A DEEP DIVE INTO THE STABILITY-PLASTICITY DILEMMA IN CLASS-INCREMENTAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

A fundamental objective in class-incremental learning is to strike a balance between stability and plasticity, where models should be both stable enough to retain knowledge learnt from previously seen classes, and plastic enough to learn concepts from new classes. While previous works demonstrate strong performance on class-incremental benchmarks, it is not clear whether their success comes from the models being stable, plastic, or a mixture of both. In this paper we aim to shed light on how effectively recent class-incremental learning algorithms address the stability-plasticity trade-off. We establish analytical tools that help measure the stability and plasticity feature representations, and employ such tools to investigate models trained with various class-incremental algorithms on large-scale class-incremental benchmarks. Surprisingly, we find that the majority of class-incremental algorithms heavily favor stability over plasticity, to the extent that the feature extractor of a model trained on the initial set of classes is no less effective than that of the final incremental model. Our observations not only inspire two simple algorithms that highlight the importance of analyzing feature representations, but also suggest that class-incremental research, in general, should strive for better feature representation learning.

1 INTRODUCTION

Despite the unprecedented success of deep learning (Vaswani et al., 2017; Radford et al., 2021; Redmon et al., 2016; Noh et al., 2015), most of deep neural networks are optimized for static use cases. However, real-world problems often require adaptivity to incoming data (Kirkpatrick et al., 2017), changes in training environments, and domain shifts (Ben-David et al., 2010; Ganin & Lempitsky, 2015). Thus, researchers have been actively working on model adaptation techniques, and have proposed a variety of continual learning approaches so far.

A naïve approach for continual learning is to simply fine-tune a model. However, such a solution is rather ineffective due to a phenomenon known as *catastrophic forgetting*, which arises as a result of high *plasticity* of a model; parameters important for the old tasks are updated to better fit the new data. On the flip side, enforcing model *stability* introduces its own set of limitations, mainly the lack of adaptivity to new data. Thus, we encounter the *stability-plasticity dilemma*: how can we balance stability and plasticity such that the model is able to learn new concepts while maintaining old ones? Finding an optimal balance these two opposing forces is a core challenge of continual learning research, and has been the main focus of many previous works (Rebuffi et al., 2017; Liu et al., 2021; Douillard et al., 2020; Kang et al., 2022; Yan et al., 2021).

We conduct an in-depth study of recent works in continual learning, specifically concentrating on class-incremental learning (CIL)—a subfield of continual learning—where new sets of classes arrive in an online fashion. We are motivated by the lack of systematic analyses in the field of CIL, which hampers our understanding of how effectively the existing algorithms balance stability and plasticity. Moreover, works that do perform analyses usually focus on the classifier, *e.g.,* classifier bias (Ahn et al., 2021; Hou et al., 2019), rather than the intermediate feature representations. However, investigating the stability and plasticity in the feature level is just as important, if not more, because the capability to learn robust representations by making full use of the model's capacity is critical to maximize the potential of CIL algorithms.

To measure plasticity, we retrain the classification layer of CIL models at various incremental stages and investigate how effectively the feature representations learn new concepts. We then investigate stability by measuring feature similarity through Centered Kernel Alignment (CKA) (Kornblith et al., 2019; Cortes et al., 2012) and by visualizing the feature distribution shift using t-SNE (van der Maaten & Hinton, 2008). Suprisingly, and possibly concerningly, our analyses show that the majority of CIL models accumulate little new knowledge in their feature representations across incremental stages. In fact, most of the analyzed CIL algorithms effectively alleviate catastrophic forgetting, but only do so by heavily overlooking model plasticity in favor of high stability. Finally, we introduce two simple algorithms based on our analyses. The first is an extension of Dynamically Expandable Representations (DER) (Yan et al., 2021), which demonstrates how our analyses may be used to improve the efficiency and efficacy CIL algorithms. The second is an exploitative method, which can be interpreted as an extreme case of DER (Yan et al., 2021) in terms of architectural design, and is similar to GDumb (Prabhu et al., 2020) in terms of motivation, since it raises significant concerns regarding the current state of CIL research.

We summarize our contributions below:

- We design and conduct analytical experiments to better understand the balance between stability and plasticity in the feature representations of continually learned models.
- We discover that the feature representations of most CIL models are subject to only trivial variations as the model is trained on incremental data. This seems to be a direct result of overweighing the importance of stability over plasticity, and thus, implies a failure to balance stability and plasticity.
- We present two simple CIL algorithms inspired by the results of our analyses. One is an exploit that highlights a major flaw in the current state of CIL research, while the other improves the efficiency and accuracy of an existing algorithm.

2 PRELIMINARIES

To set the stage for our paper, we first describe the task setting, notations, and related works.

2.1 TASK SETTING AND NOTATIONS

In continual learning, a neural network model is trained on data that arrives incrementally. More formally, after first training a model with an initial dataset $\mathcal{D}_0$, additional datasets $\{\mathcal{D}_i\}_{i=1}^N$ arrive in N sequential steps to further update the model. We collectively denote all incremental datasets as $\{\mathcal{D}_i\}_{i=0}^N$ for an N-step setting. In class-incremental learning (CIL), $\mathcal{D}_i$ corresponds to a set of classes, $\mathcal{C}_i$, and collectively, $\{\mathcal{C}_i\}_{i=0}^N$, where all classes are unique such that $|\cup_{i=0}^N \mathcal{C}_i| = \sum_{i=0}^N |\mathcal{C}_i|$. For convenience, we refer to the entire dataset as $\mathcal{D}$, and all classes as $\mathcal{C}$. Furthermore, we note that $\mathcal{D}$ may refer to either the training dataset or the validation dataset, depending on the context.

We denote the model trained on $\mathcal{D}_0$ as $\mathcal{M}_0$, and, by extension, define the set of all models trained incrementally on $\{\mathcal{D}_i\}_{i=1}^N$ as $\{\mathcal{M}_i\}_{i=0}^N$. Note that $\mathcal{M}_i$ $(i > 0)$ is first initialized with the parameters of $\mathcal{M}_{i-1}$, and trained on $\mathcal{D}_i$, and optionally with a small exemplar set sampled from $\{\mathcal{D}_i\}_{i=0}^{i-1}$. In this paper, the model architecture is based on a convolutional neural network, which at any given stage i, is composed of a feature extractor, $\mathcal{F}_i$, and classifier, $\mathcal{G}_i$:

$$\mathcal{M}_i = \mathcal{G}_i \circ \mathcal{F}_i. \tag{1}$$

The classifier typically refers to the single linear layer, although in some CIL algorithms, the classifier is implemented as a cosine classifier[1].

