# Rectification-based Knowledge Retention for Continual Learning

Pravendra Singh[1,*],     Pratik Mazumder[2,*],     Piyush Rai[2],     Vinay P. Namboodiri[2,3]

[1]Independent Researcher, India     [2]IIT Kanpur, India     [3]University of Bath, United Kingdom

pravendra1988@gmail.com, pratikm@cse.iitk.ac.in, piyush@cse.iitk.ac.in, vpn22@bath.ac.uk

## Abstract

*Deep learning models suffer from catastrophic forgetting when trained in an incremental learning setting. In this work, we propose a novel approach to address the task incremental learning problem, which involves training a model on new tasks that arrive in an incremental manner. The task incremental learning problem becomes even more challenging when the test set contains classes that are not part of the train set, i.e., a task incremental generalized zero-shot learning problem. Our approach can be used in both the zero-shot and non zero-shot task incremental learning settings. Our proposed method uses weight rectifications and affine transformations in order to adapt the model to different tasks that arrive sequentially. Specifically, we adapt the network weights to work for new tasks by "rectifying" the weights learned from the previous task. We learn these weight rectifications using very few parameters. We additionally learn affine transformations on the outputs generated by the network in order to better adapt them for the new task. We perform experiments on several datasets in both zero-shot and non zero-shot task incremental learning settings and empirically show that our approach achieves state-of-the-art results. Specifically, our approach outperforms the state-of-the-art non zero-shot task incremental learning method by over 5% on the CIFAR-100 dataset. Our approach also significantly outperforms the state-of-the-art task incremental generalized zero-shot learning method by absolute margins of 6.91% and 6.33% for the AWA1 and CUB datasets, respectively. We validate our approach using various ablation studies.*

## 1. Introduction

Deep learning models are used to solve many real-world problems, and they have even surpassed human-level performance in many tasks. However, deep learning models generally require all the training data to be available at the beginning of the training. If this is not the case, deep learning models suffer from catastrophic

*The first two authors contributed equally.

forgetting [17], and their performance on the previously seen classes or tasks starts degrading. In contrast, human beings can continually learn new classes of data without losing the previously gained knowledge. To avoid catastrophic forgetting, deep learning models should perform lifelong/continual/incremental learning [5, 18]. Deep learning models also require all the classes to be present in the train set. When the test set contains classes not seen during training, the performance of these models degrades significantly [27, 39]. This is known as the zero-shot learning problem. The incremental learning problem becomes even more challenging when coupled with the zero-shot learning problem. In this paper, we solve for the task incremental learning problem in the zero-shot and non zero-shot setting.

The task incremental learning problem involves training the model on one task at a time, where each task has a set of non-overlapping classes. When a new task becomes available to the network for training, the previous task data is no longer accessible. Training on only the new task data causes the network to forget all the previous task knowledge. Therefore, the model has to prevent the forgetting of older tasks when training on new tasks. The task incremental generalized zero-shot learning problem involves training the model on one task at a time, where each task has a set of unseen classes not seen during training, which is the same as the zero-shot learning setting. The objective of the model is to successfully identify the seen and unseen classes of all the trained tasks.

We propose a novel approach called Rectification-based Knowledge Retention (RKR) for the task incremental learning problem in the zero-shot and non zero-shot setting. Our approach (RKR) learns weight rectifications to adapt the network weights for a new task. After learning these weight rectifications, we can quickly adapt the network to work for images from that task by simply applying these weight rectifications to the network weights. We utilize an efficient technique for learning these weight rectifications to limit the model size. We also learn affine transformations (scaling factors) for all the intermediate outputs of the network that allow better adaptation of the network to the respective task.

We perform various experiments 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. Using various ablation experiments, we 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 quickly adapt the network to the respective tasks.

- 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.

## 2. Related Work

### 2.1. Incremental Learning

Incremental learning 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 [21, 1, 37]. We can categorize incremental learning methods into: replay-based, regularization-based, and dynamic network-based methods.

