Appendix: On the Stability-Plasticity Dilemma of Class-Incremental Learning

Dongwan Kim[1] Bohyung Han[1,2]
Computer Vision Laboratory, ECE[1] & IPAI[2], Seoul National University
{dongwan123, bhhan}@snu.ac.kr

1. More details on partial-DER

Table A. Performance of Partial-DER, with varying branch locations.

Method	DER	pDER (Layer 2)	pDER (Layer 3)	pDER (Layer 4)
Acc($\mathcal{M}_0', \mathcal{D}$)	60.0	60.0	60.0	60.0
Acc($\mathcal{M}_5', \mathcal{D}$)	66.9	67.8	68.0	**68.4**
$\Delta\mathcal{M}_5'$	6.9	7.8	8.0	**8.4**
Avg. Inc. Acc.	69.1	69.2	69.4	**69.7**
Acc($\mathcal{M}_5, \mathcal{D}$)	63.8	64.1	64.4	**64.7**
GMACs ($\mathcal{F}_5$)	10.9	8.0	5.9	**3.9**

1.1. Scalability issues in DER

As mentioned Section 5.1, DER offers strong CIL performance but suffers from scalability. For example, in our ImageNet-B500 5step setting, the last stage model for DER maintains 6 full ResNet-18 models. To make matters worse, in the ImageNet-B500 10step setting, DER maintains 11 full ResNet-18 models. This affects inference as well, since an input must pass through all feature extractors before classification. Although the authors do propose a masking/pruning scheme to reduce the memory complexity, they do not provide an implementation in their official code despite the fact that masking/pruning is an integral part of their algorithm. Naturally, we raise certain doubts on the reproducibility of DER with masking, and question whether the benefits of improved performance so overwhelmingly outweigh the loss of scalability.

1.2. Partial-DER ablations

The pDER method that we introduced in Section 5.1 aims to improve scalability of DER by only maintaining a subset of layers from all stages. This specific instance of pDER is the pDER Layer 4 variant, where only the parameters of ResNet Layer 4 are replicated and trained at each incremental stage. We also test with other variants:

- pDER Layer 3: replicate and train ResNet Layers 3 and 4; fix all parameters upto Layer 3
- pDER Layer 2: replicate and train ResNet Layers 2, 3, and 4; fix all parameters upto Layer 2

The results of our ablations are presented in Table A. Surprisingly, we find that as we apply DER on deeper layers, the performance actually *increases* across all metrics: Acc($\mathcal{M}_5', \mathcal{D}$) increases by 0.6%p, the average incremental accuracy improves by 0.5%p, and Acc($\mathcal{M}_5, \mathcal{D}$) improves by 0.6%p from pDER Layer 2 to Layer 4. This is achieved all while reducing the GMACs for a single-input forward pass.

2. Feature representations of ImageNet B0-10step models

In Figure A we present the classifier finetuning analysis of DER, AFC, and Oracle models on the ImageNet B0-10step setting. The purpose of this plot is to demonstrate that we observe a lack of plasticity for CIL models even in the B0-10step setting, which does not use a pre-trained base model. The purple line shows the progression of finetuning accuracy as the DER model is trained continually on 100 classes at a time. As seen in the plot, it closely follows the Oracle, but falls short by around 5% points by the last stage. On the other hand, the blue line, which represents AFC, shows a much different trend. At 100 classes, AFC has a much higher (base) accuracy of 42.1% (vs. $\sim 35\%$ of DER and Oracle), while the final accuracy is much lower at 48.8% (vs. 70.8% for the Oracle and 64.8% for DER). The 6.7% increase in accuracy is rather trivial when considering that the final model has seen 9 times the data of the base model. Thus, even in the B0-10step setting, we observe that AFC significantly lacks plasticity in its feature representations.