Experimental setting To better analyze feature representations in CIL settings, we conduct all experiments on a large-scale dataset—ImageNet-1K (Russakovsky et al., 2015) (ImageNet, for short)—with a ResNet-18 (He et al., 2016) architecture, which is the most widely-adopted architecture for ImageNet experiments. There exist two common settings for ImageNet: 1) B500-5step setting, where $|\mathcal{C}_0| = 500$, $N = 5$, and $|\mathcal{C}_i| = 100, \forall i > 0$, and 2) B500-10step setting, where $|\mathcal{C}_0| = 500$, $N = 10$,

[1]The details of the cosine classifier are described in the Appendix C.1.

and $|\mathcal{C}_{i>0}| = 50, \forall i > 0$. For both settings, each previously seen class stores 20 exemplars in the memory for subsequent stages. Due to space constraints, we present analyses with the B500-5step setting in our main paper, and B500-10step in the Appendix G.

2.2 Overview of compared methods

Naive The naive method is based on simple fine-tuning, where $\mathcal{M}_{i+1}$ is initialized by a fully trained $\mathcal{M}_i$. Only the cross-entropy loss is used to train the network, and exemplars are sampled randomly from each previously observed class.

iCARL (Rebuffi et al., 2017) Incremental Classifier and Representation Learning (iCARL) employs a simple distillation loss alongside the cross-entropy loss. iCARL also proposes herding for exemplar selection, and discovers that nearest-mean-of-exemplars classification can be beneficial for CIL.

LUCIR (Hou et al., 2019) Learning a Unified Classifier Incrementally via Rebalancing (LUCIR) proposes to use the cosine classifier for feature rebalancing purposes, and alleviates the adverse effects of classifier imbalance by using the cosine between features of the student and teacher models.

AANet (Liu et al., 2021) Adaptive Aggregation Networks (AANet) employs a two branch residual block, where one branch corresponds to a stable (fixed) block, while the other corresponds to a plastic block. The architecture proposed in AANet is a drop-in replacement for existing algorithms, and in this paper, we use AANet + LUCIR. For simplicity, we denote AANet + LUCIR simply as AANet.

SSIL (Ahn et al., 2021) Separated Softmax for Incremental Learning (SSIL) identifies that classification score bias may be caused by data imbalance, and trains the model with a separated softmax output layer, alongside task-wise knowledge distillation.

POD (Douillard et al., 2020) Pooled Outputs Distillation (POD) employs various types of pooling (such as channel, width, height, and GAP pooling) to enforce constraints on feature representations between old and new models.

AFC (Kang et al., 2022) Adaptive Feature Consolidation (AFC) first estimates the importance of each channel in the feature map based on the expected loss increase, and then adaptively restricts the updates to the important channels while leaving non-important channels relatively unconstrained.

DER (Yan et al., 2021) Dynamical Expandable Representations (DER) adds a new feature extractor at each incremental stage, and leaves feature extractors trained on older data fixed while the new feature extractor is updated. For any stage i, the outputs of all $i+1$ feature extractors are concatenated before passing through the classification layer.

3 Re-evaluating continually learned feature representations

The lack of model stability and/or plasticity often leads to weak performance, where a suboptimal feature extractor is unable to extract meaningful features. Likewise, a suboptimal classifier (*e.g.,* due to classifier bias) further exacerbates this issue. While previous works have extensively studied classifier bias (Ahn et al., 2021; Hou et al., 2019; Zhao et al., 2020), the effects of unbalanced stability/plasticity in the feature extractor has been relatively less explored, and will be the focus of this section.

3.1 Finetuning the classifier on full data

We begin by examining how the performance of feature extractors transforms over incremental steps. To eliminate any negative effects of an incrementally trained classifier, we freeze the feature extractors $\{\mathcal{F}_0, ..., \mathcal{F}_N\}$ and train a new classifier for each of the feature extractors on the full ImageNet training data, $\mathcal{D}$. In essence, we assume that the classifier is optimal (well-fitted to any given feature extractor), and evaluate the strength of a feature extractor by using the accuracy on the ImageNet validation set as a proxy measure. While retraining the classifier on $\mathcal{D}$ is a breach of CIL protocol, we emphasize that the goal of this experiment is purely to analyze CIL models from the perspective of feature representations.

From here on out, we let $\mathcal{M}'_j = \mathcal{G}' \circ \mathcal{F}_j$ denote the combination of the feature extractor from stage j and the retrained classifier, $\mathcal{G}'$. In other words, $\mathcal{F}_j$ is trained incrementally on $\{\mathcal{D}_i\}_{i=0}^{j}$, while $\mathcal{G}'$

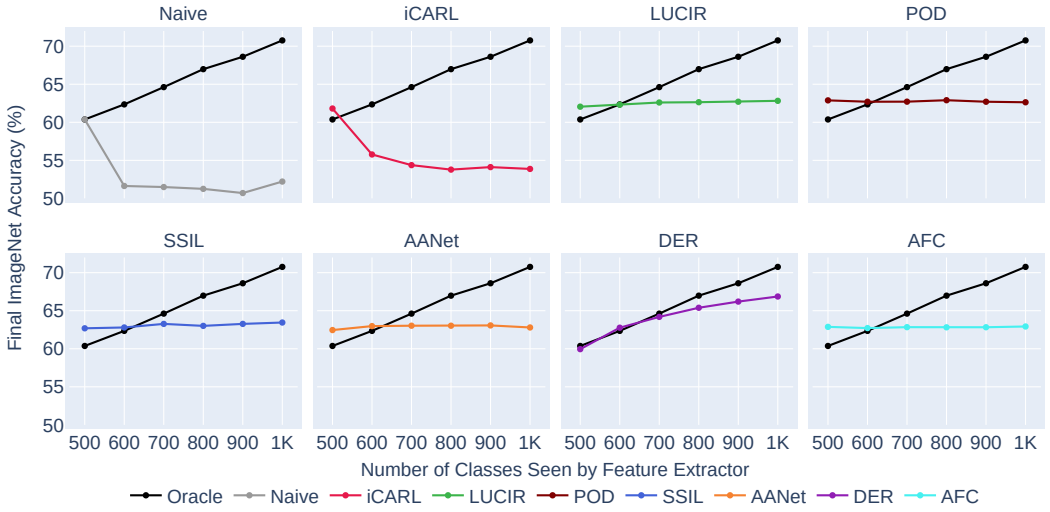

Figure 1: Accuracy on the ImageNet validation set after fine-tuning the classification layer of each incremental model (B500-5step setting) with the full ImageNet training data. The black line indicates an Oracle model trained on {500, 600, ..., 1000} classes, and serves as a point of reference for the performance on non-incremental settings.

replaces the original classifier $\mathcal{G}_j$ and is retrained on the entire dataset $\mathcal{D}$ with a frozen $\mathcal{F}_j$. Note that, while $\mathcal{G}_j$ outputs logits for $\sum_{i=0}^{j}|\mathcal{C}_i|$ classes, $\mathcal{G}'$ outputs $|\mathcal{C}|$-dimensional logits. We then define an accuracy metric, $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$, as the accuracy of model $\mathcal{M}'_j$ on the validation dataset, $\mathcal{D}_i$.

