Supplementary Materials for Rebalancing Batch Normalization for Exemplar-based Class-Incremental Learning

Sungmin Cha[1], Sungjun Cho[2], Dasol Hwang[2], Sunwon Hong[1], Moontae Lee[2,3], and Taesup Moon[1,4,5]*
[1]Department of ECE, Seoul National University [2]LG AI Research [3]University of Illinois Chicago
[4]ASRI / INMC / IPAI / AIIS, Seoul National University [5]SNU-LG AI Research Center
sungmin.cha@snu.ac.kr, {sungjun.cho, dasol.hwang}@lgresearch.ai,
zghdtnsz96@snu.ac.kr, moontae.lee@lgresearch.ai, tsmoon@snu.ac.kr

1. Implementation details of TBBN

There are some considerations to implementing our TBBN in exemplar-based CIL. Firstly, in order to use TBBN, the values for B_c, B_p, and information about task changes are required, and the ratio $\frac{B_c}{B_p}$ must be an integer. However, we believe that this information is readily available and adjustable in a general offline CIL scenario, as already shown in [1, 13]. Secondly, it should be noted that not all adaptively determined values for r can reshape a given feature map. For instance, when $\frac{B_c}{r}$ is not an integer, the tensor reshape operation F_{RS} cannot be applied. We overcome this limitation by using a simple rule for determining r. After calculating r using Equation (6) (as presented in the manuscript) at the beginning of each task training, we set a feasible r^* using the following rule:

$$r^* = \begin{cases} r, & \text{if } r \in CD(B_c, B_p) \\ M(B_c, B_p, r), & \text{otherwise} \end{cases} \tag{1}$$

where $M(B_c, B_p, r) = \max\{\hat{r} : \hat{r} \in CD(B_c, B_p) \wedge \hat{r} < r\}$ and $CD(\cdot,\cdot)$ denotes a set of common divisors between two values. Although r^* is not the exact optimal value for our TBBN, we already experimentally confirmed that using r^* is also effective for most CIL experiments in the manuscript. Finally, it should be noted that there is no difference between the original BN and our TBBN in the test phase because TBBN also maintains $\mu, \sigma^2, \gamma, \beta \in \mathbb{R}^C$ during the training phase.

2. Evaluation Metrics

Let $a_{t,i} \in [0, 1]$ denote the accuracy on the test set of task i after training on the first t tasks. The *final accuracy* $A_f = \frac{1}{T}\sum_{i=1}^{T} a_{T,i}$ measures the classification accuracy of the model at the end of training averaged across all tasks, and the *average accuracy* $A_a = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{1}{t}\sum_{i=1}^{t} a_{t,i}\right)$ measures the average accuracy until task T. Note that while these two metrics gauge the discriminative performance of the CIL pipeline, they do not reflect the stability-plasticity aspect, for which the following two metrics have been designed. The *forgetting measure* $F = \frac{1}{T}\sum_{i=1}^{T}\max_{t\in[i+1,T]}(a_{i,i} - a_{t,i})$ proposed by [3] measures the degree of forgetting by averaging the maximum decrease in accuracy of all tasks throughout the course of training. Lastly, the *learning accuracy* $A_l = \frac{1}{T}\sum_{i=1}^{T} a_{i,i}$ proposed by [11] measures the plasticity of the model by averaging the accuracy of each task immediately after training on that task. We report all measurements averaged across three runs with different seeds.

