Continual Learning with Informative Samples: An Empirical Evaluation of Coreset Strategies

Elif Ceren Gok Yildirim[a], Murat Onur Yildirim[a], Joaquin Vanschoren[a]

[a]*Mathematics and Computer Science, Eindhoven University of Technology, Groene Loper 3, Eindhoven, 5612AZ, , The Netherlands*

Abstract

Continual Learning (CL) methods usually learn from all the available data. However, this is not the case in human cognition which efficiently focuses on key experiences while disregarding the redundant information. Similarly, not all data points in a dataset have equal potential; some can be more informative than others. Especially in CL, such redundant or low-quality data can be detrimental for learning efficiency and exacerbate catastrophic forgetting. Drawing inspiration from this, we explore the potential of learning from important samples and present an empirical study for evaluating coreset selection techniques in the context of CL to stimulate research in this unexplored area. We train various continual learners on progressively larger subsets of selected samples, analyzing the learning-forgetting dynamics and uncovering the mechanisms that enhance the stability-plasticity trade-off. We present several significant observations: Learning from selectively chosen samples (i) enhances incremental accuracy, (ii) improves knowledge retention of previous tasks, and (iii) continually refines learned representations. This analysis contributes to a deeper understanding of data-selective learning strategies in CL scenarios. The code is available at `https://github.com/ElifCerenGokYildirim/Coreset-CL`.

Keywords: continual learning, catastrophic forgetting, coreset selection, data-centric ai

1. Introduction

Machine learning has achieved remarkable success in solving complex tasks, often relying on the assumption that data is available in a static and complete form. While effective in controlled scenarios, this approach falls short in dynamic environments where data and tasks evolve over time. Continual Learning (CL) bridges this gap by enabling models to learn sequentially from streaming data, retaining previously acquired knowledge while adapting to new tasks, a duality known as the stability-plasticity balance. This balance is critical for mimicking human-like learning, where accumulated knowledge is preserved (stability) yet flexibly updated with novel experiences (plasticity). However, catastrophic forgetting [1] remains a challenge in this dynamic setting wherein models tend to lose acquired knowledge from previous tasks, upon learning new ones.

Recent research has brought solutions through various techniques, including regularization-based [2, 3, 4], replay-based [5, 6, 7, 8], architecture-based [9, 10, 11, 12, 13] and prompt-based [14, 15, 16] approaches. While these methods improve performance, they share a common assumption: all training samples are equally valuable and must be exhaustively utilized. By default, this standardized practice prioritizes plasticity (integrating new information) at the risk of destabilizing learned representations, as redundant or noisy samples may overwrite critical prior knowledge. This 'learn-it-all' paradigm diverges from human learning efficiency since, as humans, we are initially exposed to vast amounts of information but intuitively filter and prioritize them, focusing on key experiences (e.g. clear and novel examples) that enrich our understanding while disregarding redundant details [17, 18, 19].

Drawing inspiration from this human cognitive ability, we present an empirical study to evaluate the stability-plasticity dynamics of training with important samples across different CL models (see Figure 1) in the most challenging scenario class-incremental learning (CIL) where the learner must predict outcomes for all encountered classes without task identifiers [20]. We believe that this comprehensive study presents the potential benefits of data-centric strategies in CL scenarios and stimulates future research in this direction. Our contributions can be summarized as:

I. This paper presents the first explicit empirical analysis of different coreset selection methods in combination with various continual learners in the class-incremental learning setting.

II. We find that training on carefully selected subsets derived from various coreset selection strategies substantially enhances incremental accuracy.

III. We demonstrate that the increase in performance among continual learners trained with selected samples is primarily driven by an improved stability-plasticity trade-off, which is largely attributable to the enhanced retention of prior knowledge.

IV. We advocate that continual learning can benefit from a data-centric approach, despite the fact that most existing research has predominantly focused on model-centric enhancements.

2. Background

2.1. Continual Learning

Continual learning can be categorized into four main groups [20] as regularization-, replay-, architecture-, and prompt-based approaches. Regularization-based methods regularize the abrupt changes in the learned parameters to prevent catastrophic forgetting [2, 3, 4]. Replay-based methods either retain selected exemplars from prior tasks or generate a subset of data points from previous tasks to alleviate forgetting [5, 6, 7, 8]. Architecture-based methods prevent forgetting by increasing model size and allocating distinct sets of parameters to individual tasks, ensuring there is no overlap between them [9, 10, 11, 12, 13]. Finally, prompt-based methods that influence the self-attention process of large pre-trained models have also received growing popularity [14, 15, 16].

2.2. Summary of CL Methods Selected for Analysis

We use 7 well-established CL models that encompass various approaches including architecture-based, replay-based, regularization-based, and prompt-based. We deliberately chose these methods to provide a comprehensive analysis since they all represent different learning strategies. For more details, please see our Appendix A.1.

DER-Architecture. Dynamically expandable representation [9] creates a new feature extractor for each task and then aggregates the features from all backbones on a single classifier. Each expanded feature extractor or backbone uses an additional auxiliary loss to differentiate better between old and new classes. Facing new tasks, it freezes the old backbone to maintain former knowledge.

FOSTER-Architecture. Feature boosting and compression [10] frames the learning process as a feature-boosting problem and aims to enhance the learning of new features. Then, it expands the continual learner on a single classifier by integrating the boosted features with a compression step to ensure that only relevant features are retained.

MEMO-Architecture. Memory efficient expandable model [11] expands the network in a more efficient way. It assumes that the initial blocks of the backbone capture the general patterns for any task and only expands the model in the last or specialized blocks that are designed to be task-specific.

iCaRL-Replay. Incremental classifier and representation learning [21] is a replay-based method that stores samples from each learned task. Upon the arrival of a new task, it uses stored exemplars together with the new one to capture the distribution at once. Therefore, it refines the features after each task with additional distillation loss to overcome abrupt shifts in the feature space.

ER-Replay. Experience replay [22] is a simple yet strong method that employs reservoir sampling to store samples from each task and randomly retrieves stored samples with the new task to capture the distribution all at once.

LwF-Regularization. Learning without forgetting [3] is solely a regularization-based method without relying on any replay buffer. It utilizes a distillation loss to prevent sudden changes in the feature space while learning new tasks.

CODA-Prompt. Continual decomposed attention-based prompting [16] is a prompt-based method that leverages pre-trained Vision Transformers (ViT) without relying on data rehearsal. It introduces a set of prompt components that are dynamically assembled based on input-conditioned weights, generating task-specific prompts for the transformer's attention layers. These generated prompts selectively guide the model's attention to relevant features for each task, to enable better stability-plasticity tradeoff.

2.3. Coreset Selection

Coreset selection approximates the distribution of the whole dataset with a small subset. Several works offer a strong theoretical motivation and prove that such subsets can closely mimic key properties of the full data by preserving diversity and maintaining representative feature coverage [23, 24, 25]. It is especially extensively examined in data-efficient supervised batch learning [26, 27, 28, 29, 30, 31, 23, 32] and active learning [33, 34]. Recently, it has also been shown that coreset selection holds promise in continual learning to construct a memory buffer from important samples [7, 8]. For example, an inspiring study [35] improved the performance in online CL setup by introducing a coreset selection method to select the most diverse samples while approximating the mean of a given batch.

However, the interplay between the coreset selection methods and continual learning models remains largely unexplored. Exploring this interaction could provide novel insight to create more efficient and advanced continual learners.

2.4. Overview of Coreset Algorithms Selected for Analysis

We employ 4 distinct coreset selection methods as well as a baseline using random selection. Once again, we carefully chose these distinct methods to offer comprehensive empirical analysis. It is important to note that these coreset selection methods require a brief initial training or warm-up phase to make informed and meaningful decisions when selecting coreset samples. We provide more details in our Appendix A.2.

Random. This selection strategy involves randomly selecting a subset of data points from the entire dataset without any specific criteria or consideration of their importance or informativeness.

Herding. Herding [30] chooses data points by evaluating the distance between the center of the original dataset and the center of the coreset within the feature space. This algorithm progressively and greedily includes one sample at a time into the coreset, aiming to minimize the distance between centers.

Uncertainty. Samples with lower confidence levels might have a stronger influence than those with higher confidence levels, thus having these samples in the coreset can be useful. Least confidence, entropy, and margin are the common metrics used to quantify sample uncertainty [31]. In this study, entropy is used as a selection metric.