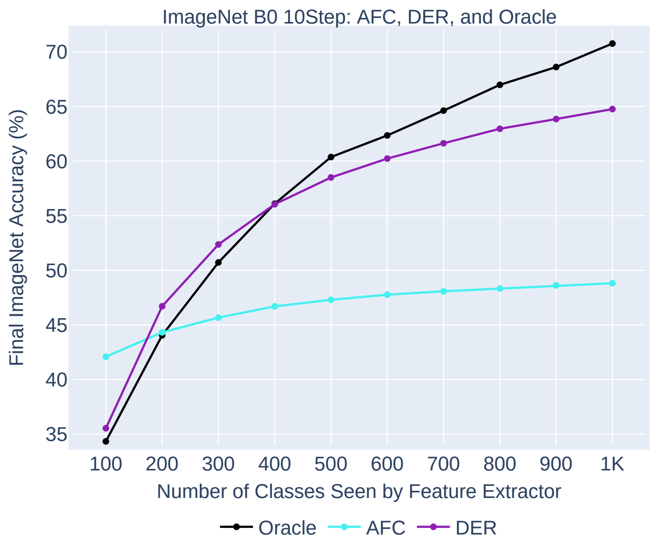

Figure A. Accuracy on the ImageNet validation set after fine-tuning the classification layer of each incremental model (B0-10step setting) with the full ImageNet training data. The black line indicates the Oracle model trained on {100, 200, ..., 1000} classes, and serves as a point of reference for the performance on non-incremental settings.

3. Mini-batch CKA

Eq. (3) of our main paper describes CKA, which takes as input $\mathbf{X} \in \mathbb{R}^{b\times z_1}$ and $\mathbf{Y} \in \mathbb{R}^{b\times z_2}$. In our case, b is equivalent to the number of samples in $\mathcal{D}_0$, *i.e.*, $|\mathcal{D}_0| = 25$K for the ImageNet validation set. Storing matrices $\mathbf{X} \in \mathbb{R}^{25000\times z_1}$ and $\mathbf{Y} \in \mathbb{R}^{25000\times z_2}$, where z_1 and z_2 are the dimensions of the flattened output features, requires excessive memory especially when CKA is computed on the GPU. Furthermore, we require $\mathbf{X}$ and $\mathbf{Y}$ for all layers of ResNet-18, which makes storing such matrices even less feasible.

To alleviate the memory burden, we utilize the mini-batch variant of CKA proposed by [5]. The main difference is that the mini-batch CKA uses an unbiased estimator of HSIC:

$$\text{HSIC}_1(\mathbf{K},\mathbf{L}) = \frac{1}{n(n-3)}\left(\text{tr}(\tilde{\mathbf{K}}\tilde{\mathbf{L}}) + \frac{\mathbf{1}^\intercal\tilde{\mathbf{K}}\mathbf{1}\mathbf{1}^\intercal\tilde{\mathbf{L}}\mathbf{1}}{(n-1)(n-2)} - \frac{2}{n-2}\mathbf{1}^\intercal\tilde{\mathbf{K}}\tilde{\mathbf{L}}\mathbf{1}\right), \tag{a}$$

where $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{L}}$ are equivalent to $\mathbf{K}$ and $\mathbf{L}$ with their diagonals set to zero, and n denotes the size of the mini-batch. Thus, given activation matrices $\mathbf{X}_i \in \mathbb{R}^{n\times z_1}$ and $\mathbf{Y}_i \in \mathbb{R}^{n\times z_2}$, the mini-batch CKA formulates to:

$$\text{CKA}_{\text{mini-batch}} = \frac{\sum_{i=1}^{k}\text{HSIC}_1(\mathbf{X}_i\mathbf{X}_i^\intercal, \mathbf{Y}_i\mathbf{Y}_i^\intercal)}{\sqrt{\sum_{i=1}^{k}\text{HSIC}_1(\mathbf{X}_i\mathbf{X}_i^\intercal, \mathbf{X}_i\mathbf{X}_i^\intercal)}\sqrt{\sum_{i=1}^{k}\text{HSIC}_1(\mathbf{Y}_i\mathbf{Y}_i^\intercal, \mathbf{Y}_i\mathbf{Y}_i^\intercal)}}, \tag{b}$$

where k denotes the number of iterations. Following [5], we use a batch size of $n = 256$ and iterate over $\mathcal{D}_0$ 10 times to compute mini-batch CKA.