Full ImageNet accuracies Figure 1 illustrates the full validation accuracy on ImageNet, $\text{Acc}(\mathcal{M}'_j, \mathcal{D})$, for all the selected algorithms. Each subplot visualizes the progression of ImageNet accuracy for the specified CIL algorithm, as well as the Oracle model (black line), $\mathcal{M}'_j{}^{\text{Oracle}}$, whose feature extractor is trained on $\cup_{i=0}^{j}\mathcal{D}_i$ all at once, before the classifier is retrained on $\mathcal{D}$. In essence, the feature extractor of $\mathcal{M}'_j{}^{\text{Oracle}}$ represents how an ideal incremental model would perform if important features from previous tasks are not forgotten and new concepts are well learned.

The plots in Figure 1 generally exhibit three distinct trends. First, we observe the plots for the Naive and iCARL models, and notice that the accuracy significantly declines after the initial stage, *i.e.,* $\text{Acc}(\mathcal{M}'_0, \mathcal{D}) > \text{Acc}(\mathcal{M}'_j, \mathcal{D}), \forall j > 0$. These results imply that both the Naive and iCARL models are subject to severe catastrophic forgetting, and their ability to extract useful features deteriorates significantly from the first incremental stage. Next, we investigate the subplots for LUCIR, POD, SSIL, AANet, and AFC. Surprisingly, for these five distinct CIL algorithms, the accuracy remains relatively unchanged across all incremental stages, despite a few minor variations. Unlike the Oracle model, which shows an almost linear increase in accuracy at each incremental step, these five CIL algorithms maintain the same accuracy, *i.e.,* $\text{Acc}(\mathcal{M}'_0, \mathcal{D}) \approx \text{Acc}(\mathcal{M}'_j, \mathcal{D}), \forall j > 0$. Thus, for these 5 algorithms, the feature extractor of $\mathcal{M}'_5$ is not particularly stronger than the feature extractor of $\mathcal{M}'_j$, $\forall j < 5$. Lastly, DER exhibits increasing accuracy with each incremental stage, indicating that the feature extractor does indeed learn new features.

ImageNet subset accuracies Diving deeper, we investigate how the accuracy of each subset in $\mathcal{D}_i$ changes at each incremental stage. In Figure 2, each subplot illustrates the change in $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$, $\forall j$, and a fixed i. For the sake of better visibility, we omit POD and SSIL and present the full plot in Figure A of the Appendix.

First, we focus on the gray curve, which represents the Naive model. We fully expect the Naive model to suffer from catastrophic forgetting, and overfit to the most recently seen set of classes. Indeed, the results corroborate our intuition; in the first plot, $\mathcal{D}_0$, we see that $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_0), \forall j > 0$, performs significantly worse compared to $\text{Acc}(\mathcal{M}'_0, \mathcal{D}_0)$. Furthermore, in the subplots for $\mathcal{D}_1 \sim \mathcal{D}_5$,

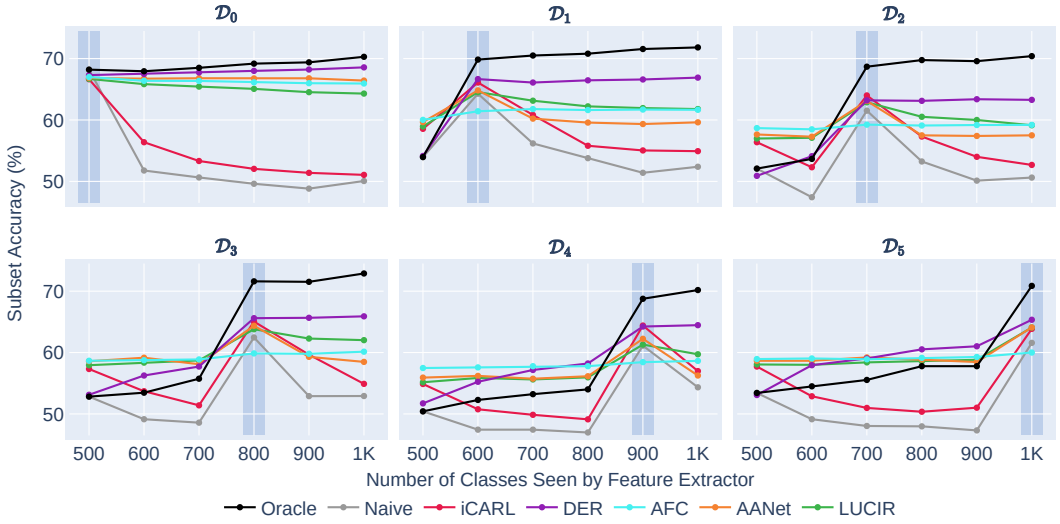

Figure 2: B500-5step subset accuracies. For the sake of visibility, we leave out SSIL and POD from these plots. We highlight the region for model $\mathcal{M}'_j$ in plot $\mathcal{C}_i$, where $i = j$.

we notice that $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$ peaks when $j = i$. Then, $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$ drops off again when $j > i$. These observations all lead to the same conclusion that the naive model suffers from catastrophic forgetting due to the high plasticity of the naive model. A similar pattern is observed for the models trained by iCARL.

Next, we shift our focus to the black curve, which represents the Oracle model. We notice that for $\mathcal{D}_0$, $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_0)$ does not drop, but rather increases as more classes are added, which suggests that knowledge from $\mathcal{C}_i, \forall i > 0$, can in fact be beneficial for performance on $\mathcal{C}_0$. Moreover, $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$ significantly increases when $j = i$. Altogether, the trends exhibited by the Oracle model represent what an ideal CIL model would demonstrate.

Finally, we look at the cyan, orange, and green curves, which represent AFC, AANet, and LUCIR, respectively. For AFC, $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$ is mostly unchanged $\forall i, j$, suggesting that the feature extractor is mostly static across all incremental stages. While this implies that no forgetting occurs, it appears to come at the cost of learning little to no new concepts. Meanwhile, both AANet and LUCIR are mostly stable, but also appear to be slightly more plastic than AFC and POD; they are quick to forget newly learned concepts since $\text{Acc}(\mathcal{M}'_j, \mathcal{D}_i)$ peaks when $j = i$ but drops back down when $j > i$.

4 Are Incrementally Learned Representations Too Static?

Our analysis in Section 3 suggests that a majority of the compared CIL algorithms appear to have high feature stability at the cost of low plasticity. This raises a question: do feature representations remain static across incremental models? In hopes to shed light on this issue, we measure the similarity of intermediate activations and visualize the feature distribution shifts between incremental models.