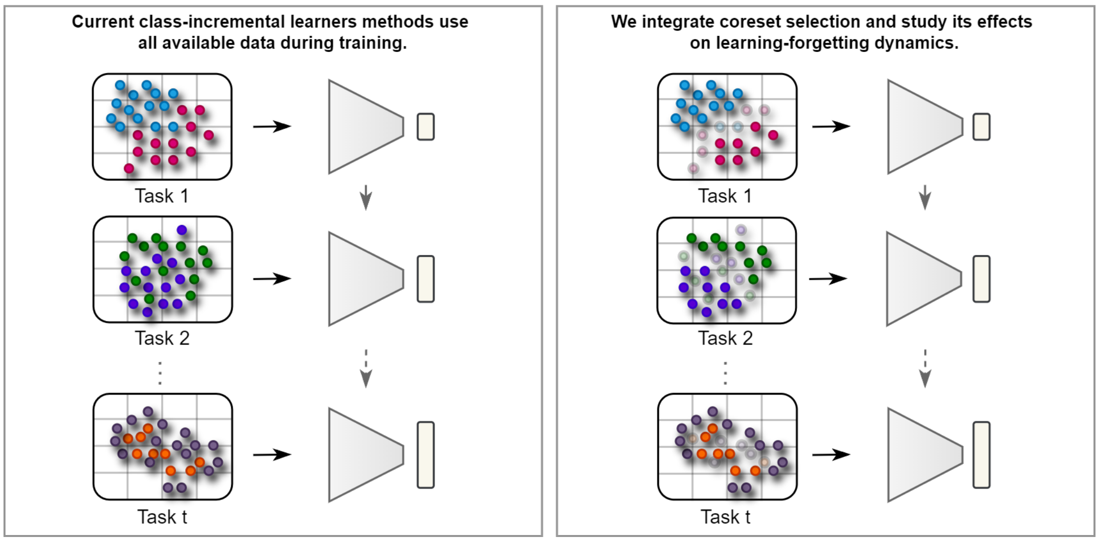

Current class-incremental learners methods use all available data during training.
Task 1
Task 2
Task t
We integrate coreset selection and study its effects on learning-forgetting dynamics.
Task 1
Task 2
Task t

Figure 1: Illustration of our evaluation protocol: Existing continual learning methods (**left**) typically utilize all available samples indiscriminately during training. In this study (**right**), we subject continual learners to a selection of the most important samples with different coreset selection methods and analyze affects on the incremental performance.

Forgetting. Forgetting selects instances which were correctly classified in one epoch and then subsequently misclassified in the following epoch during training [26]. This method provides valuable insight into the intrinsic characteristics of the training data and removes challenging or forgettable instances.

GraphCut. GraphCut partitions the dataset into subsets based on dissimilarity or information content, and data points from these subsets are then selected to form the coreset [23]. It ensures that the coreset captures the diversity and essential information of the original dataset while reducing redundancy.

3. Empirical Method

We conduct a comprehensive evaluation of existing CL methods, assessing their performance when trained on carefully selected, informative samples, as opposed to the traditional approach of full dataset training. To clarify our approach, we first present the necessary preliminaries and problem formulation in Section 3.1. Following this, we define our objective and outline the proposed training strategy in Section 3.2.

3.1. Preliminaries and Problem Formulation

Formally, we define the CL problem as a sequence of classification tasks $T_{1:t} = (T_1, T_2, ..., T_t)$. Each task T_t is drawn from an unknown distribution and consists of input pairs $(x_{i,t}, y_{i,t}) \in X_t \times Y_t$ where $x_{i,t}$ represents the sample and $y_{i,t}$ indicates the corresponding label. Note that these learning tasks are mutually exclusive, meaning that the label sets do not overlap, i.e., $Y_{t-1} \cap Y_t = \emptyset$.

From the coreset selection perspective, the objective is to identify a compact yet informative subset S_t from a given task T_t with a large number of input pairs $(x_{i,t}, y_{i,t})$. The selected S_t should preserve the essential information of the full task T_t. Therefore, model trained with subset $S_t \subset T_t$ with a condition of $|S_t| < |T_t|$ should have a similar performance compared to a model trained with T_t.

3.2. Objective and Training Strategy

We structure the training process into two distinct phases: the warm-up phase and the learning phase. This is necessary because coreset selection methods rely on analyzing the model's behavior and data representations to identify the most informative samples. Consequently, the CL model must undergo at least partial training during the warm-up phase to enable accurate sample selection. It is important to note that the duration of the warm-up phase is typically much shorter than that of the learning phase. Upon completion of the warm-up phase, the learning phase proceeds with the selected subset of samples.

Let $f_\theta(\cdot)$ denote the continual learning model with parameters θ. Then, the training process can then be expressed as follows:

$$f_{\theta^*} = \arg\min_\theta \mathcal{L}_{CL}(f_\theta, S_t, (1-\alpha)\mathrm{e}) \circ \arg\min_\theta \mathcal{L}_{CE}(f_\theta, T_t, \alpha\mathrm{e}) \tag{1}$$

Here, the second term $(f_\theta, T_t, \alpha\mathrm{e})$ represents training the model f_θ on the full training samples of task T_t with a defined time budget of $\alpha\mathrm{e}$ where hyperparameter $\alpha \in (0, 1)$ and determines the fraction of the total training budget allocated to the warm-up phase, and e is the total number of epochs available for training. Similarly, the first term $(f_\theta, S_t, (1-\alpha)\mathrm{e})$ represents the training of the model f_θ, for the remaining time budget $(1-\alpha)\mathrm{e}$, on the coreset S_t which is selected from T_t with a fraction of $s \in (0, 1)$ based on a coreset selection function $\phi(\cdot)$, so that $|S_t| = s \cdot |T_t|$. Note that $\mathcal{L}_{CE}$ represents Cross-Entropy loss and $\mathcal{L}_{CL}$ represents the loss defined by continual learning methods given in section 2.2.

To provide a more precise explanation, Algorithm 1 begins with a warm-up phase (lines 2-7) where the model f_θ observes the training samples T_t of the current task for a duration of αe. During this phase, the model trains each batch b to compute the Cross-Entropy loss $\mathcal{L}_{CE}(f_\theta, b)$. This initial exposure allows the model to capture a broad understanding of the task's characteristics.

Algorithm 1 CL Training with Coreset Samples

```
Require: Model $f_\theta$, Tasks $T_{1:t}$ with training sets $T_t$, learning rate $\eta$, total epochs e, warm-up fraction $\alpha$, coreset selection function $\phi$, coreset fraction $s$
 1: for task t = 1 to $T_t$ do
 2:     for epoch = 1 to $\lfloor \alpha e \rfloor$ do                ▷ Warm-up Phase
 3:         for each batch $b$ in $T_t$ do
 4:             Compute $\mathcal{L}_{CE}(f_\theta, b)$
 5:             Update $f_\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{CE}$
 6:         end for
 7:     end for
 8:     Use $\phi(f_\theta, T_t)$ to select $S_t \subset T_t$ with a fraction of $s$
 9:     for epoch = 1 to $\lfloor (1-\alpha)e \rfloor$ do          ▷ Learning Phase
10:         for each batch $b$ in $S_t$ do
11:             Compute $\mathcal{L}_{\text{CL}}(f_\theta, b)$
12:             Update $f_\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{CL}$
13:         end for
14:     end for
15: end for
```

Following the warm-up (line 8), the algorithm employs the coreset selection function $\phi(\cdot)$ which requires training samples for a given task T_t and the model f_θ to filter down to a coreset $S_t \subset T_t$, consisting of only a fraction s of the current task samples. The criterion for selection, depending on the coreset selection function, can target samples with high informativeness, uncertainty, or relevance, focusing on key data points.

In the learning phase (lines 9-14), which spans the remaining $(1-\alpha)$e epochs, the model is trained on batches from S_t, using specific loss function of continual learners $\mathcal{L}_{\text{CL}}(f_\theta, b)$. This refines the goal of solidifying task-specific knowledge while minimizing interference from previous tasks to prevent forgetting.