4. More implementation details

4.1. Classifier types

Linear classifier The linear classifier is a simple matrix multiplication with an added bias term, and is formulated as below:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{c}$$

where $\mathbf{y} \in \mathbb{R}^{c \times 1}$ denotes the output logit vector (with c classes), $\mathbf{x} \in \mathbb{R}^{z \times 1}$ denotes the z-dimensional output of the feature extractor, $\mathbf{b} \in \mathbb{R}^{c \times 1}$ denotes the bias term, and $\mathbf{W} \in \mathbb{R}^{c \times z}$ denotes the weight matrix. Note that some implementations do not use the bias term.

Cosine classifier The output of the cosine classifier is formulated as:

$$\mathbf{y}_i = s \cdot \frac{\mathbf{W}_i \mathbf{x}}{\|\mathbf{W}_i\| \|\mathbf{x}\|}, \tag{d}$$

where $\mathbf{y}_i$ denotes the logit for the i-th class, and $\mathbf{W}_i \in \mathbb{R}^{1 \times z}$ denotes the i-th row vector of the weight matrix, $\mathbf{W}$. Finally, s denotes the scale factor, which may be either fixed or set as a learnable parameter. Note that the scale factor is used solely for training purposes, but does not affect the prediction output during evaluation.

4.2. CIL algorithm implementation details

Common details All compared CIL models use a ResNet-18 backbone with varying classifier types. When the cosine classifier is used, the output of the feature extractor is not subject to ReLU activation, following POD [1]. Furthermore, all implementations (except SSIL) employ the same class orderings for ImageNet.

Naive and Oracle For the Naive and Oracle models, we employ the cosine classifier with a fixed scale factor of $s = 24$. We use a batch size of 512, an initial learning rate of $\mathrm{lr} = 0.1$ and a polynomial learning rate decay scheme with a power of 0.9 over the course of 120 epochs. For data augmentation, we use the standard sequence: {`random resized crop`, `random horizontal flip`}. For the naive model, instead of the herding selection scheme, we randomly select 20 exemplars from each class to fill up the exemplar set.

POD and AFC For POD and AFC, we train models using their official codes[1] [2]. One important detail to note is that both POD and AFC originally used a modified version of ResNet, where the first convolution layer (`conv1`) had `kernel_size=3, stride=1, padding=1` as opposed to `kernel_size=7, stride=2, padding=3` from the original ResNet. Thus, we fixed this detail and re-ran their code to obtain the POD and AFC models. While POD and AFC both use the local similarity classifier [1,4], we use the cosine classifier when retraining the classification layer.

iCARL, LUCIR, and AANet For iCARL, LUCIR, and AANet, we obtained the trained models from the Energy-based Latent Aligner (ELI) [3] codebase[3]. Unfortunately, we do not evaluate ELI itself, since it requires the high-level distinction between previous and new classes for prediction. iCARL, LUCIR, and AANet all use the cosine classifier, where the scale factor s is also set as a trainable parameter.

SSIL We received the trained SSIL models directly from the authors. SSIL models use an oridinary linear classifier with a bias term. Note that the trained SSIL models used a different class ordering compared to all other methods, although this does not affect the conclusions made in our paper.

DER We use the official DER code[4] to train DER models. While their code does not provide configurations for ImageNet-1K experiments, we reproduced results using the details from the paper. However, the authors do not provide code for the masking operation, and thus, the reported results are the ones for full DER without masking/pruning. This makes the final DER models (for both the 5-step and 10-step settings) extremely large. DER uses the ordinary linear classifier without the bias term.

[1]POD: https://github.com/arthurdouillard/incremental_learning.pytorch
[2]AFC: https://github.com/kminsoo/AFC
[3]ELI: https://github.com/JosephKJ/ELI
[4]DER: https://github.com/Rhyssiyan/DER-ClassIL.pytorch

Retraining the classifier on full ImageNet data To retrain the classifiers on full ImageNet data, we freeze all layers prior to the final classification layer, including Batch Normalization layers (whose running means and variances are fixed). We select the appropriate classifier layer type depending on what type of classifier was used to train the original models. We train the classifiers for 60 epochs, using the same batch size, learning rate, and learning rate decay schemes as detailed in the ”Naive and oracle” paragraph.