3. Additional Experimental Results

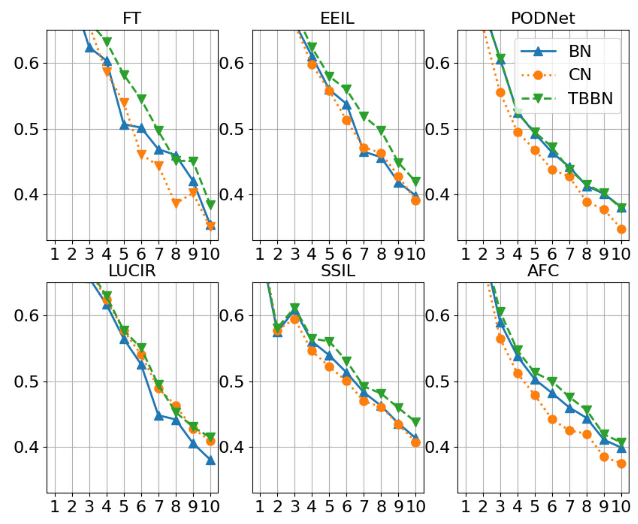
3.1. Accuracy curves

To visualize the task accuracies during training, Figure 1 displays the average classification accuracy across all previously seen tasks throughout the training process (A_f after each task). Our observations show that TBBN improves the average accuracy at every step of training compared to BN, whereas CN exhibits fluctuations that result in performance degradation when applied to AFC and PODNet.

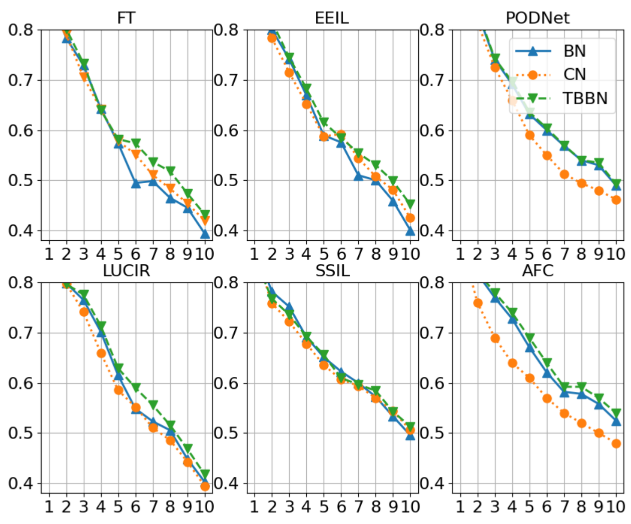
3.2. Experimental results for making a balanced batch with data augmentation

To confirm the novelty of TBBN, we conducted an experiment for making a balanced batch with data augmentation for sampled data in the exemplar memory. We selected powerful augmentation methods which are widely used for self-supervised contrastive learning, consisting of *RandomResizedCrop*, *RandomHorizontalFlip*, *ColorJitter* and *RandomGrayscale*. To make the balanced batch at each $t(> 1)$-th task's training time, we augmented each data sam-

*Corresponding author (E-mail: tsmoon@snu.ac.kr)



(a) CIFAR-100 (10 Tasks)



(b) ImageNet-100 (10 Tasks)

Figure 1. Accuracy curves during training with various CIL algorithms. X-axes show the number of tasks t, and the Y-axes show accuracies averaged across already seen t number of tasks. Replacing the BN layers (solid lines) with TBBN (dotted lines) leads to consistently better accuracies throughout training.

ple in the exemplar memory for $(t-1)\times 3$ times. This is because we set the ratio between the data points from the current task and those from the exemplar memory to $3:1$, and the sampled batch from the exemplar memory always contains data from $t-1$ numbers of the previous task. Table 1 shows the average result on CIFAR-100 for FT with balanced augmentation (FT + BalAug) with *ordinary* BN for 3 seeds. We observe that this baseline does not bring a positive performance gain, compared to the FT+BN in (Table 1, manuscript). We believe that the FT+BalAug has two limitations: 1) Despite the augmentation, the model ends up over-fitting to the samples in the exemplar memory due to the scarcity of data, and 2) $t\times 3$ times of augmentation for previous task's data enlarge the size of mini-batch for each task, causing the computation and memory cost increase during training. We believe this result further demonstrates the effectiveness of our TBBN for the exemplar-based CIL.

Table 1. Experimental results of FT + BalAug (with BN).

Acc / FM / LA	CIFAR-100 ($T=10$)
FT + BalAug	25.54 / 42.76 / 68.27

3.3. Experimental results for other CIL protocol (using base task)

We conducted experiments for another CIL scenario, which involves learning half of all classes as the first task (base task) and then incrementally learning the remaining tasks, as proposed in [4, 5]. We verified the effectiveness of TBBN for FT, LUCIR, and SS-IL on CIFAR-100 (with three seeds) in Table 2. The scenario considered here involves six tasks, where the model learns 50 classes as the base task and then continues to learn five tasks, each consisting of 10 classes. Note that this CIL scenario does not exactly correspond to the situation considered by TBBN (class-balanced tasks, see Section 3 of the manuscript). Nonetheless, the experimental results presented in Table 2 demonstrate that our TBBN can be successfully applied to several baselines, improving their performance compared to CN.

