum and predict the task with the highest mean re-weighted AML as the task of the given test data continuum (see Eqs. 9, 10, 11). After obtaining the predicted task t_p, we quickly adapt the network to the corresponding task by adding the weight rectifications to the network layer weights using Eq. 1 and scaling the intermediate outputs of the network using Eq. 2. Our approach does not require any fine-tuning or updates in this process as opposed to iTAML [5]. Finally, we use this adapted network to predict the class labels of the examples in the data continuum. For a fair comparison, we use the same data continuum size as used by [5].

$$AML_{t_i}^{v_j} = Avg(Max(M_{t_i}(DC^{v_j}))) \tag{9}$$

$$AML_{t_i}^{r} = Avg(TW_{t_i} * AML_{t_i}^{v_1}, TW_{t_i} * AML_{t_i}^{v_2}, \cdots) \tag{10}$$

$$t_p^* = arg\ max_{t_i}(AML_{t_1}^{r}, AML_{t_2}^{r}, \cdots) \tag{11}$$

Where, DC^{v_j} refers to an augmented view of DC using data augmentation v_j. $AML_{t_i}^{v_j}$ refers to the AML of DC^{v_j} for task t_i. $Avg(TW_{t_i} * AML_{t_i}^{v_1}, TW_{t_i} * AML_{t_i}^{v_2}, \cdots)$ refers to computing the mean of the re-weighted AMLs for different augmented views of DC for task t_i. $AML_{t_i}^{r}$ refers to the mean re-weighted AML of DC for task t_i. t_p^* refers to the predicted task.

We empirically validate all the design choices in our approach in Sec. 5.7.

5 Task Incremental Learning Experiments (Non Zero-shot Setting)

5.1 Datasets

We perform the task-aware incremental learning experiments on the CIFAR [38], SVHN [39], ImageNet-100 and ImageNet-1K [40] datasets for the non zero-shot setting. We perform the non zero-shot task-aware incremental learning experiments on CIFAR-100 with 10 classes per task (10 tasks). For the split CIFAR-10/100 experiments, we use all the classes of CIFAR-10 for the first task and randomly choose 5 tasks of 10 classes each from CIFAR-100. So we have 6 tasks for this setting. In the case of ImageNet-1K, we group the 1000 classes into 10 tasks of 100 classes each. In the case of ImageNet-100, we group the 100 classes into 10 tasks of 10 classes each. For the task-agnostic experimental setting where the model does not have access to task labels at test time, we perform experiments on the SVHN, CIFAR-100, ImageNet-100, and ImageNet-1K datasets. We perform experiments in this setting on the CIFAR-100 with 5, 10, and 20 classes per task resulting in 20, 10, and 5 tasks, respectively. In the case of SVHN, we have 5 tasks with 2 classes each. The number of tasks in ImageNet-100 and ImageNet-1K are 10 as in the previous setting.

5.2 Implementation Details

In our approach, we learn weight rectifications and scaling factors for each convolutional layer and fully connected layer of the network (except the classification layer). We train the complete network on the first task (base network). For every new task, we only learn weight rectifications and scaling factors for all network layers to adapt them to the new task.

For the CIFAR-100 experiments, we use the ResNet-18 architecture [41]. For the split CIFAR-10/100 experiments, we use the ResNet-32 architecture [41]. For the task-agnostic CIFAR-100 experiments, we also perform experiments using the ResNet-18/3 architecture for a fair comparison with [5]. ResNet-18/3 is basically the ResNet-18 architecture, but the filters in each layer are reduced by three times. In the above experiments, we train the network for 150 epochs for each task with the initial learning rate as 0.01, and we multiply the learning rate by 0.1 at the 50, 100, and 125 epochs. We also perform experiments with the LeNet architecture [42] on the CIFAR-100 tasks with 10 classes per task. We train the network on each task for 100 epochs with the initial learning rate as 0.01 and multiply the learning rate with

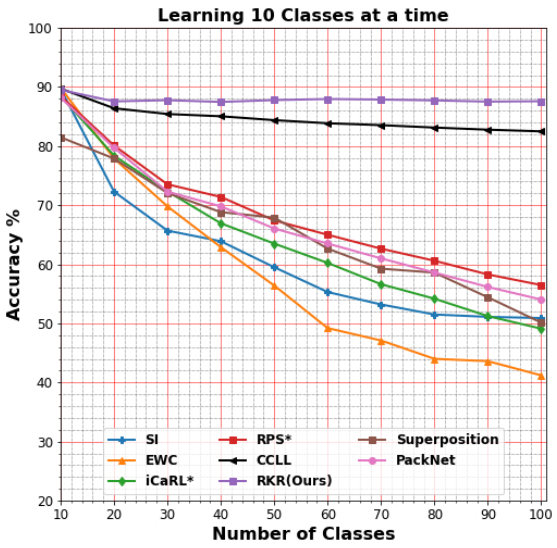


Fig. 3: Task incremental experimental results (task-aware) for the CIFAR-100 dataset with 10 classes per task using ResNet-18. '$*$' denotes replay-based approach.

0.5 at the 20, 40, 60, and 80 epochs. For the SVHN dataset, we perform experiments with the ResNet-18 architecture. For a fair comparison with [5], we also use the ResNet-18/3 architecture for the task-agnostic SVHN experiments. We train the network on each task for 150 epochs with the initial learning rate as 0.01 and multiply the learning rate with 0.1 at the 50, 100, and 125 epochs. For the ImageNet-100 and the ImageNet-1K experiments, we use the ResNet-18 architecture and train the network for 70 epochs for each task with the initial learning rate as 0.01, and we multiply the learning rate by 0.2 at the 20, 40, and 60 epochs. We use the SGD optimizer in all our experiments. We perform experiments with RKR using $K = 2$ since this is a good choice considering the accuracy/extra-parameters trade-off as shown in Table 2.