4.1 Centered Kernel Alignment (CKA)

To analyze intermediate representations of a neural network, we employ Centered Kernel Alignment (CKA) (Kornblith et al., 2019; Cortes et al., 2012), which enables us to quantify the similarity between pairs of neural network representations. CKA has been used to study the effects of increasing depth and width in a network (Nguyen et al., 2021), and to understand how the representations of Vision Transformers (Dosovitskiy et al., 2021) differ from those of convolutional neural networks (Raghu et al., 2021).

Let us consider two arbitrary layers of a neural network with z_1 and z_2 output features, respectively. Given the same set of b inputs, we denote the activation matrices as $\mathbf{X} \in \mathbb{R}^{b \times z_1}$ and $\mathbf{Y} \in \mathbb{R}^{b \times z_2}$. The

$b \times b$ Gram matrices $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ and $\mathbf{L} = \mathbf{Y}\mathbf{Y}^T$ are first centered to obtain $\mathbf{K}'$ and $\mathbf{L}'$, which are then used to compute the Hilbert-Schmidt Independence Criterion (HSIC) (Gretton et al., 2005) as

$$\text{HSIC}(\mathbf{K}, \mathbf{L}) = \frac{\text{vec}(\mathbf{K}') \cdot \text{vec}(\mathbf{L}')}{(b-1)^2}, \tag{2}$$

where $\text{vec}(\cdot)$ denotes the vectorization operation. Finally, CKA normalizes HSIC as follows:

$$\text{CKA}(\mathbf{X}, \mathbf{Y}) = \frac{\text{HSIC}(\mathbf{X}\mathbf{X}^\mathbf{T}, \mathbf{Y}\mathbf{Y}^\mathbf{T})}{\sqrt{\text{HSIC}(\mathbf{X}\mathbf{X}^\mathbf{T}, \mathbf{X}\mathbf{X}^\mathbf{T})\text{HSIC}(\mathbf{Y}\mathbf{Y}^\mathbf{T}, \mathbf{Y}\mathbf{Y}^\mathbf{T})}}. \tag{3}$$

As shown above, CKA is a normalized measure of how similar the $b \times b$ Gram matrices $\mathbf{K}$ and $\mathbf{L}$ are. Given that the Gram matrices themselves reflect the feature relationships among pairs of samples, CKA can be interpreted as a similarity of relationships among the features in $\mathbf{X}$ and $\mathbf{Y}$.

In terms of comparing feature representations, we highlight three properties that make CKA stand out. First, CKA is invariant to permutations in the columns of $\mathbf{X}$ and $\mathbf{Y}$, *i.e.,* $\text{CKA}(\mathbf{X}, \mathbf{Y}) = \text{CKA}(\mathbf{X}\mathbf{P}, \mathbf{Y})$, where $\mathbf{P} \in \{0, 1\}^{z_1 \times z_1}$ is an arbitrary permutation matrix. Second, it is invariant to isotropic scaling of $\mathbf{X}$ and $\mathbf{Y}$. Finally, it can be used to compare activations of layers with different output feature dimensions, *e.g.,* different layers of the same network, or even layers across different architectures. Such advantages make CKA suitable for analyzing the feature representations of class-incremental models, which we outline in the upcoming section.

4.2 Measuring the similarity of incremental feature representations with CKA

Given a pair of incremental feature extractors $\mathcal{F}_0$ and $\mathcal{F}_N$ trained by a select class-incremental algorithm (where $N = 5$ for the B500-5step setting and $N = 10$ for the B500-10step setting), we extract a set of features $\{\mathbf{X}_l\}_{l=0}^{L}$ from $\mathcal{F}_0$, where $\mathbf{X}_l$ denotes feature output of layer l. After extracting a corresponding set of features $\{\mathbf{Y}_l\}_{l=0}^{L}$ from $\mathcal{F}_N$, we compute CKA[1] between $\mathbf{X}_l$ and $\mathbf{Y}_l$, $\forall l \in \{0, \ldots, L\}$, using the ImageNet $\mathcal{D}_0$ validation subset. We extract activations from all convolution, Batch Normalization (Ioffe & Szegedy, 2015), and Residual Block layers of ResNet-18, which results in two sets of features, each with a cardinality of 50 ($L = 49$). A high CKA between $\mathbf{X}_l$ and $\mathbf{Y}_l$, *i.e.,* $\text{CKA}(\mathbf{X}_l, \mathbf{Y}_l) \approx 1$, indicates that the two feature representations are highly similar, and thus, remained static across incremental stages. By extension, high CKA may also serve as a strong indicator for *low forgetting* by virtue of $\mathbf{X}_l$ and $\mathbf{Y}_l$ being highly similar.

In Figure 3 we visualize the same-layer CKA between $\mathcal{F}_0$ and $\mathcal{F}_5$ trained on various CIL algorithms. Each subplot visualizes the layer-wise CKA for two algorithms: 1) the Naive method, which represents a fully-plastic baseline, and 2) the specified CIL algorithm. We first inspect the naive model, which displays relatively high CKA in early layers, but significantly deteriorates in latter layers. This observation is consistent with the notion that early layers learn low-level and widespread features, while higher layers tend to learn more class-specific information (Zeiler & Fergus, 2014). Then, a cursory examination of all other algorithms suggest that all compared CIL algorithms do indeed enforce feature representations to be similar across incremental models, albeit with varying levels of success. For example, both POD and AANet retain high feature similarities across most layers, although there are some significant drops in a few select layers (more details on CKA of AANet provided in Appendix D). Furthermore, we observe that AFC maintains high CKA across all layers, suggesting that the model trained with AFC has very high stability, and thus, undergoes little to no forgetting in each incremental step.

Finally, we present two plots for DER. In the B500-5step setting, DER consists of 6 separate feature extractors, the first of which is identical to $\mathcal{F}_0$. Thus, we plot CKA separately for the fixed feature extractor (DER (0)) and all other feature extractors (DER (1$\sim$5)). As expected, we observe maximum CKA for DER (0), and much lower CKA for DER ($1 \sim 5$).

4.3 t-SNE Visualizations

We further corroborate our observations from Section 4.2 by visualizing the feature shift t-SNE (van der Maaten & Hinton, 2008). We randomly sample 5 classes from $\mathcal{C}_5$ and 20 images from

[1]More precisely, we use the mini-batch version of CKA, which has been shown to converge to the same value of full-batch CKA, described in Eq. (3) (Nguyen et al., 2021). We provide details for mini-batch CKA in the Appendix.