Table 2. Experimental results for various representative offline CIL methods in 6 tasks scenario (starting from learning the base task). Bold indicates the best performance in each metric.

Method		CIFAR-100 w/ ResNet-32			
		$A_f(\uparrow)$	$A_a(\uparrow)$	$F(\downarrow)$	$A_l(\uparrow)$
FT	+BN	35.86	45.71	**37.14**	77.32
	+CN	36.07	46.18	38.75	79.25
	+TBBN	**37.36**	**46.95**	37.33	**79.30**
EEIL	+BN	36.93	46.63	35.39	78.17
	+CN	38.44	47.45	**33.91**	78.23
	+TBBN	**38.83**	**47.82**	34.61	**78.62**
LUCIR	+BN	38.22	50.65	**22.87**	66.33
	+CN	38.20	49.74	24.99	**68.32**
	+TBBN	**39.54**	**50.95**	23.27	67.57
SSIL	+BN	45.69	53.03	8.55	53.63
	+CN	45.12	52.59	**7.63**	51.55
	+TBBN	**46.61**	**53.48**	8.94	**55.16**

Table 3. Experimental results (20 tasks).

$A_a(\uparrow)$		CIFAR-100	ImageNet-100
FT	+BN	29.66	39.06
	+CN($G = 16$)	30.12	37.82
	+TBBN	**34.45**	**43.84**
EEIL	+BN	35.11	37.89
	+CN($G = 16$)	35.49	38.07
	+TBBN	**39.32**	**42.30**
LUCIR	+BN	34.36	39.34
	+CN($G = 16$)	34.83	36.54
	+TBBN	**37.07**	**39.90**
SSIL	+BN	36.31	43.84
	+CN($G = 16$)	36.00	43.12
	+TBBN	**38.55**	**46.08**

3.4. Additional results from a 20-task setting

We also present additional results from a 20-task setting in Table 3. We see that TBBN brings significant performance boost in various CIL scenarios.

4. Details of Experimental Settings

In the experiments using FT, EEIL [2], LUCIR [5], and SSIL [1], we followed the CIL benchmark code proposed by [8]. The network was trained using SGD with an initial learning rate of 10^{-1} and weight decay of 10^{-4}, and a mini-batch size of 64. The number of epochs and schedule for adjusting the learning rate were set differently for each dataset and scenario. We used random sampling for ImageNet-100 experiments and herding [10, 12] for CIFAR-100 experiments. Table 4 provides detailed information on experimental settings and hardware used.

Table 4. Details of experimental settings.

	10 classes $\times$ 10 tasks		5 classes $\times$ 20 tasks	
	CIFAR-100	ImageNet-100	CIFAR-100	ImageNet-100
epochs per task	160	100	160	100
epoch for lr scheduling	[80, 120]	[40, 80]	[80, 120]	[40, 80]
lr decay	1/10	1/10	1/10	1/10
mini-batch size	64	64	64	64
model	ResNet-32	ResNet-18	ResNet-32	ResNet-18
python version	3.7	3.7	3.7	3.7
pytorch version	1.7.1+cu110	1.7.1+cu110	1.7.1+cu110	1.7.1+cu110
CUDA version	11.2	11.2	11.2	11.2
CuDNN version	8.1.1	8.1.1	8.1.1	8.1.1
GPU	TITAN XP	RTX A5000	1080Ti	1080Ti

In the case of experiments using PODNet [4] and AFC [6], we obtain the experimental results by implementing their official code. Also, we run each method with the default hyperparameter setting proposed in the official code.

5. Pseudo code of TBBN

Algorithm 1 shows the Pytorch-style pseudo algorithm for TBBN's forward function. It is important to note that TBBN does not require hyperparameter tuning and only uses easily accessible information such as the task number (t) and the number of sampled current (B_c) and memory (B_p) data.

