Supplementary Information for Self-Supervised Curriculum-based Class Incremental Learning

Kartik Thakral[1], **Surbhi Mittal**[1], **Utkarsh Uppal**[2], **Bharat Giddwani**[2], **Mayank Vatsa**[1], **and Richa Singh**[1,*]

[1]IIT Jodhpur, India
[2]NVIDIA, India

Algorithms

This section outlines the formal algorithms corresponding to the proposed S^{2}C^{2}IL method. Algorithm 1 details the Stochastic Label Augmentation (SLA) process, which serves as the pretext training for each incremental task t. Additionally, we present the downstream training process that integrates Orthogonal Weight Modification (OWM) with feature smoothing, formally described in Algorithm 2. Finally, the complete S^{2}C^{2}IL methodology is summarized and formalized in Algorithm 3.

Algorithm 1: Pretext training using *SLA* for task *t*

```
Input: Images $X^t$, Downstream model conv-layer parameters $\theta_d^{t-1}$ from previous task $t-1$
Initialize: $\theta_d^0$ is zero-initialized.
Parameters: Number of epochs $E$, Number of stochastic tasks $M$, Number of classes per stochastic task $N$, Pretext
  model conv-layer parameters $\theta_p^t$, Pretext model FC parameters $\psi_p^t$, hyperparameters $a$ and $b$
Function train_pretext_model($X^t$, $\theta_d^{t-1}$)
    Initialize model $f(\theta_p^t, \upsilon_p^t, \psi_p^t)$
    $R^t = generate\_stochastic\_labels(X^t, m, n)$
    for $e$=1 to $E$ do
        $\hat{R}^t = f(X^t; \theta_p^t, \upsilon_p^t, \psi_p^t)$
        Calculate loss terms:
        $L_1 = \sum_{i=1}^{M}(\sum_{j=1}^{N} -R^t log(\hat{R}^t))$
        $L_2 = (a/2)\left\|\theta_p^t - \theta_d^{t-1}\right\|_2^2 + (b/2)\left\|\theta_p^t\right\|_2^2$
        $L = L_1 + L_2$
        Backpropagate loss $L$ and update $\theta_p^t$ and $\psi_p^t$
    end
    return $\theta_p^t$
end
Function generate_stochastic_labels($X^t$, $M$, $N$)
    for $i$=1 to $M$ do
        Sample each label from a uniform distribution:
        $R_i^t = y_i \sim \frac{1}{|N|}$
    end
    return $R^t$
end
```

Experimental Setup

The proposed algorithm is evaluated on four benchmark datasets: split-CIFAR10, split-CIFAR100, split-SVHN, and split-TinyImageNet, and two high-resolution datasets: split-STL10 and ImageNet-100. We report the average test accuracy, which is defined as the average of test accuracies achieved across all tasks. All experiments are performed using five fixed random seeds. The proposed algorithm is evaluated under two settings- (i) OWM + CL, and (ii) S^{2}C^{2}IL. In the first setting, only the

Algorithm 2: Downstream training for task t

```
Input: Images $X^t$, Labels $Y^t$, Pretext model conv-layer parameters $\theta_p^t$ for current task $t$
Initialize: $\theta_d^t$ is initialized with $\theta_p^t$.
Parameters: Number of epochs $E$, Downstream model FC parameters $\phi_d^t$, Gaussian filter $G$ with standard deviation $\sigma$,
  constant $c$
Function train_downstream_model($X^t$, $Y^t$, $\theta_d^t$)
    Initialize model $f(\theta_d^t, \phi_d^t)$
    Set $\sigma_e = 1$
    for $e$=1 to $E$ do
        $\hat{a}_n = f(X^t; \theta_d^t, \phi_d^t)$
        $\sigma_e = \sigma_e.c$
        $z_{n+1} = pool(G(\sigma_e) * \hat{a}_n)$
        $\hat{Y}^t = argmax(z_{n+1})$
        Calculate loss terms:
        $L = \sum_{i=1}^{M}(\sum_{j=1}^{N} -Y^t log(\hat{Y}^t))$
        Backpropagate loss $L$ and update $\theta_d^t$ and $\phi_d^t$ using OWM algorithm.
    end
    return $\theta_d^t$, $\phi_d^t$
end
```