4. Experimental Setting

Datasets. We use well-established continual learning datasets, specifically **Split-CIFAR10** [36], **Split-CIFAR100** [36], and **Split-ImageNet-100** [37] in our experiments to evaluate and posit our findings. **Split-CIFAR10** has 5 disjoint tasks and each task has 2 disjoint classes with 10000 samples for training and 2000 samples for testing. **Split-CIFAR100** has 10 disjoint tasks and each task has 10 disjoint classes with 5000 samples for training and 1000 samples for testing. In addition, we employ **Split-ImageNet100**, a subset of the large-scale ImageNet dataset, with images at a higher resolution of 224x224 pixels. Similar to Split-CIFAR100, Split-ImageNet100 is divided into 10 tasks, each consisting of 10 disjoint classes. The increased number of classes, fewer images per class combined with longer learning sessions, and higher resolution bring further challenges and offer a more complex scenario.

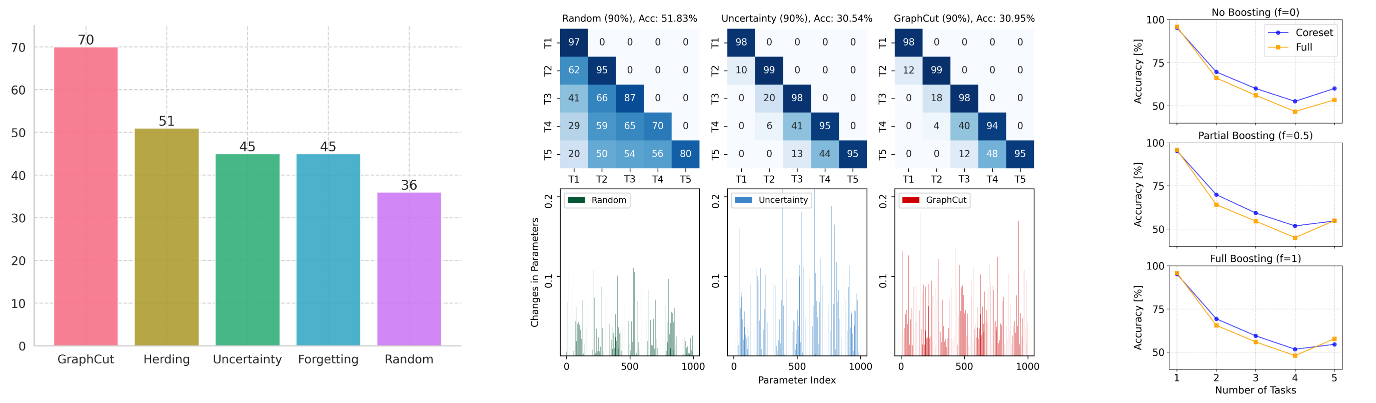
Implementation Details. We use Deepcore [27] for coreset selection methods and PYCIL [38] for the CL models. We use both from scratch (ResNet18) and pre-trained (ResNet18 and ViT) models with prior knowledge to provide a more comprehensive analysis, using standard CL metrics, which are discussed more in detail in the Appendix A.3. We set the total training budget e to 100 epochs where warmup fraction α is set to 0.1 and the remaining is allocated for the learning phase. To ensure a comprehensive evaluation across varying data availability scenarios, we set coreset fraction s to $10\%, 20\%, 50\%, 80\%$ and 90% for each task ranging from limited samples to near-full data access. We use SGD optimizer with a scheduled learning rate of 0.1 and momentum of 0.9. We set a weight decay of 5×10^{-4} for the initial task and 2×10^{-4} for subsequent tasks. We set the batch size to 128. We employ a fixed memory size: 50 per class for CIFAR10 and 20 per class for CIFAR100 and ImageNet100. This configuration follows the default setting in the PYCIL framework, which we adopt in our implementation. Note that, we do not employ coreset selection methods to construct the memory buffers, adhering instead to the original implementations. This decision was made to ensure a fair and controlled comparison, isolating the effect of coreset selection during training without modifying the rehearsal strategies themselves. For ViT, we adjust the learning rate to 1×10^{-3}, reduce the batch size to 32, and train for 20 epochs. We run all experiments on A100 GPU across distinct random seeds, where each seed governs not only parameter initialization but also task ordering to ensure that our findings are robust by reflecting performance across diverse task sequences.

5. Results and Analysis

In Section 5.1, we conduct a comprehensive analysis across diverse CL methods and different coreset algorithms with varying coreset sizes. In Section 5.2, we investigate why coreset selection improves incremental accuracy, offering insight into the stability-plasticity dynamics. In Section 5.3, we explore how these dynamics are reflected in the model's decision-making. Finally, in Section 5.4, we further examine and discuss the warm-up training, computational complexity as well as model complexity, and their effects.

5.1. Impact of Coreset Selection on Incremental Performance

Our analysis reveals that coreset selection improves incremental learning performance across different datasets and architectures (Table 1, 2, A). To quantify the effectiveness of coreset algorithms, we compare how often each selection method outperforms full training across all CL methods and datasets (Figure 2a). We observe that random sampling serves as a strong baseline while structured coreset methods are essential for maximizing incremental accuracy which is heavily influenced by the size of the coreset: When the number of samples per task is large enough (e.g., CIFAR10), the model can afford to remove a greater portion of the data without significant loss in incremental accuracy (Table 1). On the other hand, when sample sizes are limited (e.g., CIFAR100 and ImageNet100), removing a large portion of the data is more challenging, as the model may struggle to obtain essential knowledge with fewer examples (Table 2, B). Nevertheless, regardless of the dataset size, more distinct and compact representations obtained via coreset lead to increased performance due to better knowledge retention. We further explore the impact of coreset size and its effect on representational dynamics in Section 5.3.



(a) Coreset outperformance over full-sample training

(b) Parameter changes in LwF using different coreset methods

(c) Effect of boosting in FOSTER

Figure 2: (a) Coreset methods frequently outperform full-sample training, (b) LwF trained on Split-CIFAR10 demonstrates less forgetting in random selection due to fewer parameter changes and (c) Lower boosting strength improves coreset training performance over full-sample since boosting mechanism of FOSTER requires more data samples.

Table 1: Accuracy [%] across various coreset fractions and selections on **Split-CIFAR10** with **from-scratch ResNet18**. Underlined results outperform training with all samples (w/o coreset) and the best results are highlighted in bold.