5. Changes in Task Similarity

Table B. Retrained classifier accuracy on the ImageNet-C [2] B500-5step setting.

Method	$\text{Acc}(\mathcal{M}'_0, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_1, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_2, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_3, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_4, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_5, \mathcal{D})$
Perturbation	Clean	Gaussian Noise	Contrast Shift	-	-	-
AFC	70.8	70.8	70.9	71.6	71.7	72.1
DER	67.3	70.3	72.3	76.2	78.3	79.3

To investigate whether our analyses are consistent when the task similarity changes, we design an experiment under the CIL setting by injecting larger distribution shifts in each sequential task. More specifically, for each $\mathcal{D}_{i>0}$ in the ImageNet B500-5step setting, we apply perturbations/corruptions [2] to the input images instead of training with the original images. We select a unique type of perturbation for each incremental stage and apply this perturbation for all samples within a given stage, *e.g.* all samples of $\mathcal{D}_1$ are perturbed by random gaussian noise, while samples of $\mathcal{D}_2$ are perturbed by contrast shift, and so on. By doing so, we inject a domain shift between samples of different stages, thereby decreasing the task similarity. We train AFC and DER under this setting and report the results of our analysis in Table B. Table B shows that the trend is clear; while DER shows significant improvement from $\mathcal{M}_0$ to $\mathcal{M}_5$ (*i.e.* $\Delta\mathcal{M}_5 = 12.0$), AFC exhibits relatively minor improvements in performance (*i.e.* $\Delta\mathcal{M}_5 = 1.3$). Thus, our findings are consistent even when task similarity changes.

6. AANet CKA Anomaly

The CKA curve for AANet is quite interesting since it exhibits some erratic behavior. We observe 4 major drops, which start at layers 8, 20, 32, and 44. Interestingly, these layers are all 12 layers apart, and all correspond to a specific `3x3` convolution within the ResNet-18 architecture. We believe that this is due to the unique design of AANet, where two models (one “stable” and one “plastic”) are fused together after each ResNet Layer. Our hypothesis is that the representations diverge between the AANet fusion steps, and converge again when features of both branches are added together in the fusion step.

7. GDumb

Table C. Retrained classifier performance for GDumb (with CutMix [6]).

$\text{Acc}(\mathcal{M}'_0, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_1, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_2, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_3, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_4, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_5, \mathcal{D})$
61.0	56.4	56.5	57.0	56.8	56.9

Results Much like our Exploit, the motivation behind GDumb is to question the general progress in continual learning research. Thus, we do not expect the GDumb model to exhibit favorable properties, and the results in Table C match our expectations; GDumb starts with a base model ($\text{Acc}(\mathcal{M}'_0, \mathcal{D})$ = 61.0), but the becomes less performant from the first incremental stage ($56.4 \leq \text{Acc}(\mathcal{M}'_i, \mathcal{D}) \leq 56.9, \forall i > 0$).

Implementation Since GDumb does not experiment on ImageNet-1K, we reproduce GDumb in our own code base. One important detail is that for any setting with pre-trained models (e.g., ImageNet B500-5step), GDumb will use the model pre-trained on 500 classes as the initialization for each incremental step. Although their paper states that a model is trained “from scratch”, the official GDumb code actually re-loads the pre-trained model for each incremental step (for methods that require pre-training). Also, following the official implementation, we include CutMix [6] for GDumb, which is not used for any of the other compared methods.