Algorithm 3: S^{2}C^{2}IL algorithm

```
Input: Total tasks $T$, Images $X$, Labels $Y$
Parameters: Pretext model conv-layer parameters $\theta_p$, Pretext model FC parameters $\psi_p$, Downstream model conv-layer
  parameters $\theta_d$, Downstream model FC parameters $\phi_d$
for $t$ = 1 to $T$ do
    $\theta_p^t = train\_pretext\_model(X_t, \theta_d^{t-1})$                    //Algorithm 1
    $\theta_d^t$, $\phi_d^t = train\_downstream\_model(X_t, Y_t, \theta_p^t)$     //Algorithm 2
end
Evaluate model $f(\theta_d^T, \phi_d^T)$ trained for $T$ tasks.
```

curriculum-based downstream model is trained without any self-supervision. In the S^{2}C^{2}IL setting, we follow the methodology as explained in Section 3 of the main paper, and perform pre-training using the proposed SLA technique.

Datasets and Protocol: Since the focus of this work is class-incremental setting, we train and test the proposed algorithm according to the protocols defined in the works of[1] and[2]. For experiments, we have used six datasets:

(i) **Split-CIFAR10**[3] contains 60,000 32×32 color images of 10 different classes with 50,000 images in the training set and 10,000 images in the testing set. The training and evaluation is performed for 2 classes per task.

(ii) **Split-CIFAR100**[3] contains 60,000 32×32 color images of 10 different classes with 50,000 images in the training set and 10,000 images in the testing set. The training and evaluation are done for 10, 20, and 50 classes per task.

(iii) **Split-SVHN**[4] contains 60,000 32×32 color images of 10 different classes with 50,000 images in the training set and 10,000 images in the testing set. The training and evaluation are performed for 2 classes per task.

(iv) **Split-TinyImageNet**[5] contains 120,000 color images of size 64×64 from 200 different classes with 100,000 images in the training set, 10,000 in the validation set and 10,000 images in the testing set. The training and evaluation of the model are done for 5, 10, and 20 classes per task.

Comparitive Algorithms:

The results of the proposed framework are compared with various benchmark algorithms in the domain of regularization-based CIL with the exception of iCaRL. The following algorithms are used for comparison: (1) EWC[6], (2) iCaRL[7] with 2000 exemplars; (3) PGMA[2]; (4) DGM[8], (5) OWM[1], (6) MUC[9], (7) IL2A[10], (8) PASS[11], (9) SSRE[12], and FeTrIL[13]. The EWC[14], iCaRL[1], DGM[15], OWM[16], MUC[17], IL2A[18], PASS[19], SSRE[20], and FeTrIL[21] baselines are run using open-source codes with the same network architecture as the one used in S^{2}C^{2}IL. The details of this network are described in Section . Further, S^{2}C^{2}IL is compared with various memory-based approaches on the Split-TinyImageNet dataset. It should be noted that the proposed S^{2}C^{2}IL algorithm uses *no* exemplars from classes of previous tasks.

	Algorithm 1	**Algorithm 2**	**Algorithm 3**
TC	$\mathcal{O}(n^t + m)$	$\mathcal{O}(N_u + N_w^2)$	$\mathcal{O}(N_u + N_w^2)$
SC	$\mathcal{O}(mc)$	$\mathcal{O}(N_w^2)$	$\mathcal{O}(N_w^2)$

Table 1. Time Complexity (TC) and Space Complexity (SC) for each algorithm of the proposed S^{2}C^{2}IL algorithm.

Technique	**Time Taken (in seconds)**
Rotation Pretext	9 s
Colorization	8 s
SLA	2 s

Table 2. Time taken (in seconds) for an epoch by different pretext tasks on split-CIFAR100 dataset for 5 incremental tasks.