Our method can utilize task labels during testing (task-aware) similar to [22], [24], [26] and can also perform well without using task labels during testing (task-agnostic). For the task-agnostic testing, our method is evaluated on a data continuum similar to [5]. For the CIFAR-100 experiments, the data continuum size is 20 for the 5, 10, 20 task experiments. For the SVHN experiments, the data continuum size is 50. For the ImageNet-100 and ImageNet-1K experiments, the data continuum size is 50 and 100, respectively. We use the same data continuum size as [5] for a fair comparison. In the procedure for identifying the suitable values for the TW hyper-parameters, we create multiple corrupted versions of each validation data continuum by replacing 20% examples with examples from the validation data continuum of the remaining tasks. We validate this choice in Table 8. We run the process for identifying the TW hyper-parameters for only 2 iterations for the CIFAR-100, and SVHN datasets and for 5 iterations for the ImageNet dataset using a learning rate of 1e1. We run all the experiments for 5 randomly chosen task orders, each with a different first task, and report the average accuracy.

5.3 Task-Aware CIFAR Results

For the task-aware incremental learning experiments on the CIFAR-100 dataset with 10 classes per task, we perform experiments with various methods such as CCLL [26], SI [17], EWC [16], iCARL [6], RPS [23], Superposition [27] and PackNet [28].

Methods	Capacity	Accuracy
L2T [24]	100%	48.73%
EWC [16]	100%	53.72%
P&C [19] ICML'18	100%	53.54%
PGN [20]	171%	54.90%
RCL [21] NIPS'18	181%	55.26%
DEN [22] ICLR'18	181%	57.38%
APD [24] ICLR'20	135%	60.74%
CCLL [26] NIPS'20	100.7%	63.71%
RKR-Lite (Ours)	100.7%	**66.32%**
RKR (Ours)	104.3%	**69.58%**

TABLE 1: Task incremental experimental results (task-aware) for the CIFAR-100 dataset with 10 classes per task using LeNet.

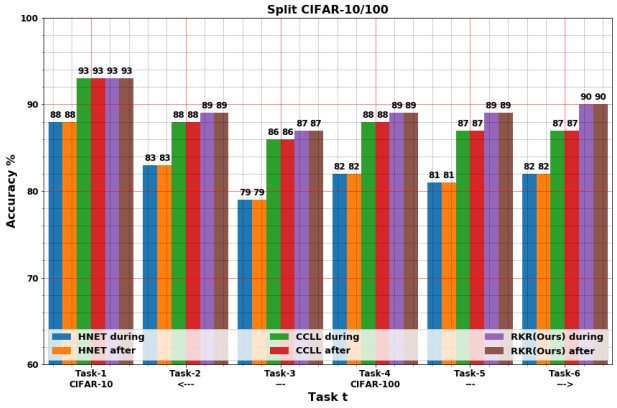


Fig. 4: Task incremental experimental results (task-aware) on split CIFAR-10/100 using ResNet-32 to check for catastrophic forgetting. We report the accuracy achieved for each task when the network is trained on that task (marked as during) and after the network has been trained on all the tasks (marked as after).

CCLL uses task labels at test time, and we modify SI and EWC to use task labels during testing for a fair comparison. We observe in Fig. 3 that our approach RKR outperforms all existing methods in this setting. RKR outperforms CCLL [26] by an absolute margin of 5.1% in the overall accuracy. Our approach performs consistently better than all other methods as more tasks arrive.

RKR applies the weight corrections to adapt the network to the new task, which is very natural and intuitive because training on a new task changes the network layer weights and, consequently, the corresponding features. In contrast, CCLL [26] adapts the network to the new task by only calibrating the convolutional layer output feature maps that are biased to the initial task. This is the reason why we see a significant performance gap between RKR and CCLL. This problem becomes even more apparent if the new task is very different from the initial task. In such a case, the features extracted by the model trained on the initial task will not be relevant to the new task, and it will be very difficult to calibrate the feature maps to correctly estimate the feature maps of the new task. For example, on taking MNIST images in the initial task and taking 10 tasks of CIFAR-100 as the subsequent tasks, the performance gap between RKR and CCLL increases from 5.1% to 16% absolute margin.

We also perform the task-aware non zero-shot incremental learning experiments on the CIFAR-100 dataset with 10 tasks using the LeNet architecture (20-50-800-500) as used in [24]. All the methods compared in Table 1 use task labels during testing (task-aware). The results in Table 1 indicate that RKR outperforms existing state-of-the-art methods. We also report the results for RKR-Lite, which uses only weight rectifications for the convolutional layers and only scaling factors for the fully connected

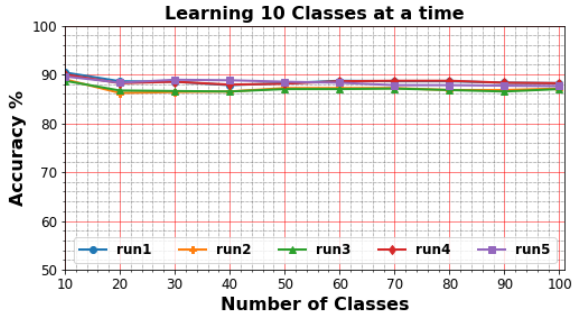


Fig. 5: Results for 5 runs of the task incremental learning experiments (task-aware) on the CIFAR-100 dataset with 10 tasks using ResNet-18 with RKR.

K	% Params. ↑	% FLOPs ↑	Accuracy
1	0.3355%	$8.6 \times 1e{-}4\%$	85.55%
2	0.5426%	$8.6 \times 1e{-}4\%$	87.60%
4	0.9569%	$8.6 \times 1e{-}4\%$	87.90%
8	1.7854%	$8.6 \times 1e{-}4\%$	88.49%

TABLE 2: Task incremental experimental results (task-aware) for the CIFAR-100 dataset with 10 tasks using ResNet-18 with RKR for different K values. We report the average accuracy of 10 tasks.

layers. RKR-Lite introduces the same number of parameters as CCLL but outperforms it by an absolute margin of 2.61%.