	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER [9]	56.91 ± 1.3	Random	51.79 ± 4.6	54.28 ± 3.8	55.68 ± 0.3	57.27 ± 2.9	55.61 ± 2.5
		Herding	41.65 ± 2.2	52.35 ± 2.5	59.79 ± 1.8	63.96 ± 1.1	62.93 ± 1.2
		Uncertainty	56.02 ± 1.7	59.48 ± 1.7	57.97 ± 0.8	62.01 ± 3.1	59.36 ± 1.5
		Forgetting	55.68 ± 2.1	60.97 ± 1.0	60.82 ± 0.3	63.46 ± 3.9	61.36 ± 0.4
		GraphCut	62.06 ± 1.9	**64.74 ± 0.5**	63.03 ± 2.0	61.17 ± 1.9	62.95 ± 1.5
FOSTER [10]	**54.79 ± 2.9**	Random	52.44 ± 5.4	52.34 ± 4.3	53.22 ± 2.8	53.93 ± 4.2	53.93 ± 3.0
		Herding	32.00 ± 2.2	39.91 ± 8.3	46.91 ± 3.3	52.82 ± 2.6	51.34 ± 1.2
		Uncertainty	45.42 ± 3.6	49.18 ± 4.6	48.94 ± 3.2	50.95 ± 2.6	49.25 ± 2.2
		Forgetting	45.44 ± 3.2	51.59 ± 4.0	49.37 ± 0.2	48.19 ± 2.6	49.10 ± 1.5
		GraphCut	50.85 ± 3.1	52.54 ± 3.7	49.94 ± 0.3	49.43 ± 0.9	49.28 ± 1.0
MEMO [11]	49.22 ± 5.5	Random	44.36 ± 4.2	45.41 ± 5.5	47.45 ± 6.4	48.93 ± 7.1	49.58 ± 7.2
		Herding	39.32 ± 0.2	45.04 ± 0.4	47.90 ± 3.1	49.98 ± 6.1	49.34 ± 6.3
		Uncertainty	38.27 ± 6.9	41.10 ± 5.0	44.99 ± 6.4	47.75 ± 6.0	47.90 ± 5.4
		Forgetting	35.04 ± 4.1	45.23 ± 5.4	47.74 ± 5.3	48.66 ± 5.5	47.78 ± 5.9
		GraphCut	51.37 ± 3.6	**52.54 ± 2.3**	49.67 ± 4.0	49.97 ± 6.0	48.35 ± 5.7
iCaRL [21]	59.54 ± 8.0	Random	47.70 ± 4.3	55.41 ± 5.4	54.56 ± 5.8	57.75 ± 7.5	57.29 ± 6.3
		Herding	40.32 ± 5.0	42.99 ± 3.3	54.02 ± 4.5	58.60 ± 6.7	59.11 ± 6.3
		Uncertainty	50.77 ± 1.5	54.41 ± 6.2	56.78 ± 6.3	57.38 ± 6.6	57.82 ± 7.1
		Forgetting	53.79 ± 4.9	57.86 ± 5.9	58.30 ± 5.9	58.90 ± 6.3	56.90 ± 7.7
		GraphCut	**61.70 ± 2.7**	61.07 ± 4.2	60.88 ± 5.6	58.80 ± 7.0	57.68 ± 7.1
ER [22]	58.51 ± 6.4	Random	51.02 ± 2.7	56.32 ± 6.2	57.79 ± 4.6	57.20 ± 6.0	57.77 ± 6.9
		Herding	41.06 ± 7.5	47.97 ± 4.0	55.87 ± 4.9	58.93 ± 4.6	58.85 ± 4.9
		Uncertainty	52.70 ± 2.4	52.99 ± 1.1	56.35 ± 6.3	57.48 ± 6.4	58.09 ± 5.4
		Forgetting	52.44 ± 3.4	55.05 ± 5.8	57.43 ± 5.7	57.00 ± 5.5	56.73 ± 6.2
		GraphCut	**63.03 ± 3.1**	60.53 ± 2.6	60.34 ± 4.4	58.69 ± 5.6	57.61 ± 5.8
LwF [3]	51.15 ± 4.3	Random	31.60 ± 0.8	41.46 ± 1.9	45.64 ± 1.5	51.21 ± 4.7	**51.83 ± 2.1**
		Herding	15.27 ± 3.8	23.75 ± 3.0	20.72 ± 0.7	27.74 ± 5.2	30.86 ± 4.1
		Uncertainty	26.89 ± 5.0	24.21 ± 3.3	28.95 ± 5.1	29.58 ± 5.8	30.54 ± 4.2
		Forgetting	27.10 ± 5.3	25.49 ± 4.0	27.66 ± 5.2	30.24 ± 5.5	30.57 ± 5.0
		GraphCut	25.34 ± 3.1	26.22 ± 3.5	29.42 ± 5.2	30.54 ± 4.2	30.95 ± 5.4

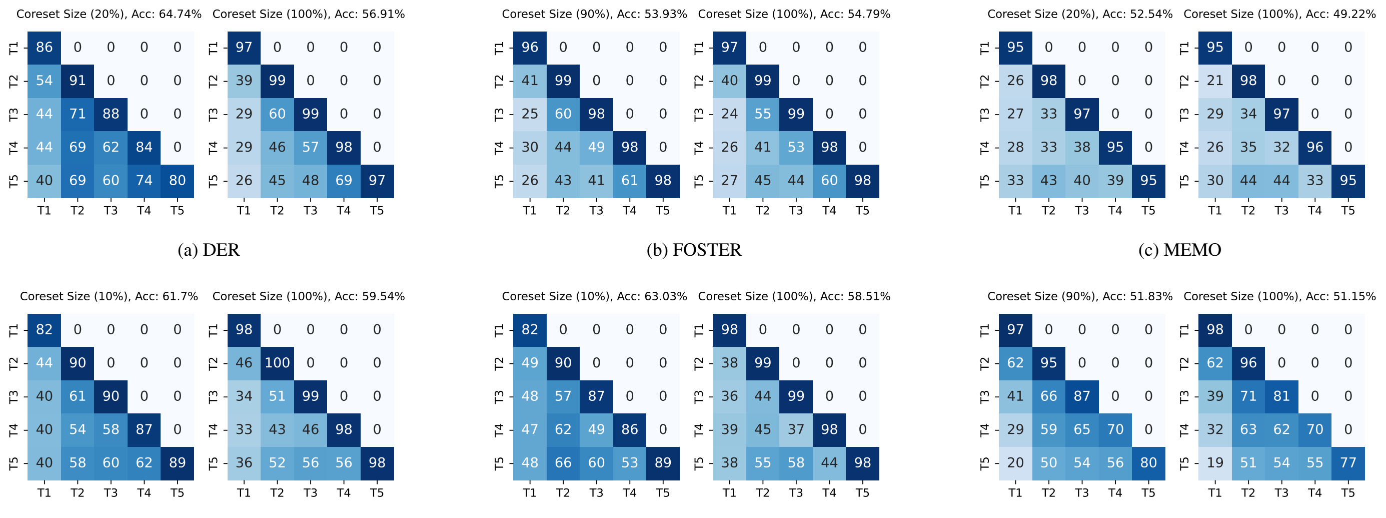
Table 2: Accuracy [%] across various coreset fractions and selections on **Split-CIFAR100** with **from-scratch ResNet18**. Underlined results outperform training with all samples (w/o coreset) and the best results are highlighted in bold.

	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER [9]	53.81 ± 1.0	Random	26.23 ± 0.6	36.35 ± 2.8	47.32 ± 2.6	53.11 ± 1.6	54.07 ± 0.1
		Herding	17.99 ± 7.5	24.79 ± 6.0	41.11 ± 2.7	52.48 ± 0.4	53.92 ± 0.8
		Uncertainty	27.54 ± 4.6	38.29 ± 3.0	49.41 ± 1.2	**55.71 ± 1.9**	54.55 ± 0.4
		Forgetting	30.32 ± 4.9	41.25 ± 1.8	49.20 ± 2.2	54.10 ± 0.3	53.68 ± 0.1
		GraphCut	29.61 ± 5.7	39.71 ± 3.4	50.35 ± 1.0	53.08 ± 0.8	54.89 ± 0.7
FOSTER [10]	**56.19 ± 2.3**	Random	23.21 ± 0.0	32.04 ± 1.3	48.95 ± 0.8	51.71 ± 1.9	53.34 ± 0.8
		Herding	10.84 ± 0.8	18.38 ± 1.1	35.15 ± 2.7	51.51 ± 0.1	53.72 ± 0.9
		Uncertainty	16.97 ± 0.1	27.37 ± 0.9	44.29 ± 3.1	55.24 ± 0.1	55.10 ± 1.7
		Forgetting	21.80 ± 0.4	32.42 ± 0.8	44.97 ± 2.9	54.59 ± 0.4	54.91 ± 1.0
		GraphCut	22.16 ± 1.6	30.40 ± 1.1	45.91 ± 2.3	53.35 ± 1.9	55.24 ± 0.5
MEMO [11]	34.23 ± 0.4	Random	20.79 ± 0.7	26.74 ± 0.1	29.62 ± 0.5	34.27 ± 0.2	34.58 ± 0.1
		Herding	13.24 ± 2.0	18.76 ± 1.5	27.26 ± 1.8	33.64 ± 0.3	**34.94 ± 0.1**
		Uncertainty	16.07 ± 2.6	23.23 ± 2.9	30.14 ± 1.7	33.41 ± 0.9	34.10 ± 1.0
		Forgetting	18.44 ± 1.9	23.37 ± 2.0	31.17 ± 0.3	33.10 ± 0.4	32.46 ± 2.2
		GraphCut	23.21 ± 1.7	27.79 ± 0.6	32.49 ± 0.6	33.61 ± 0.2	34.22 ± 0.7
iCaRL [21]	37.45 ± 1.7	Random	25.48 ± 0.2	29.87 ± 3.0	35.37 ± 2.0	37.02 ± 3.1	37.11 ± 3.0
		Herding	13.02 ± 1.2	17.24 ± 1.5	27.91 ± 1.3	38.24 ± 1.3	37.55 ± 0.8
		Uncertainty	22.47 ± 1.9	28.05 ± 1.3	35.18 ± 3.3	**40.25 ± 0.7**	39.26 ± 2.5
		Forgetting	25.00 ± 0.3	27.80 ± 1.1	33.27 ± 2.0	37.80 ± 1.0	37.44 ± 2.2
		GraphCut	24.04 ± 0.7	30.45 ± 0.2	33.31 ± 0.3	35.76 ± 3.2	38.03 ± 0.8
ER [22]	39.53 ± 1.6	Random	25.23 ± 0.3	31.58 ± 3.0	37.64 ± 1.4	39.25 ± 1.3	40.66 ± 2.0
		Herding	19.13 ± 5.4	24.90 ± 6.3	34.92 ± 4.0	40.18 ± 2.1	**41.19 ± 1.2**
		Uncertainty	25.77 ± 4.6	31.63 ± 4.3	36.61 ± 1.5	41.14 ± 0.4	39.69 ± 1.4
		Forgetting	29.53 ± 4.7	33.97 ± 3.8	36.96 ± 3.4	40.58 ± 0.7	39.92 ± 2.5
		GraphCut	32.99 ± 8.7	38.22 ± 6.4	39.55 ± 3.5	39.61 ± 2.6	39.97 ± 0.6
LwF [3]	22.82 ± 1.4	Random	11.39 ± 1.0	15.38 ± 1.3	20.26 ± 1.3	22.93 ± 2.1	**23.91 ± 1.2**
		Herding	3.67 ± 1.3	6.22 ± 0.1	12.43 ± 2.0	17.09 ± 4.6	18.08 ± 4.5
		Uncertainty	9.55 ± 0.5	12.17 ± 1.8	15.54 ± 2.8	18.72 ± 5.0	18.00 ± 4.2
		Forgetting	9.93 ± 1.3	12.75 ± 2.7	15.18 ± 2.9	17.99 ± 4.5	18.28 ± 4.4
		GraphCut	8.17 ± 0.3	10.37 ± 1.4	15.56 ± 3.4	17.26 ± 4.1	18.00 ± 4.9