Implementation Details:
For all the experiments, we use a 3-layer CNN network with three fully-connected layers. The same network architecture is used by[1]. For each incremental task, we start the model training on the pretext task. Here, we use the 3-layer CNN architecture for feature extraction and utilize these features in a multitask fashion. For our experiments, we train the model for three tasks with two classes each, i.e., the extracted features are utilized by three separate heads of fully-connected layers with two layers each. For the downstream task, the same weights from the pretext task are transferred. However, here the features are utilized by a single fully-connected layer to learn the current incremental task. The mentioned architectures used in the pretext and downstream can be better visualized in Figure 1 (a) and (b). We train all the models with stochastic gradient descent (SGD). For the pretext task, the multitask network is trained on stochastically generated labels for three tasks with two classes each. We set the learning rate to 0.001 to train it for 50 epochs. The hyperparameters a and b are fixed to 10 and 18 for split-CIFAR10 and split-CIFAR100 datasets and 5 and 12 for split-SVHN datasets, respectively. As described in Section 3 of the main paper, the model is trained for a curriculum where the training starts with σ set to 0.9 with a decay rate of 0.95 for every 10 epochs for split-CIFAR100 and split-SVHN datasets. For the split-CIFAR10 dataset, σ is set to 1 with a decay rate of 0.9 for every ten epochs. All experiments are performed for five random seeds and the performance is reported as the mean and standard deviation over all the seeds. The algorithm is implemented in Pytorch, and all the experiments are performed on a DGX station with 256 GB RAM and four 32 GB Nvidia V100 GPUs.

Additional Experiments

Time and Space Complexity of the SLA and S^{2}C^{2}IL: The space and time complexity of each algorithm utilized in the proposed S^{2}C^{2}IL algorithm is reported in Table 1. Here, n^t depicts the number of images in incremental step t, m is the count of multi-tasks utilized in pretext, c stands for the number of classes in pretext, N_u stands for the total number of neurons and N_w are the number of input weights/neuron in the backbone network.

We also compute the time taken by the proposed SLA algorithm and compare it with existing SSL techniques. Table 2 showcases that the proposed SLA algorithm takes significantly less time when compared with existing pretraining algorithms on split-CIFAR100 dataset for 5 incremental tasks. We also compute the training time of the existing algorithms (available in Table 1 from the main paper for an epoch on split-CIFAR100 dataset for 5 incremental tasks). We observe that the existing algorithms, such as MUC[22] and SSRE[12] require about 8 seconds, and IL2A[10] and FeTrIL[13] require over 200 seconds to train for a single epoch. The proposed S^{2}C^{2}IL algorithm takes 6 seconds for each epoch, making it computationally efficient when compared to most of the existing baseline algorithms.

Dataset	**Tasks**	**S^{2}C^{2}IL (w/o PWS)**	**S^{2}C^{2}IL**
split-SVHN	5	75.88 ± 0.45	77.53 ± 0.53
split-CIFAR10	5	60.75 ± 0.53	61.64 ± 0.57
split-CIFAR100	2	43.38 ± 0.20	43.98 ± 0.65
	5	35.52 ± 0.48	35.59 ± 0.49
	10	31.85 ± 0.45	31.93 ± 0.54

Table 3. Performance comparison of the proposed algorithm without and with Penultimate Weight Sharing (PWS). The proposed Stochastic Label Augmentation (SLA) task is used for unsupervised pre-training.

Impact of Penultimate Weight Sharing (PWS):

Typically, pretext task weights are used for downstream task training, with feature quality improving in deeper layers. However, deeper layers are often biased towards the pretext task[23]. To mitigate this bias, we propose Penultimate Weight Sharing (PWS), which transfers weights from all layers except the last for downstream fine-tuning. PWS shares layer weights (θ_p^t), excluding the final layer (υ_p^t), allowing the network to learn generalized features for task t. By avoiding pretext-specific weights, PWS promotes better feature transfer. To evaluate the effect of PWS, we remove the υ_p^t convolution block, aligning the pretext model's architecture with the downstream model. We pre-train the model, transfer the weights, and assess performance on various datasets. Table 3 demonstrates the significant improvements achieved by transferring weights up to the penultimate layer.