Replay-based methods require saving data from old tasks in order to fine-tune the network along with the new task data to reduce forgetting. The authors in [21] propose to use an exemplar-based prototype rehearsal technique along with distillation. The methods proposed in [11, 10] use a custom architecture to produce pseudo samples for the older tasks to be used for rehearsal.

Regularization-based methods make use of regularization techniques to prevent network outputs from changing significantly while training on new tasks to preserve the knowledge gained from the older tasks. The work in [16] proposes to use knowledge distillation as the regularization technique. In [21, 2], the authors propose to use modified classification techniques suited to continual learning in addition to the distillation loss.

Dynamic network-based methods [33, 8, 3, 28, 15] use network expansions/modifications for training new tasks. The work in [23] proposes to create an extra network for each new task with lateral connections to the networks of the older tasks. The method proposed in [34] uses reinforcement learning to determine how many neurons to add

for each new task. DEN [36] performs efficient selective retraining and dynamically expands the network for each task with only the required amount of units. The method proposed in [20] uses a random path selection methodology for each task. The authors in [35] propose an order-robust approach APD, which uses task-shared and task-adaptive parameters. Recently the authors in [26] proposed CCLL that calibrates the feature maps of convolutional layer outputs to perform incremental learning.

Our method RKR follows the dynamic network approach, but it is the first work that learns rectifications for the layer weights and outputs to adapt the model to any task quickly. 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.

### 2.2. Zero-Shot Learning

Zero-Shot Learning (ZSL) [27, 39, 40, 4] is a problem setting where the model has to recognize classes not seen during training. In the generalized zero-shot learning setting, the test data can be 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 way to deal with ZSL is to learn an embedding space where the image data and the class embedding are close to each other [4]. Another popular approach is to generate images/features of unseen classes by using their class embeddings [7, 41]. The authors in [32] propose f-CLSWGAN, which uses conditional Wasserstein GANs, to generate features for unseen classes. Cycle-WGAN [7] improves upon f-CLSWGAN by using reconstruction regularization in order to preserve the discriminative features of classes. CADA-VAE [24] uses a VAE to learn to generate class embedding features in a latent embedding space to train a zero-shot classifier. However, these methods require data replay to work in the continual learning setting [30].

Recently, the authors in [30] proposed an approach LZSL for the task incremental generalized zero-shot learning problem. LZSL 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 [30]. 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.
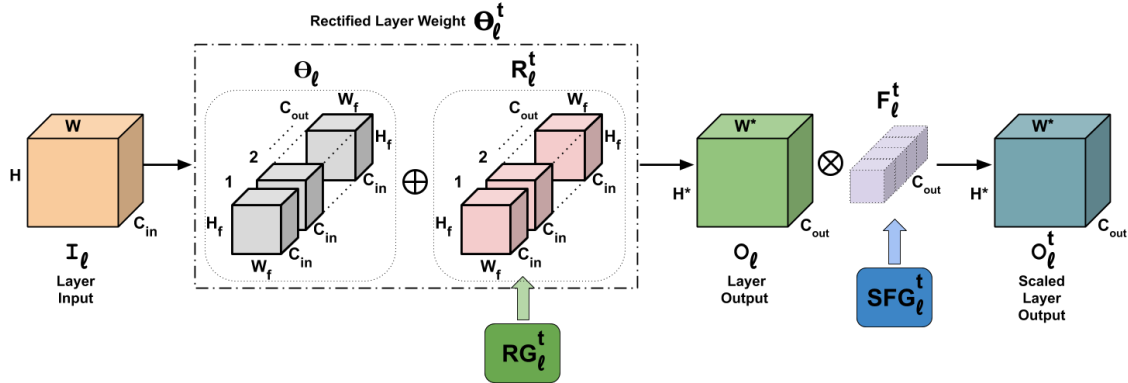
Figure 1: RKR for a convolutional layer. The weight rectifications $R_l^t$ produced by the rectification generator (RG$_l^t$) are added to the layer weights ($\Theta_l$) of the convolutional layer $l$ for task $t$. The task-adapted convolution layer weights ($\Theta_l^t$) are applied to the input $I_l$ 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 ($O_l^t$).