Regardless of the model architecture, using coreset strategies consistently outperforms training on the full dataset with some nuances. Smaller convolutional neural networks, such as from-scratch ResNet18 (Table 2) and pre-trained ResNet18 (Table 4), require larger coresets to ensure an increase in performance, while larger and more powerful transformer-based models like pre-trained ViTs (Table A) can perform exceptionally well with smaller coresets.

From the perspective of CL, all learners highly benefit from the selected coresets, except FOSTER and LwF. The way FOSTER works requires to identify critical elements that were potentially overlooked or misinterpreted by the original model during the learning process. For instance, in the initial stages of learning, certain features may have been deemed less significant than others. However, as the model progresses and encounters new concepts, previously redundant features may become crucial. FOSTER addresses these dynamics by employing a feature-boosting mechanism, which aims to highlight the evolving importance of features over time. However, this mechanism necessitates more samples to capture the intricate relationships between features effectively. Consequently, training with the full dataset enables the model to comprehensively understand the underlying patterns and correlations among the features. Therefore, training FOSTER with coreset samples does not improve the stability-plasticity balance, as the limited sample size restricts the effectiveness of its feature-boosting mechanism.

To further investigate this phenomenon, we conduct an ablation study on FOSTER, evaluating the model under varying boosting fractions. We find that in the absence of boosting, the coreset selection can help to achieve a better stability-plasticity balance as illustrated in Figure 2c since the model is no longer dependent on feature re-evaluation over large sample contexts. The trend also shows that the performance of coreset training deteriorates when boosting fraction increases.



(a) DER (b) FOSTER (c) MEMO

(d) iCaRL (e) ER (f) LwF

Figure 3: Accuracy [%] of each task after every learning session with Split-CIFAR10. This comparison includes the performance using all samples *vs.* the best performing coreset selection, which may involve different coreset fractions. The underlying reason for the improved accuracy is attributed to reduced forgetting.

Table 3: Accuracy [%] on **Split-CIFAR10** with a **pre-trained ResNet18** shows that training with coreset samples improves incremental performance.

	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER	67.85 ± 3.30	Random	40.18 ± 5.28	53.93 ± 3.36	61.35 ± 2.37	66.66 ± 2.36	67.07 ± 2.51
		Herding	57.35 ± 0.45	61.48 ± 1.32	65.84 ± 2.66	68.68 ± 3.74	**71.36 ± 1.48**
		Uncertainty	61.23 ± 0.14	63.38 ± 0.40	67.63 ± 1.37	70.75 ± 2.71	70.92 ± 2.08
		Forgetting	61.00 ± 0.23	65.02 ± 0.66	67.86 ± 1.84	71.72 ± 2.08	69.67 ± 2.78
		GraphCut	62.00 ± 2.03	64.87 ± 1.92	68.39 ± 0.98	71.72 ± 1.65	71.19 ± 2.77
FOSTER	57.85 ± 3.09	Random	42.82 ± 7.84	46.24 ± 2.57	60.15 ± 2.88	57.89 ± 4.07	58.32 ± 5.71
		Herding	48.72 ± 4.27	50.35 ± 2.35	54.76 ± 4.46	56.71 ± 2.53	57.06 ± 3.39
		Uncertainty	54.51 ± 1.48	58.51 ± 2.97	58.34 ± 3.54	56.85 ± 4.81	56.35 ± 3.38
		Forgetting	52.26 ± 0.45	55.52 ± 5.48	57.61 ± 3.53	57.65 ± 2.90	55.98 ± 2.98
		GraphCut	53.84 ± 3.70	**59.27 ± 3.28**	58.04 ± 3.86	57.57 ± 3.71	56.09 ± 2.59
MEMO	55.65 ± 8.06	Random	37.49 ± 4.08	43.77 ± 10.63	48.74 ± 7.63	53.90 ± 2.21	59.34 ± 4.88
		Herding	34.50 ± 7.48	44.94 ± 12.11	55.14 ± 7.53	**62.84 ± 5.82**	61.34 ± 5.19
		Uncertainty	43.02 ± 5.27	50.06 ± 6.13	54.55 ± 6.44	61.21 ± 5.79	62.00 ± 5.72
		Forgetting	37.64 ± 4.28	49.77 ± 8.80	54.98 ± 6.70	62.84 ± 5.78	61.84 ± 6.93
		GraphCut	47.23 ± 3.19	52.04 ± 8.08	55.96 ± 6.87	61.57 ± 5.18	61.37 ± 5.61
iCaRL	53.37 ± 5.94	Random	38.85 ± 0.13	47.22 ± 7.77	48.32 ± 3.87	48.97 ± 3.02	52.03 ± 5.62
		Herding	53.52 ± 2.71	55.21 ± 1.45	53.68 ± 6.33	55.42 ± 5.13	55.38 ± 4.59
		Uncertainty	53.72 ± 3.14	56.03 ± 1.67	53.81 ± 5.12	56.82 ± 6.18	54.73 ± 5.88
		Forgetting	53.20 ± 0.90	56.00 ± 4.88	54.76 ± 5.06	55.62 ± 5.33	54.98 ± 6.39
		GraphCut	**57.99 ± 2.41**	57.98 ± 3.45	57.03 ± 3.85	55.63 ± 4.50	57.79 ± 5.47
ER	45.01 ± 5.56	Random	41.21 ± 2.43	43.55 ± 6.68	43.21 ± 5.02	44.16 ± 6.60	44.56 ± 6.71
		Herding	38.28 ± 4.17	41.91 ± 3.25	47.91 ± 2.85	44.76 ± 7.06	43.17 ± 6.39
		Uncertainty	36.23 ± 3.22	40.28 ± 7.42	42.19 ± 6.85	44.01 ± 8.18	43.81 ± 5.51
		Forgetting	34.70 ± 3.03	42.90 ± 5.67	44.66 ± 6.07	44.41 ± 6.35	43.95 ± 6.01
		GraphCut	**52.26 ± 3.93**	50.82 ± 4.91	46.33 ± 5.23	44.35 ± 7.20	45.11 ± 7.77
LwF	53.94 ± 0.79	Random	30.80 ± 1.42	41.67 ± 1.89	45.95 ± 3.11	51.04 ± 0.38	**54.74 ± 0.44**
		Herding	17.65 ± 0.23	21.74 ± 3.19	26.41 ± 3.72	29.85 ± 6.65	31.53 ± 6.01
		Uncertainty	25.21 ± 5.02	26.38 ± 6.01	27.76 ± 6.16	30.68 ± 6.37	32.13 ± 6.92
		Forgetting	23.68 ± 1.81	26.99 ± 5.19	27.60 ± 5.33	30.74 ± 5.98	30.82 ± 6.81
		GraphCut	26.45 ± 5.28	25.23 ± 4.16	27.79 ± 5.35	31.05 ± 5.38	31.78 ± 5.26

Table 4: Accuracy [%] on **Split-CIFAR100** with a **pre-trained ResNet18** shows that training with coreset samples improves incremental performance.