Split CIFAR-10/100: For the split CIFAR-10/100 tasks, we use the ResNet-32 architecture and compare our method RKR with CCLL, and HNET [43], which are the state-of-the-art methods for this setup and use task labels during testing. We observe in Fig. 4 that RKR not only prevents catastrophic forgetting just like CCLL and HNET but also outperforms these methods. Therefore, RKR helps in avoiding catastrophic forgetting without significantly affecting the network's ability to learn each task properly.

Different First Task: As mentioned in Sec. 4.1, Θ_l is trained only on the first task. Therefore, we perform experiments with different first tasks. From Fig. 5, we observe that the performance of RKR is stable for different first tasks with different task orders.

Value of K: We perform experiments on the CIFAR-100 dataset using ResNet-18 with RKR for different values of K. The results in Table 2 indicate that $K = 2$ is a good choice while considering extra-parameters/accuracy trade-off. Therefore, we use $K = 2$ for all our experiments. We also observe that the FLOPs increase due to RKR is insignificant.

Significance of Components: We observe in Table 3 that without weight corrections the model performs lower by absolute margins of 7.3% and 8.4% for CIFAR-100 using LeNet and ResNet-18, respectively. Without scaling, the model performance suffers slightly. This is because LeNet and ResNet-18 contain primarily convolutional layers and the scaling factors are more effective for fully connected layers. In the convolutional layers, we learn a single scaling factor for an entire feature map (one channel of the convolutional layer output). Whereas, in the fully connected layer, we learn a scaling factor for each component of the fully connected layer output. Consequently, the scaling factors learned for the fully connected layers are more effective at properly adapting it to the given task. We observe in task incremental generalized zero-shot learning that scaling helps to improve the RKR performance (Sec. 6.4). This is because the visual feature encoder under consideration in that setting contains only fully connected layers.

Base Network Training: We perform experiments to study

Arch.	Wt. Rect.	Scaling	% Params. ↑	% FLOPs ↑	Acc.
LeNet	✗	✓	0.0497%	$6.4 \times 1e{-}4\%$	62.3%
	✓	✗	0.3795%	0.0%	69.1%
	✓	✓	0.4292%	$6.4 \times 1e{-}4\%$	69.6%
Res-18	✗	✓	0.1283%	$8.6 \times 1e{-}4\%$	79.2%
	✓	✗	0.4143%	0.0%	87.5%
	✓	✓	0.5426%	$8.6 \times 1e{-}4\%$	87.6%

TABLE 3: Task incremental experimental results (task-aware) for the CIFAR-100 dataset with 10 tasks using LeNet, and ResNet-18 with different components of RKR ($K = 2$). ✓ and ✗ refer to presence and absence of the components, respectively.

Base Network	Acc.
Random Init.	79.21%
Trained on the First Task $t = 1$	87.60%

TABLE 4: Task incremental experimental results (task-aware) for the CIFAR-100 dataset with 10 tasks using ResNet-18 for the RKR ($K = 2$) model in which the base network is either randomly initialized before being frozen (Random Init.) or trained on the first task ($t = 1$) before being frozen.

the effect of not training the base network on the first task and only learning the weight rectifications and scaling factors for all the tasks. The results in Table 4 indicate that when the base network is not trained and is only randomly initialized before being frozen, the model performance falls drastically. This is because RKR introduces very few rectification parameters per task, and these few parameters are not sufficient to properly learn from the training data of the task.

Forward Knowledge Transfer: In our proposed approach, when we train the model on a new task, we initialize the parameters of RG and SFG from the previous task. If we train these parameters from scratch for every new task, the model performance falls by an absolute margin of 1.45% for CIFAR-100 using ResNet-18. This demonstrates the forward transfer of knowledge in RKR.

5.4 Task-Aware ImageNet Results

We observe in Table 5, that RKR significantly outperforms the state-of-the-art CCLL method for both the ImageNet-100 and ImageNet-1K datasets. Specifically, our method outperforms CCLL by absolute margins of 0.8% and 3.1% (top-5 accuracy) for the ImageNet-100 and ImageNet-1K datasets, respectively, even though both RKR and CCLL introduce around 0.5% extra parameters per task. It should also be noted that CCLL introduces 0.98% extra FLOPs in the model, whereas RKR introduces only $2.8 \times 1e{-}4\%$ extra FLOPs, which is very insignificant.

5.5 Task-Aware SVHN Results

We report the results for the task-aware non zero-shot incremental learning experiments for the SVHN dataset in Table 6. The reported result is the average class prediction accuracy for the final fifth session (A_5) in which the model has already been trained on all the five tasks of SVHN. We observe that our proposed method RKR outperforms CCLL using the ResNet-18 architecture.

5.6 Task-Agnostic SVHN Results

For the task-agnostic SVHN non zero-shot task incremental learning experiments, we compare RKR with several methods such as EWC [16], Online-EWC [19], SI [17], MAS [15], RPS-Net [23] and iTAML [5]. We report the final session accuracy (A_5) for all the methods. In Table 6, we observe that our method RKR