## 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 [30], 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 $\Theta_l$ 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 $\Theta_l$ 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 $\Theta_l^*$ 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 $\Theta_l$ for each task $t$ (Figs. 1, 2).

$$\Theta_l^t = \Theta_l \oplus R_l^t \qquad (1)$$

where $\Theta_l$ 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 $t$, $\Theta_l^t$ refers to the rectified weights of the layer $l$ for task $t$, $\oplus$ refers to element-wise addition. The layer weight $\Theta_l$ 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 $\Theta_l^t$.

Apart from the weight rectifications, we also learn scaling factors for performing affine transformations on the intermediate outputs generated by each layer of the network. We use a scaling factor generator (SFG) for learning 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.

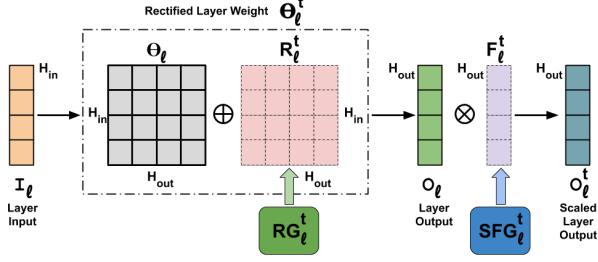$$O_l^t = O_l \otimes F_l^t \qquad (2)$$

Figure 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 ($\Theta_l$) of the fully connected layer $l$ for task $t$. The task-adapted fully connected layer weights ($\Theta_l^t$) 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 $O_l$ 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 quickly 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 requires 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 matrix factorization 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:

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

where `MATMUL` refers to matrix multiplication.

We apply the adapted convolution layer weights ($\Theta_l^t$) 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 $O_l$.

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

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 $\Theta_l$. Let $\Theta_l$ 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 $\Theta_l$ element-wise to produce the adapted layer weight $\Theta_l^t$. Therefore, RG computes the weight rectifications $R_l^t$ for task $t$ as follows:

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

where `MATMUL` refers to matrix multiplication.

We apply the adapted fully connected layer weights ($\Theta_l^t$) 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., $H_{out}$.

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

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.

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

## 5.1. Datasets

We perform the task incremental learning experiments on the CIFAR [13] and ImageNet [22] datasets for the non zero-shot setting. We perform experiments on CIFAR-100 with 10 tasks where each task contains 10 classes. For 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.

## 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 full 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. During testing, depending on the task, we apply the corresponding weight rectifications and scaling factors to the base network to work for that task.

For CIFAR-100 experiments, we use the ResNet-18 architecture [9]. For split CIFAR-10/100 experiments, we use ResNet-32 architecture [9]. 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 [14] on CIFAR-100. We train the network on each task for 100 epochs with the initial learning rate as 0.01 and multiply the learning rate with 0.5 at the 20, 40, 60, and 80 epochs. For 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. In all the cases, we run experiments for 5 random task orders and report the average accuracy. We perform experiments with $K = 2$ since this is a good choice considering the accuracy/extra-parameters trade-off as shown in Table 2. Our method utilizes task labels during testing similar to [26, 35, 36].