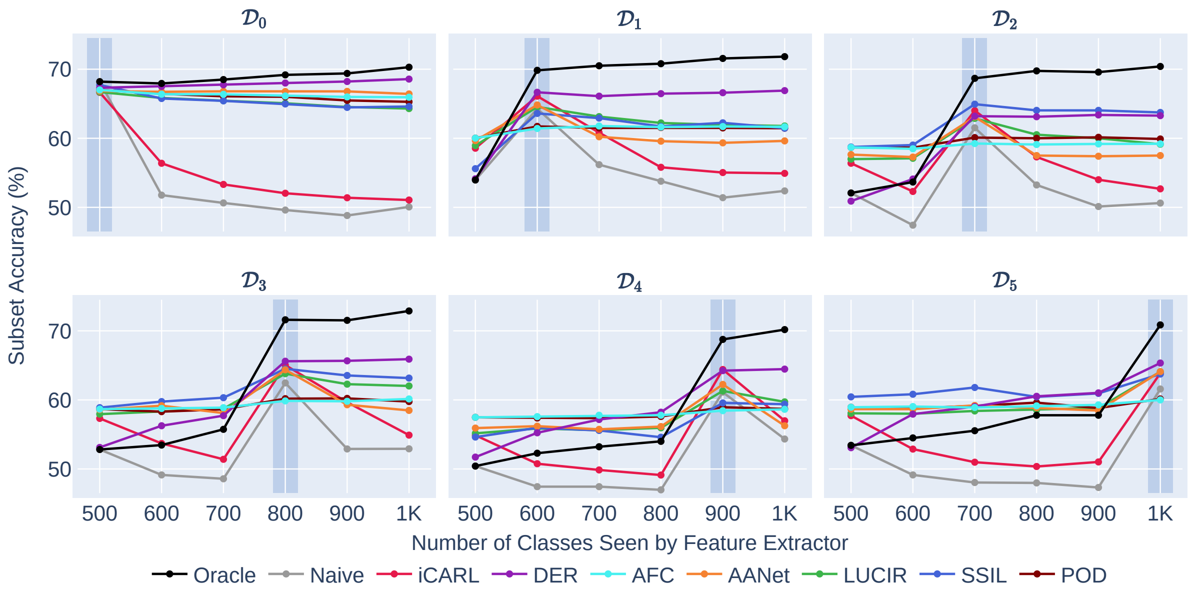

Figure B. B500-10step subset accuracies for all methods. Note that SSIL uses a different class ordering. We highlight the region for model $\mathcal{M}'_j$ in plot $\mathcal{C}_i$, where $i = j$.