Datasets	Method	Task	1	2	3	4	5	6	7	8	9	Final Acc.
ImageNet-100/10	LwF [12] TPAMI'18	Aware	99.3	95.2	85.9	73.9	63.7	54.8	50.1	44.5	40.7	36.7
	CCLL [26] NIPS'20	Aware	99.8	99.0	99.2	98.6	98.4	98.5	98.2	97.7	97.8	97.9
	RKR (Ours)	Aware	99.6	99.1	99.2	99.0	99.0	99.0	98.9	98.5	98.6	**98.7**$_{+0.8}$
	iCaRL* [6] CVPR'17	Agnostic	99.3	97.2	93.5	91.0	87.5	82.1	77.1	72.8	67.1	63.5
	RPS-Net* [23] NIPS'19	Agnostic	100.0	97.4	94.3	92.7	89.4	86.6	83.9	82.4	79.4	74.1
	iTAML* [5] CVPR'20	Agnostic	99.4	96.4	94.4	93.0	92.4	90.6	89.9	90.3	90.3	89.8
	RKR (Ours)	Agnostic	99.6	99.1	99.2	99.0	99.0	99.0	98.9	98.5	97.16	**97.8**$_{+9.0}$
ImageNet-1K/10	CCLL [26] NIPS'20	Aware	91.4	88.3	86.5	86.6	84.6	83.5	82.7	81.7	81.2	81.3
	RKR (Ours)	Aware	90.2	88.7	88.1	88.2	86.6	85.7	85.0	84.2	83.8	**84.4**$_{+3.1}$
	iCaRL* [6] CVPR'17	Agnostic	90.1	82.8	76.1	69.8	63.3	57.2	53.5	49.8	46.7	44.1
	RPS-Net* [23] NIPS'19	Agnostic	90.2	88.4	82.4	75.9	66.9	62.5	57.2	54.2	51.9	48.8
	iTAML* [5] CVPR'20	Agnostic	91.5	89.0	85.7	84.0	80.1	76.7	70.2	71.0	67.9	63.2
	RKR (Ours)	Agnostic	90.2	88.7	86.9	87.7	83.9	82.2	79.0	78.1	77.2	**77.0**$_{+13.8}$

TABLE 5: Task incremental learning experiments (task-aware and task-agnostic) on the ImageNet-100 and ImageNet-1K datasets with 10 tasks. The reported accuracy for each task is the average of all accuracies up to that task. '$*$' denotes replay-based approach.

Methods	Task	SVHN(A_5)
GEM* [11]	Aware	75.61%
CCLL [26]	Aware	98.20%
RKR (Ours)	Aware	**99.04%**
EWC [16]	Agnostic	18.21%
Online-EWC [19]	Agnostic	18.50%
SI [17]	Agnostic	17.33%
MAS [15]	Agnostic	17.32%
RPS-Net* [23]	Agnostic	88.91%
iTAML*† [5]	Agnostic	93.97%
RKR† (Ours)	Agnostic	**97.71%**
RKR (Ours)	Agnostic	**98.12%**

TABLE 6: Task incremental experimental results (task-aware and task-agnostic) for the SVHN dataset with 5 tasks. The result for the other methods have been taken from [5], [26]. '$*$' denotes replay-based methods. † denotes model using ResNet-18/3 architecture.

with ResNet-18/3 outperforms all the compared methods and also outperforms iTAML (which also uses ResNet-18/3) by an absolute margin of 3.74%. We also observe that RKR with ResNet-18 outperforms all the methods. The average task prediction accuracy for RKR is over 98% for this setting.

5.7 Task-Agnostic CIFAR Results

For the task-agnostic non zero-shot incremental learning experiments on the CIFAR-100 datasets, we compare our approach RKR with various methods such as DMC [14], MAS [15], LwF [12], SI [17], EWC [16], RWalk [18], iCARL [6], RPS [23] and iTAML [5]. Fig. 6 shows the results for the non zero-shot experiments for the CIFAR-100 experiments with 5, 10, and 20 classes per task, respectively. From the results, we observe that as the number of tasks increases, the performance of iTAML and other methods drop significantly. However, the performance of RKR remains relatively stable as the number of tasks increases. Our results indicate that for the 10 classes per task experiments, our proposed method RKR with ResNet-18/3 significantly outperforms iTAML (which also uses ResNet-18/3) by an absolute margin of 6.22%. Similarly, for the 20 and 5 classes per task experiments, RKR with ResNet-18/3 significantly outperforms iTAML by absolute margins of 7.07% and 5.4%, respectively. RKR with ResNet-18 outperforms all other methods in each of these settings. The average task prediction accuracy for our method is over 90% for these experiments.

Significance of Components of the Task-Agnostic Testing Process: We observe in Table 7 that using the task weight hyper-parameters improves the model performance by an absolute margin of 1.52% for CIFAR-100 using ResNet-18. Additionally, using data augmentation approaches during the testing process as described in Sec. 4.4 further improves the model performance by an absolute margin of 1.29%.

TW	Aug	Acc.
✗	✗	84.19%
✓	✗	85.71%
✓	✓	87.00%

TABLE 7: Task incremental experimental results (task-agnostic) for the CIFAR-100 dataset with 10 tasks using ResNet-18 with different components of the task-agnostic testing process for RKR ($K = 2$). Aug refers to using data augmentation approaches in our proposed testing process.

Corruption	0%	10%	20%	30%
Acc.	86.23%	86.37%	87.00%	86.10%

TABLE 8: Task incremental experimental results (task-agnostic) for the CIFAR-100 dataset with 10 tasks using ResNet-18 for RKR ($K = 2$) with different corruption levels in the validation data continuum used for obtaining TW values.

Level of Corruption: In our proposed approach, we introduce corruption into the validation data continuum to increase the data points for the process of identifying suitable values for the TW hyper-parameters. We perform experiments to determine the most suitable level of corruption. We observe in Table 8 that 20% corruption is the most suitable level of corruption.

Effectiveness of using Data Augmentation in the task-agnostic testing process: As discussed in Sec. 4.4, differently augmented views of the same data continuum help in correctly identifying the correct task of the DC. The results in Table 9 indicate that even though the original DC has the highest AML for the wrong task, the average AML for the correct task across the differently augmented views of the DC is the highest and thereby, helps to correctly identify the task of the DC.

5.8 Task-Agnostic ImageNet Results

We perform task-agnostic non zero-shot incremental learning experiments on the large-scale ImageNet-100 and ImageNet-1K datasets. The results in Table 5 indicate that RKR significantly outperforms existing methods. Specifically, RKR significantly outperforms iTAML by absolute margins of 9.0% and 13.8% for the ImageNet-100 and ImageNet-1K datasets. The average final session task prediction accuracy for RKR is over 99% and 90% for ImageNet-100 and ImageNet-1K in this setting.