	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER	55.85 ± 0.38	Random	20.38 ± 3.27	30.82 ± 0.76	44.96 ± 0.28	53.41 ± 1.96	52.23 ± 0.84
		Herding	16.33 ± 4.78	22.13 ± 8.92	47.52 ± 2.47	55.51 ± 0.89	56.74 ± 1.09
		Uncertainty	30.03 ± 0.62	40.53 ± 0.98	52.21 ± 0.78	56.94 ± 0.97	**57.22 ± 0.59**
		Forgetting	30.08 ± 4.11	37.48 ± 5.50	51.88 ± 0.81	56.18 ± 1.53	56.16 ± 1.08
		GraphCut	28.20 ± 1.64	38.79 ± 1.66	50.94 ± 1.59	55.76 ± 0.68	56.95 ± 1.77
FOSTER	56.63 ± 1.11	Random	16.25 ± 0.27	19.71 ± 0.45	34.21 ± 3.55	50.80 ± 0.07	50.65 ± 1.36
		Herding	12.51 ± 0.03	17.86 ± 1.39	37.88 ± 1.58	54.25 ± 2.37	55.40 ± 2.13
		Uncertainty	14.87 ± 1.03	23.91 ± 0.86	45.93 ± 1.50	55.21 ± 2.26	**56.65 ± 2.27**
		Forgetting	18.44 ± 0.72	24.46 ± 1.84	44.04 ± 0.33	55.45 ± 2.08	56.30 ± 1.21
		GraphCut	17.87 ± 2.30	22.10 ± 3.81	44.94 ± 0.94	55.51 ± 1.93	56.60 ± 2.16
MEMO	46.70 ± 3.64	Random	17.21 ± 1.91	25.29 ± 0.42	38.54 ± 3.05	43.16 ± 2.88	46.32 ± 3.75
		Herding	10.94 ± 0.72	20.13 ± 0.21	36.26 ± 0.94	44.29 ± 0.75	**46.87 ± 0.24**
		Uncertainty	17.85 ± 1.05	24.54 ± 0.15	37.92 ± 0.73	44.87 ± 0.30	46.10 ± 0.57
		Forgetting	21.56 ± 0.52	28.20 ± 0.51	38.59 ± 1.06	44.49 ± 0.88	45.86 ± 0.58
		GraphCut	27.60 ± 5.53	33.44 ± 4.45	40.38 ± 0.13	44.54 ± 0.29	45.60 ± 0.08
iCaRL	32.90 ± 0.80	Random	20.09 ± 0.72	22.25 ± 0.93	30.08 ± 0.04	30.40 ± 1.16	33.60 ± 0.66
		Herding	18.46 ± 0.72	24.80 ± 1.56	32.74 ± 2.12	34.70 ± 2.10	34.74 ± 2.08
		Uncertainty	22.70 ± 0.23	27.82 ± 0.88	32.68 ± 1.42	33.44 ± 1.26	34.04 ± 1.65
		Forgetting	24.22 ± 0.69	30.00 ± 1.38	33.85 ± 2.05	34.16 ± 2.72	35.21 ± 2.10
		GraphCut	28.88 ± 0.34	30.93 ± 2.39	**35.40 ± 1.56**	34.17 ± 0.96	34.02 ± 1.47
ER	24.58 ± 0.46	Random	16.6 ± 3.59	22.35 ± 0.04	26.09 ± 0.34	25.42 ± 0.10	24.91 ± 0.16
		Herding	15.2 ± 0.8	19.9 ± 0.32	25.16 ± 0.97	25.94 ± 1.52	25.30 ± 0.83
		Uncertainty	14.4 ± 0.46	17.56 ± 0.62	22.78 ± 0.24	24.04 ± 0.14	25.58 ± 0.61
		Forgetting	19.01 ± 0.63	21.72 ± 0.14	25.57 ± 0.69	25.69 ± 0.89	26.26 ± 1.55
		GraphCut	27.01 ± 0.34	**28.99 ± 1.63**	27.52 ± 0.57	26.03 ± 1.43	25.43 ± 0.86
LwF	24.31 ± 0.57	Random	10.39 ± 0.36	12.63 ± 1.40	20.69 ± 0.70	22.78 ± 0.38	**25.01 ± 0.46**
		Herding	4.15 ± 0.11	5.44 ± 0.10	9.47 ± 0.84	13.11 ± 1.53	13.77 ± 0.96
		Uncertainty	7.42 ± 0.01	9.15 ± 0.22	11.00 ± 0.58	13.29 ± 1.18	14.46 ± 0.99
		Forgetting	7.26 ± 0.24	8.22 ± 0.17	10.89 ± 0.86	13.06 ± 1.14	14.04 ± 0.94
		GraphCut	6.59 ± 0.32	7.23 ± 0.32	11.13 ± 0.67	13.21 ± 1.21	13.65 ± 1.13

Our analysis on LwF shows that when more advanced coreset selection methods, such as Uncertainty and GraphCut, are employed, it demonstrates superior adaptability to the current task. However, this enhanced adaptability comes at a cost of catastrophic forgetting. To unravel the root cause of this forgetting phenomenon, we examine the changes in model parameters between consecutive tasks. We found that Uncertainty and GraphCut induce abrupt changes in the parameters, whereas it is comparatively smaller with randomly selected samples, as shown in Figure 2b.

This is because, coreset selection strategies (e.g., Herding, Uncertainty, and GraphCut) prioritize the most informative samples specific to the current task. When the CL approach, such as LwF, relies solely on regularization, this prioritization can lead to overfitting the current task's distribution. Such overfitting amplifies significant representation shifts, resulting in abrupt parameter updates that results in catastrophic forgetting of previously learned tasks. In contrast, random selection implicitly incorporates as a form of regularizer with a greater diversity and variability in the sample distribution across tasks, which helps to mitigate overfitting and results in more stable parameter updates. This suggests that the traditional regularization methods may not be as effective as replay-based approaches when considering coreset utilization.

5.2. *Plasticity-Stability Balance and Task Retention*

Coreset training influences the balance between learning new information (plasticity) and retaining previous knowledge (stability). Our per-task accuracy analysis with confusion matrices reveals that models trained on coreset samples generally exhibit stronger knowledge retention, even though they may sometimes sacrifice performance on the current task (Figure 3).

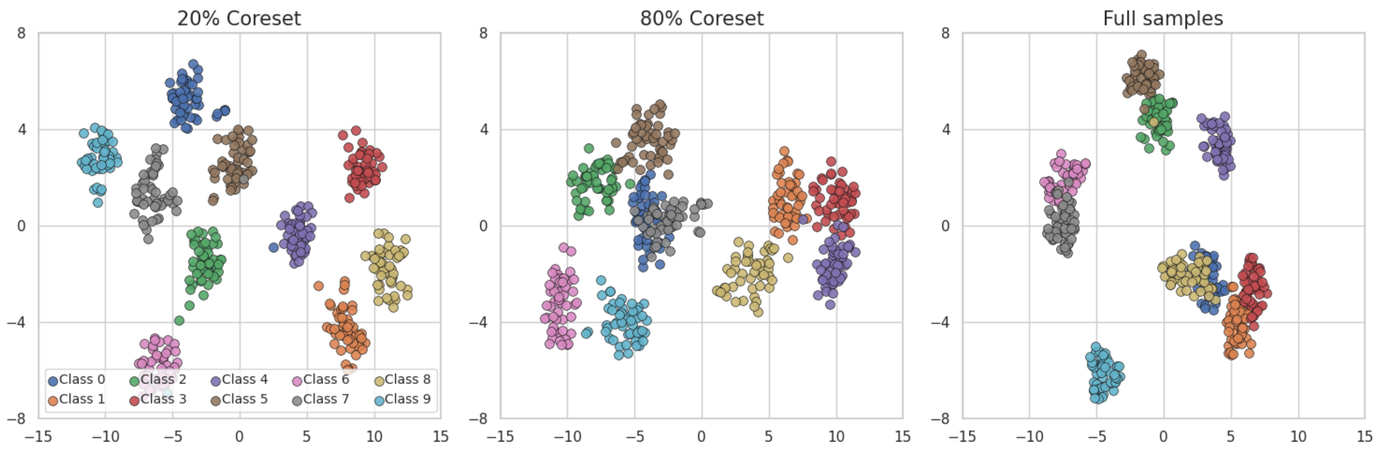


Figure 4: DER's representation of all classes on Split-CIFAR10 with varying coresets selected with GraphCut, compared to the full samples. When it is trained with 20%coreset samples, it exhibits better ability to have distinct representations.