## 5.3. CIFAR-100 Results

For the experiments on the CIFAR-100 incremental learning tasks using 10 classes at a time, we perform experiments with various methods such as CCLL [26], SI [38], EWC [12], iCARL [21] and RPS [20]. CCLL uses task labels at test time, and we modify SI and EWC to use task labels during testing for a fair comparison. We additionally report the result for iCaRL, RPS-Net which are replay-
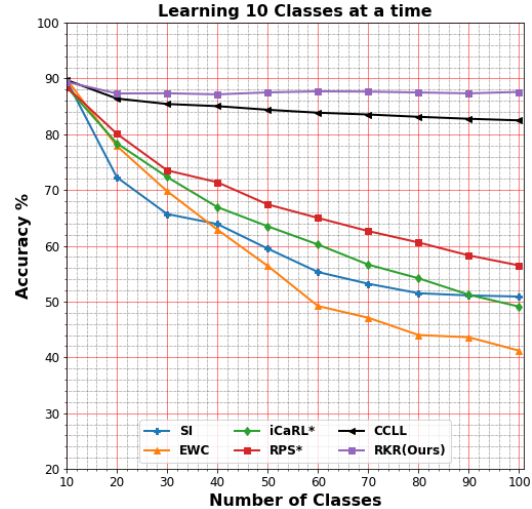


Figure 3: Experimental results on CIFAR-100 using ResNet-18. '$*$' denotes replay-based approach.

based methods. These methods store data from the previous tasks as additional data to use this data along with the new task data to train the network to reduce catastrophic forgetting. RKR does not store previous task data and is more scalable in this regard. We provide these additional results for completeness (replay-based and non replay-based). However, replay-based should not be directly compared with RKR. Figure 3 indicates that our approach RKR outperforms all existing methods. 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 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 experiments on the CIFAR-100 tasks using the LeNet architecture (20-50-800-500) as used in [35]. All the methods compared in Table 1 use task labels during testing. The results in Table 1 indicate that RKR outperforms existing state-of-the-art methods. Specifically, our

| Methods | Capacity | Accuracy |
|---|---|---|
| L2T [35] | 100% | 48.73% |
| EWC [12] | 100% | 53.72% |
| P&C [25] ICML'18 | 100% | 53.54% |
| PGN [23] | 171% | 54.90% |
| RCL [34] NIPS'18 | 181% | 55.26% |
| DEN [36] ICLR'18 | 181% | 57.38% |
| APD [35] 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**% |

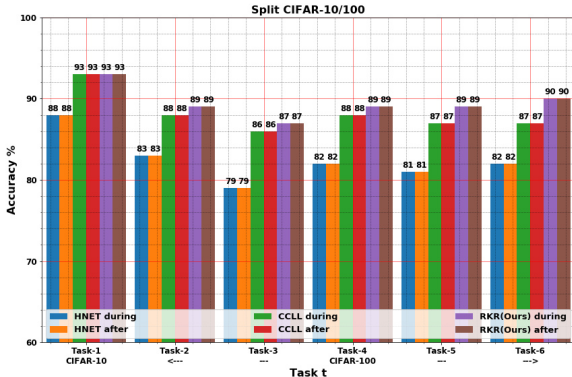Table 1: Experimental results on CIFAR-100 using LeNet.



Figure 4: Experimental results on split CIFAR-10/100 using ResNet-32 to check for catastrophic forgetting. We report the achieved accuracy 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).

approach RKR outperforms CCLL by an absolute margin of 5.87%. 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 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 [28], which are the state-of-the-art methods for this setup and use task labels during testing. We observe in Fig. 4 that RKR prevents catastrophic forgetting just like CCLL and HNET. 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, $\Theta_l$ 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 with ResNet-18 using different values of K. The results in Table 2 indicates that for $K = 2$ 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.
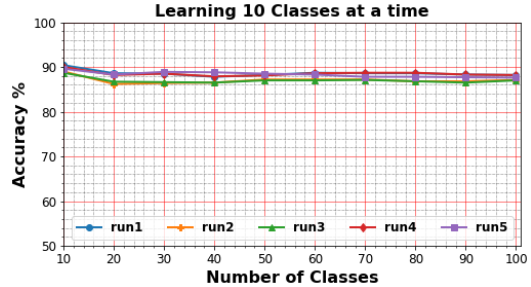


Figure 5: Experimental results for 5 runs of task incremental learning experiments using RKR on the CIFAR-100 dataset with ResNet-18.

| 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: Experimental results on CIFAR-100 with ResNet-18 using RKR for different values of K. We report the average accuracy of the 10 tasks.

| 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: Experimental results on CIFAR-100 with LeNet and ResNet-18 using RKR ($K = 2$) with different components. ✓ and ✗ refer to presence and absence respectively.