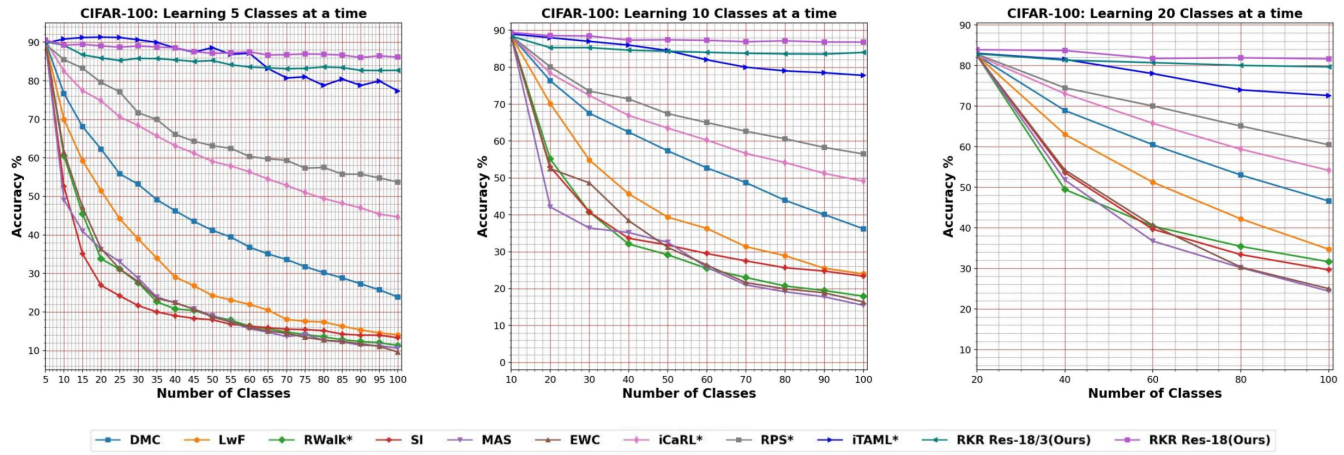


Fig. 6: Task incremental experimental results (task-agnostic) for the CIFAR-100 dataset with 5, 10 and 20 classes per task using ResNet-18/3 and ResNet-18. 5, 10 and 20 classes per task corresponds to 20, 10 and 5 tasks. Results for the other methods have been taken from [5]. '$*$' denotes replay-based approach.

	Tasks									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
DC	0.68	0.59	0.84	0.58	0.63	**0.86**	0.63	0.66	0.70	0.84
DC Aug. 1	0.72	0.63	**0.85**	0.68	0.67	0.81	0.71	0.66	0.80	0.81
DC Aug. 2	0.63	0.64	**0.83**	0.66	0.67	0.78	0.68	0.65	0.69	0.79
DC Aug. 3	0.71	0.62	**0.87**	0.60	0.64	0.75	0.68	0.71	0.77	0.84
Average	0.69	0.62	**0.85**	0.63	0.65	0.80	0.68	0.67	0.74	0.82

TABLE 9: AML values for a random test data continuum of the CIFAR-100 dataset (10 classes at a time) corresponding to the 10 tasks. DC represents the original data continuum. DC Aug. 1, DC Aug. 2, and DC Aug. 3 refer to 3 differently augmented views of DC obtained using the horizontal flip, rotation, and crop data augmentation approaches, respectively. Average represents the task-wise mean of the AML. Bold AML value for a DC refers to the maximum AML for that DC. The correct task of this DC is task 2.

6 Task Incremental Learning Experiments (Generalized Zero-Shot Setting)

The authors in [34] propose a task incremental generalized zero-shot learning setting using CADA-VAE [33] as the base architecture. This setting involves training the network on a sequence of datasets containing seen and unseen classes for generalized zero-shot learning. The image/visual features encoder in the network is common for all datasets and suffers from catastrophic forgetting in this setting. We apply our proposed approach RKR to solve the task incremental generalized zero-shot learning problem. Specifically, we learn weight rectifications and scaling factors to adapt the image/visual features encoder to any previously seen dataset/task without requiring any fine-tuning. For further details, please refer to the supplementary materials.

6.1 Testing Process

In the task-aware incremental generalized zero-shot learning setting, the model has access to the dataset/task label during test time [34]. Therefore, RKR quickly adapts the image encoder to the respective dataset using the corresponding weight rectifications and scaling factors using Eqs. 1, 2 and performs the generalized zero-shot classification.

We also explore a task-agnostic incremental generalized zero-shot learning setting, where the model does not have access to the dataset label during test time, and instead, at test time, we receive a seen/unseen class test data continuum with an unknown dataset/task. In order to solve this problem, we modify our approach similar to the non zero-shot setting. We follow the process for predicting the task as described in Sec. 4.4 to predict the dataset of the test data continuum in this setting.

6.2 Datasets

We experiment with four benchmark datasets for the task incremental generalized zero-shot learning (GZSL) problem i.e. Attribute Pascal and Yahoo (aPY) [44], Animals with Attributes 1 (AWA1) [45], Caltech-UCSD-Birds 200-2011 (CUB) [46], and SUN Attribute dataset (SUN) [47]. We extract the image features of 2048 dimensions from the final pooling layer of an ImageNet pre-trained ResNet-101. We follow the training split proposed in [45] so that the test classes do not overlap with the training classes.

6.3 Implementation Details

In this setting, RKR applies the weight corrections and scaling transformations to the visual features encoder of CADA-VAE framework used in this setting (refer to the supplementary materials for further details). We use $K = 16$ to generate weight rectifications in this setting and report the average results of 5 runs for our method. The CADA-VAE framework in this setting contains only fully connected layers. The SCM module in CCLL [26] only calibrates the convolutional layer outputs (feature maps). Therefore, CCLL is not compatible with CADA-VAE. We compare our method RKR with LZSL [34] and with the baseline methods proposed in [34] i.e., a) Sequential Fine-tuning (SFT): model is fine-tuned on new tasks sequentially, and the model parameters are initialized from the model trained on the previous task, b) L1 regularization (L1): model weights are initialized with the weights of the model trained on the previous task, and the model is trained with an L1-regularization loss between the previous and current network weights, c) L2 regularization (L2): same as (b) but with L2-regularization loss, d) "Base": model trained sequentially on all tasks without using any incremental learning methods or fine-tuning, e) "Original": trains separate networks for each task.