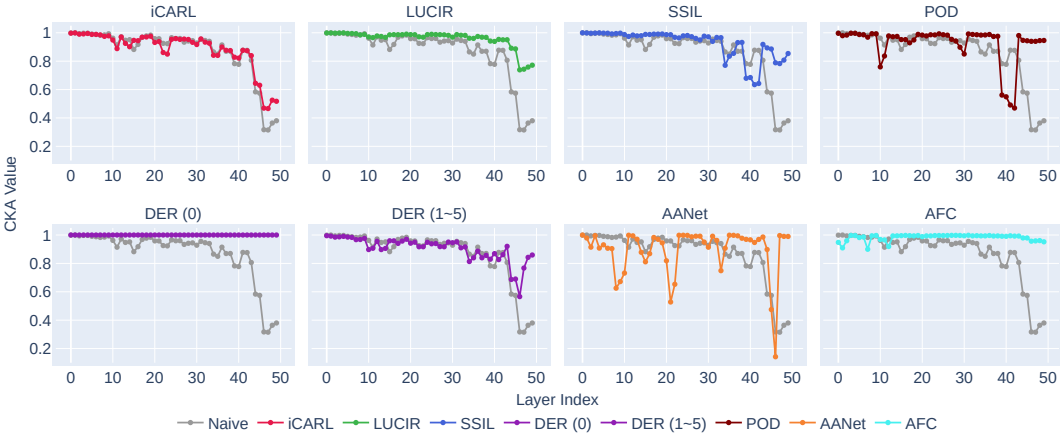

Figure 3: Same-layer CKA values between $\mathcal{F}_0$ and $\mathcal{F}_5$ for incremental models trained with each CIL algorithm on the ImageNet B500 5-step setting. The x-axis spans the layer index of ResNet-18, while the y-axis represents CKA. CKA is evaluated using the $\mathcal{D}_0$ validation set. Each plot is accompanied by the CKA for the naive model, which acts as a point of reference.

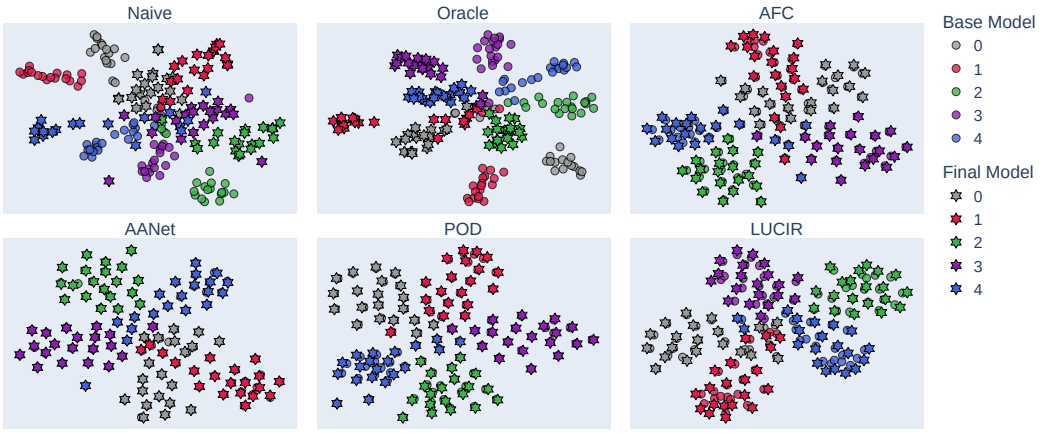

Figure 4: t-SNE visualization of 5 classes of $\mathcal{C}_5$, with 20 samples from each class. Each color represents a distinct class, and features of $\mathcal{F}_0$ and $\mathcal{F}_5$ are depicted with a circle and star, respectively.

each selected class. Then, we compute the feature representations of each image using both $\mathcal{F}_0$ and $\mathcal{F}_5$, and visualize all 200 features in a single t-SNE plot.

Figure 4 shows the t-SNE plots of the Naive and Oracle feature extractors, as well as those of four CIL algorithms that exhibit high stability. In the Naive and Oracle plots, we observe that the same-class features have shifted significantly, since the $\mathcal{F}_0$ and $\mathcal{F}_5$ features are clustered in different regions. This is expected since both models exhibit high plasticity. In the AFC, AANet, POD, and LUCIR plots, however, the feature shift is trivial. Interestingly, most of the $\mathcal{F}_0$ and $\mathcal{F}_5$ features corresponding to the same inputs are overlapped, even though $\mathcal{F}_5$ has been trained on $\{\mathcal{D}_i\}_{i=1}^5$, while $\mathcal{F}_0$ has not. Once again, this suggests that these algorithms excel at maintaining important knowledge from previous tasks, but fail to acquire new knowledge from incremental data.

5 Improving Class-Incremental Learning Based on Observations

We now outline two methods to that aim to demonstrate how the analyses presented in Sections 3 and 4 may be used to improve CIL models.

5.1 PARTIAL-DER

Figures 1 and 2 show that DER not only retains knowledge from old classes, but is also adept at learning concepts from new classes. Such properties of DER are favorable and a step in the right direction for the future of CIL research. DER achieves this by freezing and maintaining $\{\mathcal{F}_j\}_{j=0}^{i-1}$ at the i-th incremental stage, and adding a new fully plastic feature extractor (with the same number of parameters as $\mathcal{F}_0$) for the current set of classes. This allows the feature extractor to fully learn concepts of new classes all the while maintaining knowledge of old classes. However, a major issue of DER is scalability, since for an N-step CIL model, a single forward pass through $\mathcal{F}_N$ requires $N+1$ times computation compared to $\mathcal{F}_0$.

Based on our CKA analysis in Figure 3, we propose a modification of DER, called partial-DER (pDER), which not only makes DER much more efficient, but also improves the overall performance as well. In Section 4.2 we observed that even the Naive model maintains high feature similarity in the lower layers of the network, *i.e.,* the feature representations of lower layers do not change much across $\mathcal{F}_0$ and $\mathcal{F}_5$ even when the model is fully plastic. This suggests that lower layers are inherently stable. Thus, instead of maintaining N full feature extractors for an N-step setting, pDER fixes the lower subset of layers in $\mathcal{F}_0$, and only applies DER on the upper subset of layers. More specifically, we consider all layers upto ResNet's Layer 4 as the lower subset, and apply DER only for Layer 4. We find that this simple modification reduces the GMACs of a forward pass through a DER model by upto 65%[2], while improving $\text{Acc}(\mathcal{M}'_5, \mathcal{D})$ and $\text{Acc}(\mathcal{M}_5, \mathcal{D})$ by 1.5%p and 0.9%p, respectively.

5.2 EXPLOITING STATIC FEATURE EXTRACTORS

The second method, which we name "Exploit", is based on the observation that the feature extractors in most CIL models remain static over the course of incremental stages. In such case, we can significantly improve the training efficiency, achieve strong performance, and eliminate the need for previous-class exemplars by simply freezing the base feature extractor, $\mathcal{F}_0$. This exploit can be interpreted as an extreme case of pDER, where branching occurs in the classification layer.