**Significance of Components:** In Table 3, we observe 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. However, we observe in task incremental generalized zero-shot learning that scaling helps to improve the RKR performance (Sec. 6.3).

**Forward Knowledge Transfer:** In RKR, 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 the ResNet-18. Therefore, forward transfer of knowledge occurs in RKR.

## 5.4. ImageNet-1K Results

From Table 4, we observe that RKR significantly outperforms the state-of-the-art CCLL method. Specifically, our method outperforms CCLL by an absolute margin of 3.1% (top-5 accuracy) 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.

| Method | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Final Acc. |
|---|---|---|---|---|---|---|---|---|---|---|
| iCaRL* [21] CVPR'17 | 90.1 | 82.8 | 76.1 | 69.8 | 63.3 | 57.2 | 53.5 | 49.8 | 46.7 | 44.1 |
| RPS-Net* [20] NIPS'19 | 90.2 | 88.4 | 82.4 | 75.9 | 66.9 | 62.5 | 57.2 | 54.2 | 51.9 | 48.8 |
| EEIL* [2] ECCV'18 | 95.0 | 95.5 | 86.0 | 77.5 | 71.0 | 68.0 | 62.0 | 59.8 | 55.0 | 52.0 |
| CCLL [26] NIPS'20 | 91.4 | 88.3 | 86.5 | 86.6 | 84.6 | 83.5 | 82.7 | 81.7 | 81.2 | 81.3 |
| RKR(Ours) | 90.2 | 88.7 | 88.1 | 88.2 | 86.6 | 85.7 | 85.0 | 84.2 | 83.8 | **84.4**$_{+3.1}$ |

Table 4: Task incremental learning experiments on the ImageNet-1K dataset with 10 tasks. The reported accuracy for each task is the average of all accuracies up to that task. '∗' denotes replay-based approach.

| Method | Total Mem. | aPY | | | AWA1 | | | CUB | | | SUN | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | U | S | H | U | S | H | U | S | H | U | S | H |
| Base | 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 | 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 | 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 | 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 | 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) | 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** |
| Original | 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 5: Classification accuracy (%) of incremental generalized zero-shot learning on the sequence of datasets aPY, AWA1, CUB, and SUN for our method RKR and other methods. LZSL [30] is the state-of-the-art method in this setting.

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

## 6.1. Datasets

We experiment with four datasets for the task incremental generalized zero-shot learning (GZSL) problem i.e. Attribute Pascal and Yahoo (aPY) [6], Animals with Attributes 1 (AWA1) [31], Caltech-UCSD-Birds 200-2011 (CUB) [29], and SUN Attribute dataset (SUN) [19]. 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 [31] so that the test classes do not overlap with the training classes.

## 6.2. Implementation Details

For the task incremental generalized zero-shot learning problem, we use the setting described in [30]. The authors in [30] propose a task incremental generalized zero-shot learning problem where each task is a separate dataset and show how a popular zero-shot learning approach, CADA-VAE [24] suffers from catastrophic forgetting in the image/visual feature encoder in this setting. We apply our approach to this setting and show that RKR achieves state-of-the-art results in this setting. In our approach, we apply the weight corrections and scaling transformations to the visual features encoder. We use $K = 16$ to generate weight rectifications in this setting and report the average results of 5 runs for our method. Please refer to the supplementary materials for further details.

We apply RKR to the CADA-VAE framework, which contains only fully connected layers. SCM in CCLL only calibrates convolutional layer outputs (feature maps). Therefore, CCLL cannot be applied to CADA-VAE. We compare our method RKR with LZSL [30] and with the baseline methods proposed in [30] i.e., a) Sequential Fine-tuning (SFT): model is fine-tuned on new tasks sequentially,

| 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 6: Experimental results for RKR with different first dataset in the task incremental GZSL problem.