For the task-agnostic setting, we use a test data continuum of size 100. LZSL [34] also requires dataset/task label during testing.

Method	Task	Total	aPY			AWA1			CUB			SUN		
		Mem.	U	S	H	U	S	H	U	S	H	U	S	H
Base	Aware	100%	6.69	0.59	1.09	5.14	0.92	1.56	0.87	0.67	0.76	43.40	33.95	38.10
SFT	Aware	100%	24.24	23.21	23.71	47.27	55.18	50.92	35.46	34.74	35.10	38.47	36.10	37.20
L1	Aware	200%	26.42	29.79	28.01	49.64	58.23	53.59	35.11	32.31	33.65	40.14	34.11	36.88
L2	Aware	200%	24.08	23.61	23.84	46.71	59.07	52.17	35.53	33.24	34.35	42.08	32.33	36.56
LZSL	Aware	200%	29.11	43.29	34.81	51.17	63.66	56.73	38.82	45.81	42.03	42.43	31.78	36.34
RKR(Ours)	Aware	113%	33.39	51.34	**40.46**	58.79	69.36	**63.64**	47.52	49.22	**48.36**	42.22	36.01	**38.87**
LZSL	Agnostic	200%	27.94	43.29	33.96	51.17	63.66	56.73	36.49	45.81	40.62	38.19	31.78	34.69
RKR(Ours)	Agnostic	113%	31.05	51.34	**38.70**	57.03	69.36	**62.59**	47.52	49.22	**48.36**	42.22	36.01	**38.87**
Original	Aware	400%	30.36	59.36	40.18	57.30	72.80	64.10	53.50	51.60	52.40	35.70	47.20	42.60

TABLE 10: Classification accuracy (%) of the task incremental GZSL experiments (task-aware and task-agnostic) on the sequence of datasets aPY, AWA1, CUB, and SUN for our method RKR and other methods. LZSL [34] is the state-of-the-art method.

aPY			AWA1			CUB			SUN		
U	S	H	U	S	H	U	S	H	U	S	H
33.39	51.34	40.46	58.79	69.36	63.64	47.52	49.22	48.36	42.22	36.01	38.87
AWA1			aPY			CUB			SUN		
U	S	H	U	S	H	U	S	H	U	S	H
61.93	66.49	64.13	30.96	55.25	39.68	48.06	50.36	49.18	47.08	31.78	37.95
CUB			AWA1			aPY			SUN		
U	S	H	U	S	H	U	S	H	U	S	H
51.11	53.88	52.46	56.02	70.01	62.24	30.82	53.39	39.08	46.25	32.05	37.87
SUN			AWA1			CUB			aPY		
U	S	H	U	S	H	U	S	H	U	S	H
45.28	36.67	40.52	57.81	67.91	62.46	47.51	49.48	48.47	31.21	57.87	40.55

TABLE 11: Experimental results (task-aware) for RKR with different first dataset in the task incremental GZSL problem.

We use the same settings for identifying the suitable values for the TW hyper-parameters as described in Sec. 5.2. We run the process for identifying the TW hyper-parameters for only 2 iterations using a learning rate of 1e1 in this setting. Therefore, for a fair comparison, we apply the same dataset prediction approach to LZSL.

6.4 Task-Aware Results

Table 10 compares the performance of our method (task-aware) with the baselines, and LZSL [34] using the three evaluation metrics: unseen average class accuracy (U), seen average class accuracy (S), and harmonic mean of the two (H). The sequence of tasks/datasets is aPY, AWA1, CUB, and SUN, for a fair comparison with the other methods.

Table 10 also reports the total memory required by each method for the four tasks. LZSL requires 200% memory for the image feature encoder as it stores the image features encoder trained on the previous task to calculate the knowledge distillation loss. The L1 and L2 baselines also require 200% memory as they store the image features encoder trained on the previous task to calculate the L1/L2 loss between the weights of the two encoders. The "Original" model trains four separate networks for the four tasks and requires 400% total memory. Our method RKR requires around 3.28% additional parameters for each task. Therefore, on four tasks, RKR requires a total of about 113% memory for the image features encoder.

The "Base" model performs extremely badly on the first three tasks and manifests a clear case of catastrophic forgetting. SFT performs better than the "Base" model since it fine-tunes the model on the new task. However, its performance starts dropping for the older tasks as it learns new tasks. The forgetting is lower in SFT but is still substantial. We observe similar forgetting for the L1 and L2 baselines. Our method RKR significantly outperforms LZSL [34] as well as all the baseline methods. Specifically, RKR outperforms the state-of-the-art method LZSL by absolute margins of 5.65%, 6.91%, 6.33%, and 2.53% for the aPY, AWA1, CUB, and SUN datasets, respectively. We also compare the average H values across the four datasets. The average H values are 10.2%, 36.73%, 38.03%, 36.73% and 42.48% for base, SFT, L1, L2 and LZSL [34] respectively. The average H value for RKR is 47.83%, and that of the "Original" model is 49.82%. Therefore, RKR is significantly closer to the "Original" model as compared to LZSL.

Different First Task: Table 11 contains the results for different sequences of tasks/datasets having different first dataset. The H values for the AWA1 dataset with the first dataset as aPY, CUB, and SUN are 63.64%, 62.24%, and 62.46%. Considering the fact that aPY, CUB, and SUN have a large variation in the number of classes (aPY = 32, CUB = 200, SUN = 717), this variation in the result is minor. We observe the same pattern for the other three tasks with different first tasks. Therefore, our method works well for this setting, irrespective of the choice of the first task.