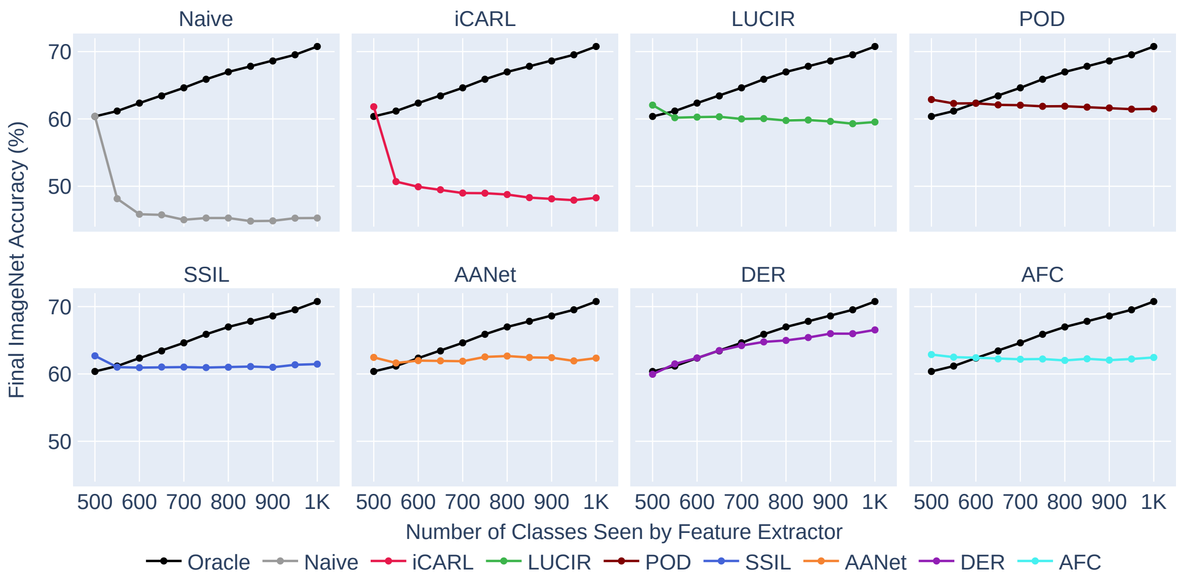

Figure C. Accuracy on the ImageNet validation set after fine-tuning the classification layer of each incremental model (B500-10step setting) with the full ImageNet training data. The black line indicates an oracle model trained on $\{500, 550, 600, ..., 1000\}$ classes, and serves as a point of reference for the performance on non-incremental settings.

8. Full 5 step subset accuracies

We present the full ImageNet B500-5step subset accuracies in Figure B.

9. 10 step results

In this section, we present figures for analyses on the ImageNet-1K B500-10step setting. Overall, the evaluated CIL algorithms all show similar trends in both the B500-5step and B500-10step settings. Thus, the B500-10step results serve to validate our observations made on the B500-5step setting.