Sample selection before training is also crucial in enhancing the data quality utilized during the replay or memory construction phase in continual learning. By filtering out potentially irrelevant or redundant data points beforehand, it ensures that only informative and representative samples would be candidates for memory construction. This contributes to enhanced retention of learned knowledge from previous tasks over time by focusing on key patterns. Consequently, all CL approaches except FOSTER demonstrate noticeable improvement in knowledge retention when trained on coreset samples.

Additionally, we observe that pre-trained models tend to enhance stability without compromising plasticity. This indicates that they are particularly effective for incremental learning scenarios when utilized on strategically selected coreset samples (Table 3, 4, A). Further details, including per-task accuracy after each learning session on pre-trained models are provided in Appendix A.4. Overall, these observations indicate that the enhanced incremental performance with coreset selection is primarily attributed to knowledge retention.

5.3. *Coreset Training and Representation Dynamics*

To further understand the impact of coreset selection on knowledge retention for continual learning, we analyze how coreset training influences the model's representation space. Specifically, we visualize the learned feature distributions using t-SNE plots for models trained with 20% and 80% coreset fractions, as well as full-sample training (Figure 4). Our goal is to examine whether coreset training leads to more distinct class boundaries, potentially contributing to improved knowledge retention. Our findings on Split-CIFAR10 scenario indicate that when using a smaller coreset (e.g., 20%), the model demonstrates a clearer separation between class representations, maintaining well-defined decision boundaries. This suggests that training on a compact yet informative subset of the data enables the model to develop more distinct class-wise feature spaces, which could be beneficial for incremental learning. This observation is supported by a higher average inter-class distance of 11.68 in the t-SNE-embedded space, indicating more distinct clusters. As the coreset fraction increases to 80%, we observe that class representations begin to overlap, reducing their separability, with the corresponding inter-class distance dropping to 10.71. This trend continues in the full training setting, where the inter-class distance further decreases to 10.67, and certain classes from the initial task (class 0) and the last task (class 8) exhibit significant overlap, potentially leading to increased forgetting of initial tasks. This observation supports the idea that redundant and uninformative training data may introduce the risk of increasing misclassification.

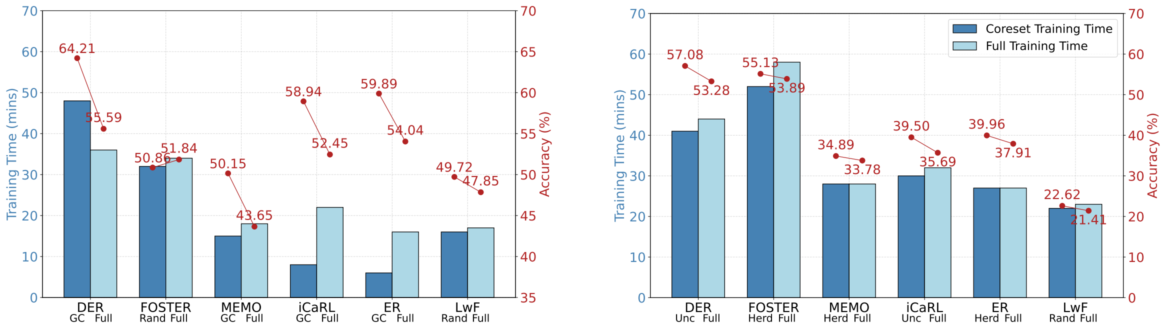
However, it is important to acknowledge that these trends are not universally observed across all datasets and experimental settings. For instance, in Split-CIFAR100, where each class already has a relatively small number of samples (500 per class), 20% coreset fraction may be too restrictive, leading to a loss of critical information. In such scenarios, we find that larger coresets (e.g., 80% or 90%) can actually outperform full dataset training, as they strike a better balance between data sufficiency and noise reduction. Overall, our results emphasize that coreset plays a crucial role in shaping the model's feature representations and, consequently, its stability-plasticity tradeoff.

5.4. *Further Analysis and Discussion*

In this section, we provide complementary empirical analyses to better understand different aspects of our approach. We begin with an ablation study examining the effect of varying the number of warm-up epochs on model performance. Next, we analyze the computational complexity of different coreset selection methods, offering a comparative discussion of their time efficiency vs. accuracy. Finally, we explore the interaction between different backbone and coreset sizes through an additional discussion.

5.4.1. *Ablation study for the warm-up training*

We present an analysis to support our choice of allocating 10% of the training budget to the warm-up phase in our two-stage learning framework. This warm-up phase precedes the main continual learning stage and is needed to perform coreset selection. The remaining 90% of the budget is dedicated to the continual learning phase.



(a) CIFAR-10 (b) CIFAR-100

Figure 5: Training time vs. accuracy trade-offs across continual learning methods using their best-performing coreset selection strategies.

Table 5: Ablation of different warm-up fractions.

Warm-up epoch fraction	0.01	0.05	0.1	0.15	0.2
Herding	50.73	51.96	**52.16**	51.35	51.98
Uncertainty	51.87	53.98	**54.34**	54.05	54.02
Forgetting	52.98	53.50	**54.40**	54.26	54.03
GraphCut	51.91	52.96	53.16	**53.68**	53.04

Experiments were carried out on the DER method with a coreset fraction of 80%, testing four selection strategies of herding, uncertainty, forgetting, and graphcut. As shown in Table 5, setting the warm-up fraction to 0.1 leads to strong performance across all strategies. Although minor improvements are observed beyond fraction 0.1 in some cases, the overall performance gains are limited. These results empirically support our decision to set the warm-up fraction to $\alpha = 0.1$, striking a balance between computational efficiency and selection quality.

5.4.2. *Computational Complexity*

To comprehensively evaluate the trade-off between computational efficiency and predictive performance, we conducted an in-depth analysis of training time versus accuracy for each CL algorithm, pairing it with its best-performing coreset selection method. For example, DER is paired with GraphCut, as this combination yielded the best overall performance among all coreset selection techniques evaluated for DER. This design choice aims to reflect a realistic deployment scenario, where practitioners would adopt the most effective coreset strategy available for a given CL approach.

The results are presented in Figure 5 provides side-by-side comparisons for Split-CIFAR-10 and Split-CIFAR-100. Our analysis reveals that coreset-based training leads to reduced training times compared to full-data training, in many cases. One notable exception is the DER–GraphCut combination, which incurs a higher training cost than full-data training. This is due to the inherent complexity of architecture expansion mechanism in DER and the computational overhead of the GraphCut algorithm, involving graph construction and partitioning. In more complex or large-scale CL settings, such graph-based methods may become computationally burdensome. Nonetheless, this pairing still demonstrates a significant improvement in incremental accuracy, illustrating a beneficial trade-off in terms of performance gains.

Table 6: Comparison of different backbones trained from scratch on Split-CIFAR10 with different fraction sizes of Uncertainty with DER method.

	w/o Coreset	0.1	0.2	0.5	0.8	0.9
ResNet18	56.91 ± 1.3	56.02 ± 1.7	<u>59.48</u> ± 1.7	<u>57.97</u> ± 0.8	**62.01** ± 3.1	<u>59.36</u> ± 1.5
ResNet50	57.20 ± 0.0	51.1 ± 3.8	55.17 ± 2.8	<u>57.82</u> ± 3.0	**61.37** ± 4.0	<u>60.47</u> ± 2.9
ResNet101	58.36 ± 2.8	51.25 ± 0.1	55.73 ± 0.7	<u>59.51</u> ± 2.0	<u>63.28</u> ± 3.2	**64.72** ± 2.1

5.4.3. *Model Complexity*

To investigate how model capacity interacts with coreset size in continual learning, we train ResNet18, ResNet50, and ResNet101 from scratch using uncertainty-based coreset selection in combination with DER. Across all architectures, models trained with selected coresets outperform those trained on the full dataset, demonstrating the effectiveness of coreset-based training under different backbone settings (Table 6). The smaller ResNet18 consistently performs better than its deeper counterparts at lower coreset fractions (e.g., 0.1 and 0.2) when trained from scratch. As the coreset size increases, larger backbones such as ResNet50 and ResNet101 begin to exhibit clear performance advantages. In mid- to high-range fractions (e.g., 0.5 and above), these models not only outperform ResNet18 but also surpass their own performance when trained on the full dataset.