Significance of Components: We observe in Table 12 that without weight rectifications, RKR performs lower by absolute margins of 6.8%, 9.28%, and 6.24% for the AWA1, CUB, and SUN datasets, respectively. Similarly, without scaling, RKR performs lower by absolute margins of 3.71%, 4.57%, and 3.59% for the AWA1, CUB, and SUN datasets, respectively. Therefore, both weight rectifications and scaling factors are vital in this setting.

Value of K: Table 13 reports performances of RKR with different values of K. We observe that RKR with $K = 16$ performs close to RKR with $K = 32$ for most of the datasets but requires significantly less total memory, i.e., 113% vs. 126%. Therefore, we choose $K = 16$ for all our experiments in this setting, which significantly outperforms the state-of-the-art method.

Forward Knowledge Transfer In RKR, when a new task becomes available for training, we initialize the RG and SFG parameters from the previous task. We also experiment with training these parameters from scratch for each task. Table 14 reports the performance of our method RKR (task-aware) with the two types of initialization for the RG and SFG parameters. When we initialize these parameters from scratch, the model performs lower by absolute margins of 3.72%, 7.57%, and 7.14% from the other case, for AWA1, CUB, and SUN datasets, respectively. Therefore, forward transfer of knowledge takes place in RKR.

6.5 Task-Agnostic Results

Table 10 compares the performance of our method with LZSL [34] in the test setting where the model does not have access to the dataset/task labels (task-agnostic) during testing. The sequence of tasks/datasets is aPY, AWA1, CUB, and SUN, for a fair comparison with LZSL. We observe that RKR significantly outperforms LZSL even in this setting for all the datasets. This is even more significant considering we have applied the same

Wt. Rec.	Scaling	aPY			AWA1			CUB			SUN		
		U	S	H	U	S	H	U	S	H	U	S	H
✓	✗	32.34	52.78	40.10	52.50	69.80	59.93	45.12	42.54	43.79	40.97	30.97	35.28
✗	✓	29.97	52.08	38.05	52.82	61.53	56.84	39.17	39.00	39.08	36.81	29.30	32.63
✓	✓	33.39	51.34	40.46	58.79	69.36	63.64	47.52	49.22	48.36	42.22	36.01	38.87

TABLE 12: Classification accuracy (%) of the task incremental generalized zero-shot learning (task-aware) experiments on the aPY, AWA1, CUB and SUN datasets using RKR with different combinations of its components. Wt. Rec. refers to weight rectifications.

K	Train Mem.	aPY			AWA1			CUB			SUN		
		U	S	H	U	S	H	U	S	H	U	S	H
1	101%	30.33	58.78	40.01	55.66	69.49	61.81	40.11	40.68	40.39	40.56	30.50	34.82
4	104%	31.10	56.55	40.13	55.44	69.16	61.55	39.37	47.95	43.24	42.71	30.78	35.77
16	113%	33.39	51.34	40.46	58.79	69.36	63.64	47.52	49.22	48.36	42.22	36.01	38.87
32	126%	33.60	51.67	40.72	59.50	69.61	64.16	49.04	51.08	50.04	45.00	34.92	39.33

TABLE 13: Classification accuracy (%) of the task incremental generalized zero-shot learning (task-aware) experiments using our proposed RKR with different values of K.

Initialization	aPY			AWA1			CUB			SUN		
	U	S	H	U	S	H	U	S	H	U	S	H
Random	33.39	51.34	40.46	54.11	67.13	59.92	37.84	44.25	40.79	36.11	28.29	31.73
Previous	33.39	51.34	40.46	58.79	69.36	63.64	47.52	49.22	48.36	42.22	36.01	38.87

TABLE 14: Classification accuracy (%) of the task incremental generalized zero-shot learning (task-aware) experiments using our proposed RKR with different types of initialization: 1) random 2) from previous task.

dataset prediction approach to LZSL. In fact, the task-agnostic RKR also significantly outperforms the task-aware LZSL method that uses task labels during testing. For the CUB and SUN datasets, the task prediction accuracy is 100% for our method. The average dataset prediction accuracy in this setting is over 99%.

7 Conclusion

In this paper, we proposed a novel Rectification-based Knowledge Retention (RKR) approach to address the non zero-shot and zero-shot task incremental learning problems. We showed how RKR learns weight rectifications for the network weights and scaling factors for rectifying the intermediate outputs of the network in order to adapt the network to work for any given task. Our proposed method works for both the task-aware and task-agnostic test settings. We experimentally showed that our method significantly outperforms the state-of-the-art methods. We validated the components of our method using various ablation experiments. In the future, we would like to apply our approach to other computer vision tasks such as incremental object detection and segmentation.

References

[1] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[2] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[3] L. Zhang, T. Xiang, and S. Gong, "Learning a deep embedding model for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2021–2030.

[4] P. Singh, P. Mazumder, P. Rai, and V. P. Namboodiri, "Rectification-based knowledge retention for continual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[5] J. Rajasegaran, S. Khan, M. Hayat, F. S. Khan, and M. Shah, "itaml: An incremental task-agnostic meta-learning approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 588–13 597.

[6] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[7] R. Aljundi, K. Kelchtermans, and T. Tuytelaars, "Task-free continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 254–11 263.

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

[9] R. Kemker and C. Kanan, "Fearnet: Brain-inspired model for incremental learning," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=SJ1Xmf-Rb

[10] N. Kamra, U. Gupta, and Y. Liu, "Deep generative dual memory network for continual learning," *arXiv preprint arXiv:1710.10368*, 2017. [Online]. Available: https://arxiv.org/pdf/1710.10368.pdf

[11] D. Lopez-Paz and M. A. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 6467–6476.

[12] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2018.

[13] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 233–248.