We first train $\mathcal{M}_0$ on $\mathcal{D}_0$, and fix the feature extractor, $\mathcal{F}_0$, for all subsequent incremental steps. The weight of the Cosine classifier in $\mathcal{M}_0$, $\mathcal{G}_0$, is denoted as $\mathbf{W}_0 \in \mathbb{R}^{F \times |\mathcal{C}_0|}$, where F is the output feature dimension of the feature extractor. For each incremental step i, we train a new weight matrix for the classifier, $\mathbf{W}_i \in \mathbb{R}^{F \times |\mathcal{C}_i|}$, where the cross-entropy loss is only computed for $\mathcal{C}_i$, *i.e.,* the softmax operation only considers the logits for $|\mathcal{C}_i|$ classes. Since $\mathcal{F}_0$ is fixed for all stages, we do not need to update $\mathbf{W}_j, \forall j < i$. After training on $\mathcal{D}_N$, we concatenate the set of weight matrices $\{\mathbf{W}_i\}_{i=0}^{N}$ to produce a single weight matrix, $\mathbf{W} \in \mathbb{R}^{F \times |\mathcal{C}|}$. Finally, we compose $\mathcal{G}$ (parametrized by weight $\mathbf{W}$) with $\mathcal{F}_0$ to obtain the final model: $\mathcal{M}_N = \mathcal{G} \circ \mathcal{F}_0$.

Compared to CIL algorithms that usually train $\mathcal{M}$ for $90 \sim 120$ epochs on each $\mathcal{D}_i$, our exploit only requires around 10 epochs of training to converge for each incremental stage. Furthermore, we only need to compute gradients for $\mathcal{G}$, which reduces computational burden of training. These two factors make training extremely fast compared to traditional algorithms that tune the entire model and require many epochs of training. Despite requiring only a fraction of the computation, our exploit achieves final ImageNet accuracy of 60.5%, and an average incremental accuracy of 67.2%, which is competitive against the methods compared in this paper.

6 DISCUSSIONS

Based on the plots in Figure 1, we define the following metric:

$$\Delta\mathcal{M}'_i := \text{Acc}(\mathcal{M}'_i, \mathcal{D}) - \text{Acc}(\mathcal{M}'_0, \mathcal{D}) \tag{4}$$

$\Delta\mathcal{M}'_i$ measures the relative performance of improvement of $\mathcal{F}_i$ over $\mathcal{F}_0$, *i.e.,* how much the feature extractor improves as training classes are added. A high positive $\Delta\mathcal{M}'_i$ indicates that the feature extractor is able to learn new concepts incrementally. On the other hand, $\Delta\mathcal{M}'_i \simeq 0$ indicates that the feature extractor is stable, but not plastic enough to acquire new knowledge. Finally, a large

[2] $\mathcal{F}_5$ of DER requires 10.9 GMACs for one forward pass of batch size 1, while $\mathcal{F}_5$ of pDER requires 3.9GMACs. More details provided in Appendix A

Table 1: Summary of all compared CIL algorithms. *Methods introduced in Section 5

Method	Naive	iCARL	LUCIR	SSIL	POD	AANet	AFC	DER	Exploit*	pDER*	Oracle
$\text{Acc}(\mathcal{M}'_0, \mathcal{D})$	60.4	61.8	62.1	62.5	62.9	62.5	62.9	60.0	62.9	60.0	60.4
$\text{Acc}(\mathcal{M}'_5, \mathcal{D})$	52.2	53.9	62.8	63.5	62.6	62.8	62.9	66.9	62.9	68.4	70.8
$\Delta\mathcal{M}'_5$	-8.2	-7.9	0.7	1.0	-0.3	0.3	0.0	6.9	0.0	8.4	10.4

Table 2: Average incremental accuracy and Final ImageNet accuracy of all compared CIL algorithms. †SSIL uses a different class ordering than all other methods; thus, inter-method comparisons with SSIL may not be appropriate. *Methods introduced in Section 5

Method	Naive	iCARL	LUCIR	SSIL†	POD	AANet	AFC	DER	Exploit*	pDER*	Oracle
Avg. Inc. Acc.	41.4	31.5	55.6	65.8	66.3	57.1	66.4	69.1	67.2	69.7	72.8
$\text{Acc}(\mathcal{M}_5, \mathcal{D})$	31.5	17.2	41.0	59.8	58.6	43.3	59.0	63.8	60.5	64.7	70.8

negative $\Delta\mathcal{M}'_i$ represents severe catastrophic forgetting in the feature extractor. Ultimately, CIL models should strive to maximize $\Delta\mathcal{M}'_i$ in order to facilitate better feature representation learning.

Are we learning continually? Table 1 quantitatively summarizes all the compared CIL algorithms in terms of $\text{Acc}(\mathcal{M}'_0, \mathcal{D})$, $\text{Acc}(\mathcal{M}'_5, \mathcal{D})$, $\Delta\mathcal{M}'_5$. Looking at LUCIR, SSIL, POD, AANet, and AFC in Table 1, we notice that there is almost no difference between $\text{Acc}(\mathcal{M}'_0, \mathcal{D})$ and $\text{Acc}(\mathcal{M}'_5, \mathcal{D})$, which all lie in the range of $62\% \sim 63\%$. Furthermore, there exists a significant gap between $\text{Acc}(\mathcal{M}'_5, \mathcal{D})$ of the aforementioned algorithms and $\text{Acc}(\mathcal{M}'_5, \mathcal{D}) = 70.8\%$ of the Oracle model. This implies that, while these methods do mitigate catastrophic forgetting (to varying degrees), they suffer from a lack of plasticity, *i.e.,* are unable to learn continually. Moreover, these results also imply that if feature representations do not accumulate new knowledge through incremental data, the highest achievable performance is still 7∼8%p shy of the Oracle model (even with an optimal classifier). Thus, the plasticity of feature representations is an absolutely crucial aspect of incremental learning.

Limitations of traditional metrics In Table 2 we present the widely adopted metrics in CIL: 1) average incremental accuracy of all compared CIL algorithms on the B500-5step setting, and 2) the final accuracy on ImageNet, $\text{Acc}(\mathcal{M}_5, \mathcal{D})$. In particular, we note that our Exploit in Table 2 outperforms all but one (DER) CIL algorithms on both metrics, despite leaving the feature extractor $\mathcal{F}_0$ fixed. Furthermore, LUCIR and AANet demonstrate weak performance here, as their average incremental and final ImageNet accuracies are roughly 10%p and 20%p lower than POD, AFC, DER, SSIL, and Exploit. However, according to Table 1, all the aforementioned CIL algorithms demonstrate similar $\text{Acc}(\mathcal{M}'_0, \mathcal{D})$ and $\text{Acc}(\mathcal{M}'_5, \mathcal{D})$ scores. This suggests that while the feature representations of all compared methods (excluding Naive, iCARL, and DER) all have similar levels of discriminativeness, it is not well expressed in terms of average incremental accuracy nor $\text{Acc}(\mathcal{M}_5, \mathcal{D})$. Clearly, high average incremental accuracy and final (ImageNet) accuracy are not really indicative of how much the model has *learned* continually; yet, these metrics have become the de-facto standard in CIL research. This should be alarming, for both researchers and practitioners — ambiguous metrics deliver a false illusion of progress, and may lead researchers to develop algorithms that seem to outperform other state-of-the-art algorithms, but are completely misaligned with the motivation behind continual learning (such as our Exploit). Thus, we hope that the analyses in our work will facilitate better evaluation of CIL algorithms, and inspire researchers to focus more on stronger feature representation learning of incremental models.