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.

## 6.3. Results

Table 5 compares the performance of our method with the baselines, and LZSL [30] 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 5 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 en-

| 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 7: Classification accuracy (%) of task incremental generalized zero-shot learning 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 8: Classification accuracy (%) of task incremental generalized zero-shot learning using RKR with different $K$ values.

| 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 9: Classification accuracy (%) of task incremental generalized zero-shot learning using our proposed RKR with different types of initialization: 1) random 2) from previous task.

coder 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 [30] 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 [30] 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 6 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.

**Significance of Components:** From Table 7 we observe 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 8 reports performances of RKR with different values of $K$. $K = 16$ performs close to $K = 32$ 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 9 reports the performance of our method RKR 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.

# 7. Conclusion

We propose a novel approach called Rectification-based Knowledge Retention (RKR) for the task incremental learning problem. RKR learns rectifications to adapt the network weights and intermediate outputs for every new task. We empirically show that our approach significantly outperforms the state-of-the-art methods for task incremental learning problem in both the zero-shot and non zero-shot settings.

# References

[1] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11254–11263, 2019.

[2] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248, 2018.

[3] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations (ICLR)*, 2019.

[4] Long Chen, Hanwang Zhang, Jun Xiao, Wei Liu, and Shih-Fu Chang. Zero-shot visual recognition using semantics-preserving adversarial embedding networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1043–1052, 2018.

[5] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

[6] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785. IEEE, 2009.

[7] Rafael Felix, Vijay BG Kumar, Ian Reid, and Gustavo Carneiro. Multi-modal cycle-consistent generalized zero-shot learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2018.

[8] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[10] Nitin Kamra, Umang Gupta, and Yan Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017.

[11] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[12] 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*, 114(13):3521–3526, 2017.

[13] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[15] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3925–3934, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[16] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

[17] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pages 109–165. Elsevier, 1989.

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

[19] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2751–2758. IEEE, 2012.

[20] Jathushan Rajasegaran, Munawar Hayat, Salman H Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 12669–12679. Curran Associates, Inc., 2019.

[21] 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.

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[23] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[24] Edgar Schonfeld, Sayna Ebrahimi, Samarth Sinha, Trevor Darrell, and Zeynep Akata. Generalized zero-and few-shot learning via aligned variational autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8247–8255, 2019.

[25] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, pages 4535–4544, 2018.

[26] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. Calibrating cnns for lifelong learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Infor-*

*mation Processing Systems*, volume 33, pages 15579–15590. Curran Associates, Inc., 2020.

[27] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in Neural Information Processing Systems*, pages 935–943, 2013.

[28] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2020.

[29] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset, 2011.

[30] Kun Wei, Cheng Deng, and Xu Yang. Lifelong zero-shot learning. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 551–557. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[31] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2018.

[32] Yongqin Xian, Tobias Lorenz, Bernt Schiele, and Zeynep Akata. Feature generating networks for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5542–5551, 2018.

[33] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 177–186, 2014.

[34] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *Advances in Neural Information Processing Systems*, pages 899–908, 2018.

[35] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations (ICLR)*, 2020.

[36] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[37] Lu Yu, Bartlomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6982–6991, 2020.

[38] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

[39] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2021–2030, 2017.

[40] Bo Zhao, Xinwei Sun, Yanwei Fu, Yuan Yao, and Yizhou Wang. Msplit lbi: Realizing feature selection and dense estimation simultaneously in few-shot and zero-shot learning. *arXiv preprint arXiv:1806.04360*, 2018.

[41] Yizhe Zhu, Mohamed Elhoseiny, Bingchen Liu, Xi Peng, and Ahmed Elgammal. A generative adversarial approach for zero-shot learning from noisy texts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1004–1013, 2018.