References

1. Zeng, G., Chen, Y., Cui, B. & Yu, S. Continual learning of context-dependent processing in neural networks. *Nat. Mach. Intell.* **1**, 364–372 (2019).
2. Hu, W. *et al.* Overcoming catastrophic forgetting for continual learning via model adaptation. In *International conference on learning representations* (2018).
3. Krizhevsky, A. Learning multiple layers of features from tiny images. 32–33 (2009).
4. Netzer, Y. *et al.* Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011* (2011).
5. Le, Y. & Yang, X. Tiny imagenet visual recognition challenge. *CS 231N* **7**, 3 (2015).
6. Kirkpatrick, J. *et al.* Overcoming catastrophic forgetting in neural networks. *Proc. national academy sciences* **114**, 3521–3526 (2017).
7. Rebuffi, S.-A., Kolesnikov, A., Sperl, G. & Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2001–2010 (2017).
8. Ostapenko, O., Puscas, M., Klein, T., Jahnichen, P. & Nabi, M. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11321–11329 (2019).
9. Liu, Y. *et al.* More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16*, 699–716 (Springer, 2020).
10. Zhu, F., Cheng, Z., Zhang, X.-Y. & Liu, C.-l. Class-incremental learning via dual augmentation. *Adv. Neural Inf. Process. Syst.* **34**, 14306–14318 (2021).
11. Zhu, F., Zhang, X.-Y., Wang, C., Yin, F. & Liu, C.-L. Prototype augmentation and self-supervision for incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5871–5880 (2021).
12. Zhu, K., Zhai, W., Cao, Y., Luo, J. & Zha, Z.-J. Self-sustaining representation expansion for non-exemplar class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9296–9305 (2022).
13. Petit, G., Popescu, A., Schindler, H., Picard, D. & Delezoide, B. Fetril: Feature translation for exemplar-free class-incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 3911–3920 (2023).
14. Masana, M. *et al.* Class-incremental learning: Survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis Mach. Intell.* **45**, 5513–5533, DOI: 10.1109/TPAMI.2022.3213473 (2023).
15. Ostapenko, O., Puscas, M., Klein, T., Jahnichen, P. & Nabi, M. Learning to remember: A synaptic plasticity driven framework for continual learning. https://github.com/SAP-archive/machine-learning-dgm?tab=readme-ov-file (2019). Accessed: January 10, 2025.
16. Zeng, G., Chen, Y., Cui, B. & Yu, S. Continual learning of context-dependent processing in neural networks. https://github.com/beijixiong3510/OWM (2019). Accessed: January 10, 2025.
17. Liu, Y. *et al.* More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning. https://github.com/liuyudut/MUC (2020). Accessed: January 10, 2025.
18. Zhu, F., Cheng, Z., Zhang, X.-Y. & Liu, C.-l. Class-incremental learning via dual augmentation. https://github.com/Impression2805/IL2A (2021). Accessed: January 10, 2025.

19. Zhu, F., Zhang, X.-Y., Wang, C., Yin, F. & Liu, C.-L. Prototype augmentation and self-supervision for incremental learning. https://github.com/Impression2805/CVPR21_PASS (2021). Accessed: January 10, 2025.

20. Zhu, K., Zhai, W., Cao, Y., Luo, J. & Zha, Z.-J. Self-sustaining representation expansion for non-exemplar class-incremental learning. https://github.com/zhukaii/SSRE/ (2022). Accessed: January 10, 2025.

21. Zhou, D.-W. & Wang, F.-Y. Pycil: A python toolbox for class-incremental learning. https://github.com/G-U-N/PyCIL (2024). Accessed: January 10, 2025.

22. Liu, X. *et al.* Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 226–227 (2020).

23. Misra, I. & Maaten, L. v. d. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6707–6717 (2020).