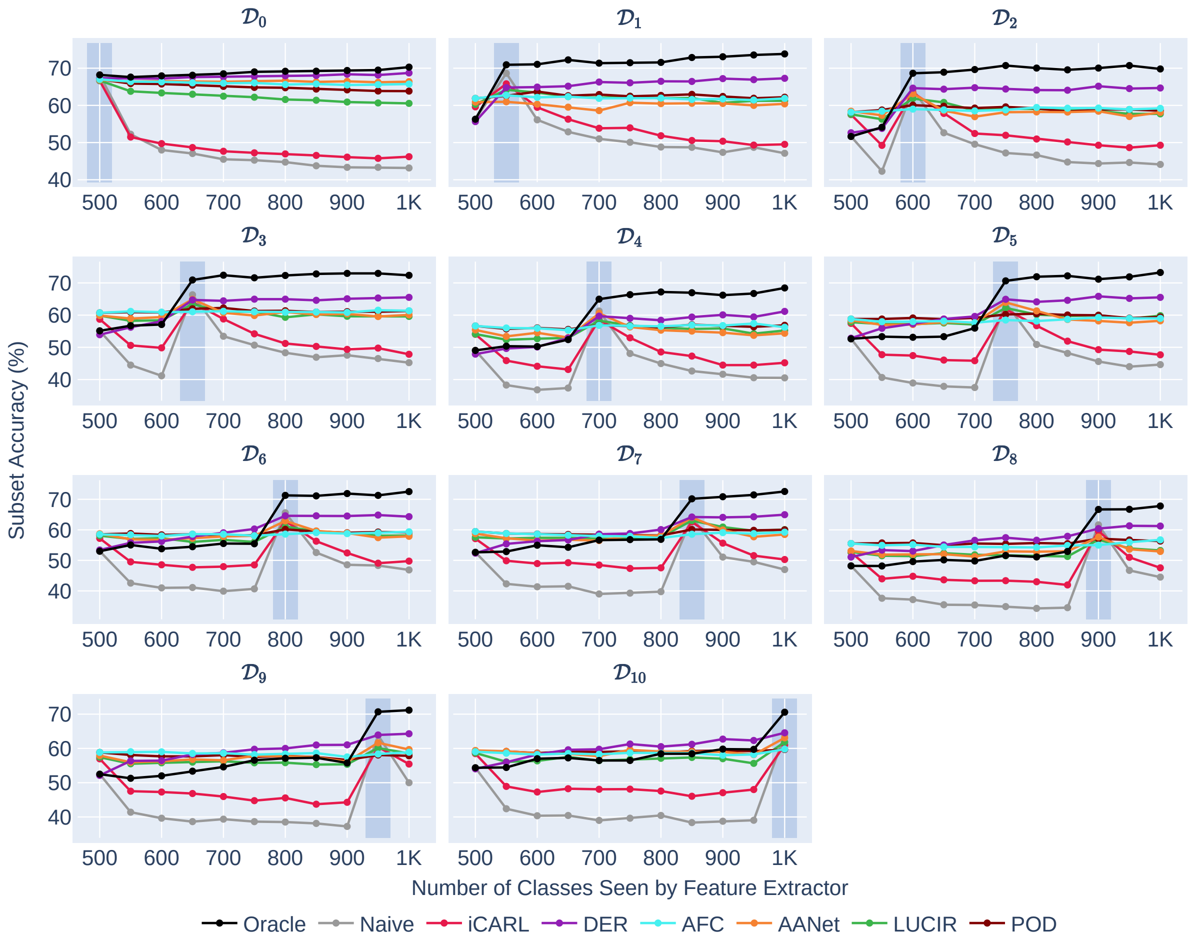

Figure D. B500-10step subset accuracies for all methods. Note that SSIL uses a different class ordering. We highlight the region for model $\mathcal{M}'_j$ in plot $\mathcal{C}_i$, where $i = j$.

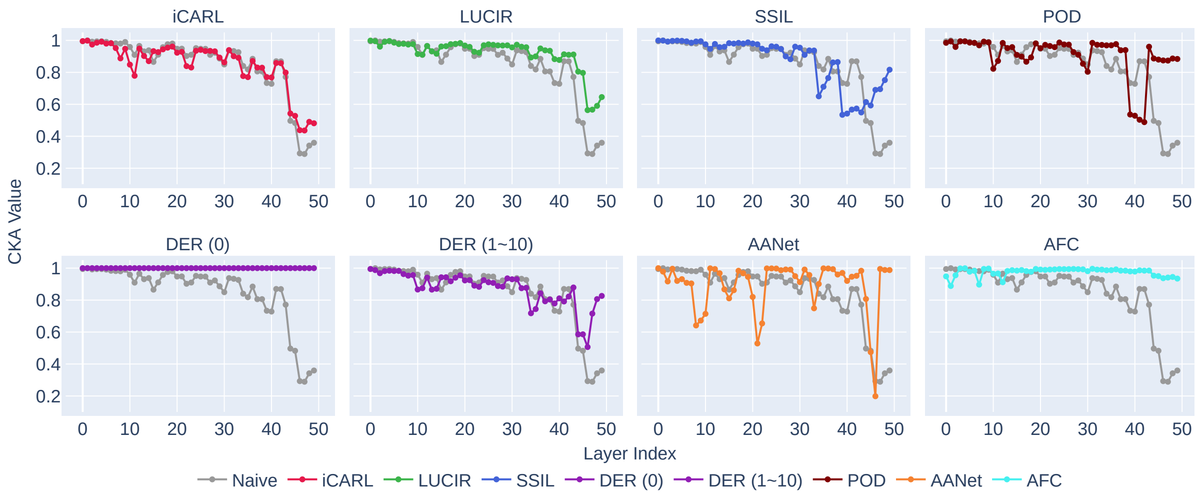

Figure E. Same-layer CKA values between $\mathcal{M}_0$ and $\mathcal{M}_{10}$ for incremental models trained with each CIL algorithm on the ImageNet B500 10-step setting. The x-axis spans the layer index of ResNet-18, while the y-axis represents CKA. CKA is evaluated using the $\mathcal{D}_0$ validation set. Each plot is accompanied by the CKA for the naive model, which acts as a point of reference.

References

[1] Arthur Douillard, Matthieu Cord, Charles Ollion, and Thomas Robert. PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning. In *ECCV*, 2020. 3

[2] D. Hendrycks and T. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019. 4

[3] KJ Joseph, Salman Khan, Fahad Shahbaz Khan, Rao Muhammad Anwar, and Vineeth Balasubramanian. Energy-based latent aligner for incremental learning. In *CVPR*, 2022. 3

[4] Minsoo Kang, Jaeyoo Park, and Bohyung Han. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *CVPR*, 2022. 3

[5] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *ICLR*, 2021. 2

[6] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 4