7 Conclusion

We took a deep dive into how effectively modern CIL algorithms address the stability-plasticity dilemma. We introduced evaluation protocols that help us better understand the stability and plasticity of feature representations. Our evaluations of recent works showed that many CIL methods are too fixated on the notion of alleviating catastrophic forgetting, to the extent that the feature extractor rarely learns any new concepts after the initial stage of training (on $\mathcal{D}_0$). Based on this observation, we introduced two simple algorithms that improve upon an existing algorithm, and exploit the shortcomings of the standard evaluation metrics for CIL research. All in all, we hope that our findings will propel CIL research to focus more on stronger continual learning of feature representations.

Ethics statement Fairness in AI systems has attracted a lot of attention in recent years, and has been fueling progressive research in the field of debiasing. A potential ethical concern of class-incremental learning is that the model is prone to learning biases for the most recent set of classes. In fact, classifier bias is a widely known issue in the class-incremental setting (of the compared CIL algorithms in our paper, SSIL and LUCIR explicity aim to reduce classifier bias). Since our work focuses on the feature representations, it does not help to address the potentially harmful aspects of classifier bias in CIL models.

Reproducibility statement The codes for baseline models reported in our paper are all open-sourced by authors of previous works. Please refer to Appendix C.2 for more implementation details of each baseline model, as well as links to the open-source codes.

REFERENCES

Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. SS-IL: separated softmax for incremental learning. In *ICCV*, 2021.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Mach. Learn.*, 2010.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *JMLR*, 2012.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

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

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, 2015.

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic Learning Theory*, 2005.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a Unified Classifier Incrementally via Rebalancing. In *CVPR*, 2019.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 2015.

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

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

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*, 2017.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019.

Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive Aggregation Networks for Class-Incremental Learning. In *CVPR*, 2021.

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. URL https://openreview.net/forum?id=KJNcAkY8tY4.

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. In *ICLR*, 2015.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. In *ECCV*, 2020.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In *NeurIPS*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/652cf38361a209088302ba2b8b7f51e0-Paper.pdf.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental Classifier and Representation Learning. In *CVPR*, 2017.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NIPS*, 2017.

Shipeng Yan, Jiangwei Xie, and Xuming He. DER: Dynamically Expandable Representation for Class Incremental Learning. In *CVPR*, 2021.

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.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Bowen Zhao, Xi Xiao, Guojun Gan, Bin Zhang, and Shu-Tao Xia. Maintaining Discrimination and Fairness in Class incremental Learning. In *CVPR*, 2020.

A 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**

A.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.

A.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.

B 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.,* $|\mathcal{D}_0| = 25\text{K}$ 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 Nguyen et al. (2021). The main difference is that the mini-batch CKA uses an unbiased estimator of HSIC:

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

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:

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

where k denotes the number of iterations. Following Nguyen et al. (2021), we use a batch size of $n = 256$ and iterate over $\mathcal{D}_0$ 10 times to compute mini-batch CKA.

C More implementation details

C.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.

C.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 (Douillard et al., 2020). 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 $\text{lr} = 0.1$ 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 (Douillard et al., 2020; Kang et al., 2022), 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) (Joseph et al., 2022) 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.

[1]POD: `https://github.com/arthurdouillard/incremental_learning.pytorch`
[2]AFC: `https://github.com/kminsoo/AFC`
[3]ELI: `https://github.com/JosephKJ/ELI`

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.

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.

D 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.

E GDUMB

Table B: Retrained classifier performance for GDumb (with CutMix (Yun et al., 2019)).

$\text{Acc}(\mathcal{M}'_0, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_1, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_2, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_3, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_4, \mathcal{D})$	$\text{Acc}(\mathcal{M}'_5, \mathcal{D})$
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 B 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 (Yun et al., 2019) for GDumb, which is not used for any of the other compared methods.

F FULL 5 STEP SUBSET ACCURACIES

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

[4]DER: `https://github.com/Rhyssiyan/DER-ClassIL.pytorch`

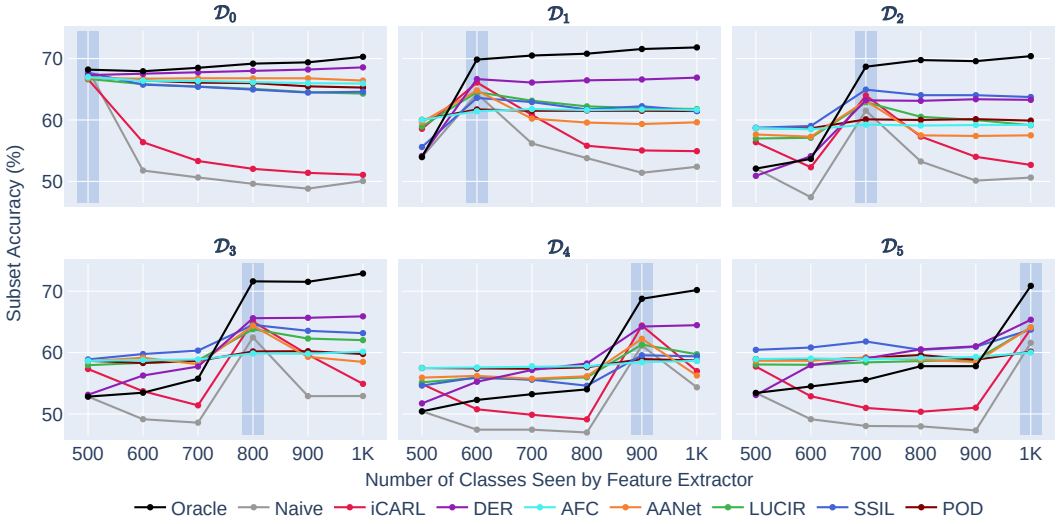

Figure A: B500-10step subset accuracies for all methods. Note that SSIL uses a different class ordering. We highlight the region for model $\mathcal{M}'_j$ in plot $\mathcal{C}_i$, where $i = j$.

G 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.

For ease of viewing, we also include HTML files for each figure in the main paper and appendix. These HTML files are interactive, meaning that viewers can inspect individual points, filter plots by algorithm (by clicking or double-clicking the corresponding label in the legend), and rescale the axis as needed.