[14] J. Zhang, J. Zhang, S. Ghosh, D. Li, S. Tasci, L. Heck, H. Zhang, and C.-C. Jay Kuo, "Class-incremental learning via deep model consolidation," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar 2020.

[15] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 139–154.

[16] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[17] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3987–3995.

[18] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," *Lecture Notes in Computer Science*, p. 556–572, 2018.

[19] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *ICML*, 2018, pp. 4535–4544.

[20] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural

networks," *arXiv preprint arXiv:1606.04671*, 2016. [Online]. Available: https://arxiv.org/pdf/1606.04671.pdf

[21] J. Xu and Z. Zhu, "Reinforced continual learning," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 899–908.

[22] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=Sk7KsfW0-

[23] J. Rajasegaran, M. Hayat, S. H. Khan, F. S. Khan, and L. Shao, "Random path selection for continual learning," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 12 669–12 679.

[24] J. Yoon, S. Kim, E. Yang, and S. J. Hwang, "Scalable and order-robust continual learning with additive parameter decomposition," in *International Conference on Learning Representations (ICLR)*, 2020. [Online]. Available: https://openreview.net/forum?id=r1gdj2EKPB

[25] A. Rosenfeld and J. K. Tsotsos, "Incremental learning through deep adaptation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 3, pp. 651–663, 2020.

[26] P. Singh, V. K. Verma, P. Mazumder, L. Carin, and P. Rai, "Calibrating cnns for lifelong learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 15 579–15 590.

[27] B. Cheung, A. Terekhov, Y. Chen, P. Agrawal, and B. Olshausen, "Superposition of many models into one," *Advances in neural information processing systems*, vol. 32, 2019.

[28] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 7765–7773.

[29] L. Chen, H. Zhang, J. Xiao, W. Liu, and S.-F. Chang, "Zero-shot visual recognition using semantics-preserving adversarial embedding networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1043–1052.

[30] R. Felix, V. B. Kumar, I. Reid, and G. Carneiro, "Multi-modal cycle-consistent generalized zero-shot learning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 21–37.

[31] Y. Zhu, M. Elhoseiny, B. Liu, X. Peng, and A. Elgammal, "A generative adversarial approach for zero-shot learning from noisy texts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1004–1013.

[32] Y. Xian, T. Lorenz, B. Schiele, and Z. Akata, "Feature generating networks for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5542–5551.

[33] E. Schonfeld, S. Ebrahimi, S. Sinha, T. Darrell, and Z. Akata, "Generalized zero-and few-shot learning via aligned variational autoencoders," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8247–8255.

[34] K. Wei, C. Deng, and X. Yang, "Lifelong zero-shot learning," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 551–557, main track.

[35] M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 41–50.

[36] W. Liu, X. Wang, J. Owens, and Y. Li, "Energy-based out-of-distribution detection," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 464–21 475, 2020.

[37] J. J. Thiagarajan, B. Venkatesh, P. Sattigeri, and P.-T. Bremer, "Building calibrated deep models via uncertainty matching with auxiliary interval predictors," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 6005–6012.

[38] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[39] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe, "Continual learning with hypernetworks," in *International Conference on Learning Representations (ICLR)*, 2020. [Online]. Available: https://openreview.net/forum?id=SJgwNerKvB

[44] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 1778–1785.

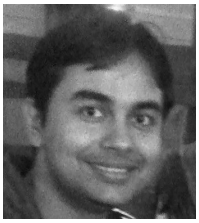
[45] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 9, pp. 2251–2265, 2018.

[46] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011. [Online]. Available: https://authors.library.caltech.edu/27452/1/CUB_200_2011.pdf

[47] G. Patterson and J. Hays, "Sun attribute database: Discovering, annotating, and recognizing scene attributes," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 2751–2758.

Pratik Mazumder received his M.Tech.-Ph.D. joint degree from Indian Institute of Technology, Kanpur. He is currently an Assistant Professor in the CSE department at IIT Jodhpur, India. His research interests include data-efficient learning techniques, few-shot learning, zero-shot learning and continual learning. He has published papers at prestigious international conferences and journals such as CVPR, ECCV, NeurIPS, AAAI, ICASSP, WACV, PR, Knowledge-based Systems, Neurocomputing, and others.

Pravendra Singh received his Ph.D. degree from Indian Institute of Technology, Kanpur. He is currently an Assistant Professor in the CSE department at IIT, Roorkee, India. His research interests include model compression, zero/few-shot learning, continual learning, deep learning. He has published papers at prestigious international conferences and journals, including CVPR, ECCV, WACV, NeurIPS, AAAI, IJCAI, ICASSP, IJCV, PR, Neurocomputing, IEEE JSTSP, and others.



Piyush Rai received his Ph.D. in Computer Science from University of Utah. He was a postdoctoral fellow at UT Austin, and thereafter he was a research faculty in ECE at Duke University. He is currently an Associate Professor in the CSE department at IIT Kanpur, India. His research interests include Bayesian modeling and inference, probabilistic latent variable models, few-shot and continual learning. He is also a recipient of various awards, some of which include IBM Faculty Award, Google India Faculty Award, Visvesvaraya Young Faculty Fellowship. He has published papers at prestigious international conferences and journals, including ICML, CVPR, NeurIPS, AAAI, IJCAI, IJCV, IEEE JSTSP, MLJ, and others.

Vinay P. Namboodiri received his M.Tech. and Ph.D. degrees from the Indian Institute of Technology, Bombay. He was a postdoctoral fellow at KU Leuven. He is currently a faculty member at University of Bath. He was previously an Associate Professor in the CSE department at IIT Kanpur, India. His research interests include visual recognition with scarce supervision, multimodal deep learning, explainable AI. He has served as an area chair for CVPR, ICCV, ACCV, WACV and BMVC. He has published papers at prestigious international conferences and journals, including CVPR, ICCV, ECCV, NeurIPS, AAAI, IJCAI, IJCV,IEEE TIP, PR, Neurocomputing, and others.