6. Experiments for Online CL.

Table 5 presents the online CL results for CIFAR-100. We follow the experimental settings (ResNet-18, 20 tasks, single epoch and 2000 exemplars) proposed in [7], and only conduct experiments using finetuning (FT) for comparison. Our results show that CN with ($G = 8$) outperforms TBBN in both class-IL and task-IL. However, the performance gain of CN for class-IL is not as substantial as in [9] This trend was also shown previously with SplitTinyIMN in Table 4 of [9].

Table 5. Experimental results (20 tasks).

$A_a(\uparrow)$		Class-IL	Task-IL
FT	+BN	10.77	64.39
	+CN ($G = 8$)	**10.94**	**68.70**
	+CN ($G = 16$)	8.43	64.23
	+TBBN	10.12	67.43

Algorithm 1 Pytorch-style pseudo algorithm of the forward function of TBBN. Note that running_mean and running_val ($\in \mathbf{R}^C$) are initialized to 0 and 1, and gamma and beta ($\in \mathbf{R}^C$) are initialized to 1 and 0, respectively. m is the hyperparameter for the exponential moving average of running mean and standard deviation, and we set it to 0.9

```
def **TBBN_forward**(x, t, $B_c$, $B_p$, train):
    if train:
        if t $\neq$ 1:    # set r
            r $= B_c/B_p \cdot (t-1)$
        else:
            r $= 1$
        if r not in CD($B_c$, $B_p$):    # find r$^*$, CD returns common divisor of given two values (check Equation (1))
            r $= M(B_c, B_p,$ r)

        $N, C, H, W$ =x.shape
        # make balanced batch
        curr_batch = x[:$B_c$].reshape($B_c$/r, $C$·r, $H, W$)
        prev_batch = x[$B_c$:].repeat(1,r, 1, 1)
        bal_batch = concat((curr_batch, prev_batch), dim $= 1$)

        # calculate balanced mean and variance
        bal_mean = bal_batch.mean(dim $= [0, 2, 3]$)
        bal_val = bal_batch.val(dim $= [0, 2, 3]$)

        # normalize reshaped input batch
        bal_batch= $($bal_batch $-$ bal_mean$)/($bal_val $+\epsilon)$.sqrt()
        # affine-transform the normalized batch
        bal_batch=bal_batch$*$gamma.repeat(r) $+$ beta.repeat(r)

        # reshape bal_batch to the original shape
        bal_batch_curr = bal_batch[: $B_c$/r].reshape($B_c, C, H, W$)
        bal_batch_prev = bal_batch[$B_c$/r:].reshape($B_p$, r, $C, H, W$).mean(dim=1)
        x = concat((bal_batch_curr, bal_batch_prev), dim=1)

        # update running_mean and running_val
        running_mean_temp $=$ running_mean.repeat(r)
        running_val_temp $=$ running_val.repeat(r)
        running_mean_temp $=$ m $*$ running_mean_temp $+(1-$m$)*$ bal_mean
        running_val_temp $=$ m $*$ running_val_temp $+(1-$m$)*$ bal_val

        running_mean $=$ running_mean_temp.reshape(r, $C$).mean(dim = 0)
        running_val $=$ running_val_temp.reshape(r, $C$).mean(dim = 0)


    else:
        x= $($x$-$ running_mean$)/($running_val $+\epsilon)$.sqrt()
        x=x$*$gamma $+$ beta

    return x
```

References

[1] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (CVPR)*, pages 844–853, 2021.

[2] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremen-

tal learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248, 2018.

[3] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[4] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. PODNet: Pooled outputs distillation for small-tasks incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 86–102, 2020.

[5] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 831–839, 2019.

[6] Minsoo Kang, Jaeyoo Park, and Bohyung Han. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16071–16080, 2022.

[7] Zheda Mai. *Online Continual Learning in Image Classification*. PhD thesis, University of Toronto (Canada), 2021.

[8] Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: Survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[9] Quang Pham, Chenghao Liu, and Steven HOI. Continual normalization: Rethinking batch normalization for online continual learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[10] 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 (CVPR)*, pages 2001–2010, 2017.

[11] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[12] Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128, 2009.

[13] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 374–382, 2019.