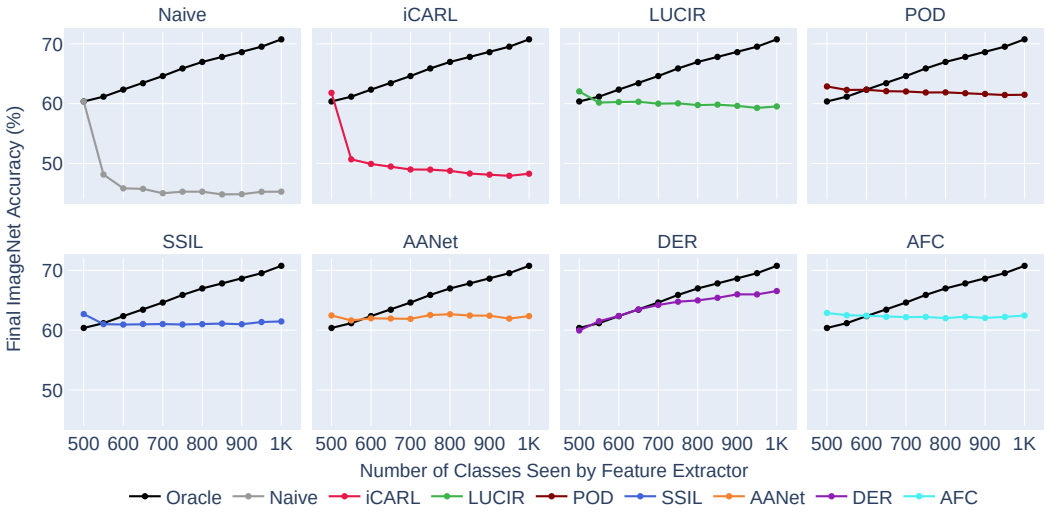

Figure B: Accuracy on the ImageNet validation set after fine-tuning the classification layer of each incremental model (B500-10step setting) with the full ImageNet training data. The black line indicates an oracle model trained on {500, 550, 600, ..., 1000} classes, and serves as a point of reference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

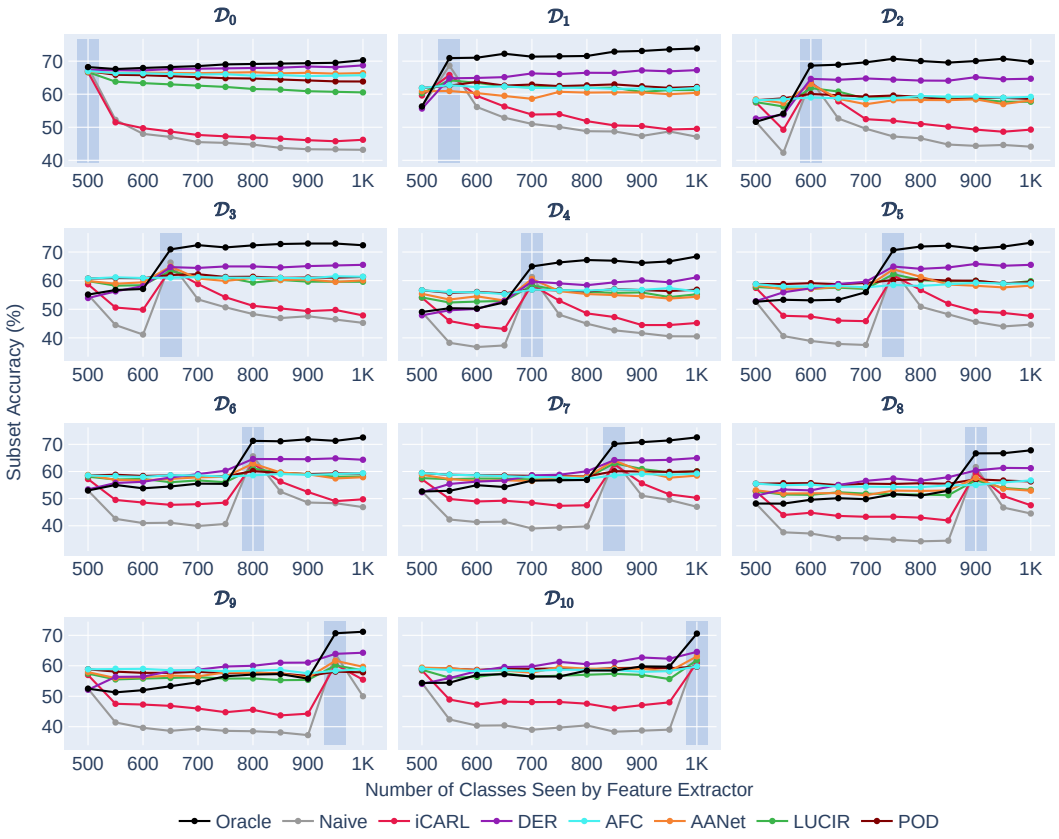

Figure C: B500-10step subset accuracies for all methods. Note that SSIL uses a different class ordering. We highlight the region for model $\mathcal{M}'_j$ in plot $\mathcal{C}_i$, where $i = j$.

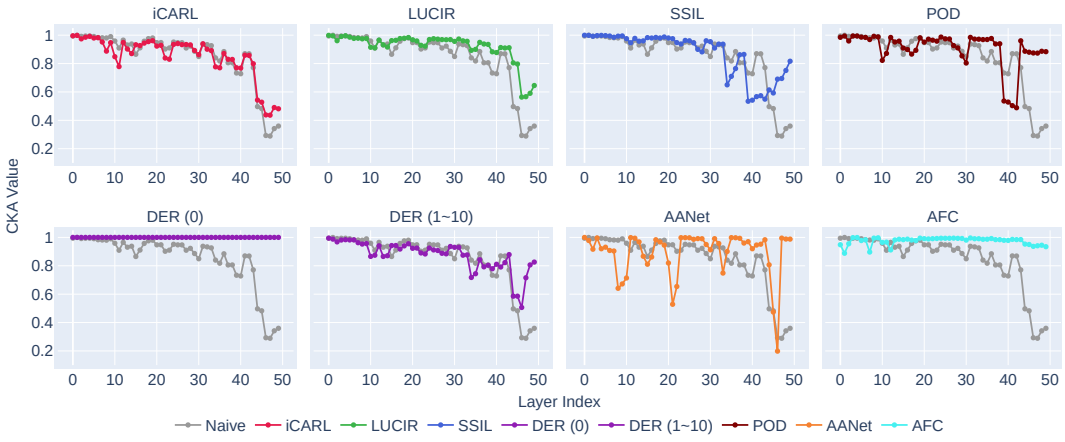

Figure D: Same-layer CKA values between $\mathcal{M}_0$ and $\mathcal{M}_{10}$ for incremental models trained with each CIL algorithm on the ImageNet B500 10-step setting. The x-axis spans the layer index of ResNet-18, while the y-axis represents CKA. CKA is evaluated using the $\mathcal{D}_0$ validation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