6. Conclusion

Existing CL approaches predominantly use all available data during training yet not all samples carry equal informational value and not need to go under the training process. In this study, we explore the underutilized potential of selective learning from key samples, demonstrating that model performance is strongly influenced by both the quality and quantity of data. Our empirical analysis yields three key findings that challenge and extend current CL methodologies. We show that learning from coreset samples enhances incremental performance. We attribute this improvement to better stability-plasticity balance across tasks, achieved by reducing redundancy and focusing on high-value information. Further, we observe that models trained with coresets maintain clearer class distinctions by the end of all sessions. This distinct class separability directly contributes to improved knowledge retention, as models can better differentiate between old and new tasks. Among the methods we evaluated, GraphCut frequently yielded strong performance across diverse scenarios and setups. Furthermore, our analysis highlights that DER, iCaRL, and ER methods benefit significantly from coreset training by achieving better stability-plasticity tradeoffs. Overall, our findings with the provided analysis underscore the potential of data-centric CL. Future studies would aim to achieve optimal stability with enhanced plasticity with CL-tailored coreset strategies. Such advancements could further enhance CL performance, enabling models to adapt more effectively to evolving tasks while maintaining robust knowledge retention.

.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24. Elsevier, 1989.

[2] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13), 2017.

[3] Zhizhong Li and Derek Hoiem. Learning without forgetting. *TPAMI*, 40(12), 2017.

[4] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *NeurIPS*, 2017.

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[6] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *NeurIPS*, 2017.

[7] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *NeurIPS*, 2019.

[8] Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *NeurIPS*, 2020.

[9] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, 2021.

[10] Fu-Yun Wang, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. Foster: Feature boosting and compression for class-incremental learning. In *ECCV*. Springer, 2022.

[11] Da-Wei Zhou, Qi-Wei Wang, Han-Jia Ye, and De-Chuan Zhan. A model or 603 exemplars: Towards memory-efficient class-incremental learning. *arXiv preprint arXiv:2205.13218*, 2022.

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

[13] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019.

[14] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *CVPR*, 2022.

[15] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *ECCV*. Springer, 2022.

[16] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *CVPR*, 2023.

[17] Mattia F Pagnotta, David Pascucci, and Gijs Plomp. Selective attention involves a feature-specific sequential release from inhibitory gating. *Neuroimage*, 246:118782, 2022.

[18] LA Jones, PJ Hills, KM Dick, SP Jones, and P Bright. Cognitive mechanisms associated with auditory sensory gating. *Brain and cognition*, 102, 2016.

[19] Michael I Posner and Steven E Petersen. The attention system of the human brain. *Annual review of neuroscience*, 13(1), 1990.

[20] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[21] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

[22] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *NeurIPS*, 2019.

[23] Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*. PMLR, 2021.

[24] Alaa Maalouf, Gilad Eini, Ben Mussay, Dan Feldman, and Margarita Osadchy. A unified approach to coreset learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[25] Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 2018.

[26] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[27] Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. In *International Conference on Database and Expert Systems Applications*. Springer, 2022.

[28] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

[29] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. *NeurIPS*, 2021.

[30] Max Welling. Herding dynamical weights to learn. In *ICML*, 2009.

[31] Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

[32] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *ICML*, 2020.

[33] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *ICML*, 2015.

[34] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[35] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. In *ICLR*, 2022.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Toronto, ON, Canada*, 2009.

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115, 2015.

[38] Da-Wei Zhou, Fu-Yun Wang, Han-Jia Ye, and De-Chuan Zhan. Pycil: a python toolbox for class-incremental learning. *SCIENCE CHINA Information Sciences*, 66(9), 2023.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[40] Alexey Dosovitskiy, Lucas Beyer, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2021.

Appendix A. Appendix

In this appendix, we first give more detailed explanations of the continual learning methods and coreset selection methods used in our experiments. This includes a comprehensive overview of the baseline methods and their key characteristics. Next, we provide more details about our implementation for the backbones we used and the metrics that we evaluated. Then, we share the accuracy of each task after every learning session for the Split-CIFAR100 dataset trained with ResNet18, similar to Figure 3. Finally, we provide more results with pre-trained ResNet18 and pre-trained ViT on Split-CIFAR10 and Split-CIFAR100.

Appendix A.1. Continual Learning Approaches

In our evaluation, we selected a diverse set of continual learning methods to ensure a comprehensive analysis, including regularization-based, replay-based, architecture-based, and prompt-based approaches. In total, we evaluated seven different methods.

Appendix A.1.1. Regularization-based Methods

Regularization-based methods utilize a single backbone, meaning they rely on one fixed architecture without altering its structure. These methods operate without accessing any memory data, working solely with the data from the current task. This constraint makes them particularly challenging compared to other approaches. To mitigate catastrophic forgetting, these methods regularize weight updates during the learning of each new task. By carefully controlling the extent of weight changes, they ensure that the model retains knowledge from previous tasks.

LwF is one of the most well-known and well-established regularization approaches in continual learning. It tackles catastrophic forgetting by leveraging knowledge distillation to transfer knowledge from a previously trained model (the teacher) to the current model (the student) as new tasks are introduced. When training on a new task, LwF preserves the knowledge of earlier tasks by ensuring the current model reproduces the predictions of the teacher model for the classes associated with prior tasks. Specifically, the teacher model is frozen after completing a task and generates *soft labels* for the new training data, which represent the probability distribution over previously learned classes. Formally, the learning process is guided by two losses: cross-entropy loss $\mathcal{L}_{\text{CE}}$ given in Eq A.1 where y_i is the true label and p_i is the predicted probability for the i-th input, and the distillation loss $\mathcal{L}_{\text{KL}}$ given in Eq A.2 where $q_{\text{teacher}}(x_i)$ is the probability distribution from the teacher model and $q_{\text{student}}(x_i)$ is the probability distribution from the current model for the same input.

$$L_{\text{CE}} = -\sum_{i=1}^{N} y_i \log(p_i) \tag{A.1}$$

$$L_{\text{KL}} = \sum_{i=1}^{N} \text{KL}\left(q_{\text{teacher}}(x_i) \parallel q_{\text{student}}(x_i)\right) \tag{A.2}$$

Cross-entropy loss ensures that the model performs well on the current task and the distillation loss helps the model retain knowledge from previously learned tasks. It measures the difference between the predicted probability distributions of the current model and the teacher model for previously seen examples. This is typically calculated using the Kullback-Leibler (KL) divergence.

Finally, the CL loss $\mathcal{L}_{\text{CL}}$ for LwF is the combination of the cross-entropy loss $\mathcal{L}_{\text{CE}}$ and the distillation loss $\mathcal{L}_{\text{KL}}$ with a scaling factor of λ that controls the importance of the distillation loss:

$$L_{\text{CL}}^{\text{LwF}} = L_{\text{CE}} + \lambda L_{\text{KL}} \tag{A.3}$$

Appendix A.1.2. Replay-based Methods

Replay-based methods, on the other hand, employ an additional memory buffer to store a subset of past task data. While learning new tasks, these methods simultaneously utilize the memory buffer samples M together with current task samples N, allowing the model to retain a degree of knowledge about previous tasks. This mechanism provides a practical way to alleviate forgetting.

ER is a key replay-based method in continual learning. It maintains a memory buffer containing data from previous tasks and combines this replayed data with the new task data. The model then computes the cross-entropy loss, given in Eq. A.1, to evaluate how well the model's predictions align with the true labels for both the current task and the replayed task samples.

$$L_{\text{CL}}^{\text{ER}} = -\sum_{i=1}^{N+M} y_i \log(p_i) \tag{A.4}$$

iCaRL differs from the ER method by introducing a specific memory selection strategy, known as herding, and incorporating a distillation loss into its training objective. While ER relies solely on cross-entropy loss for current and replayed data, iCaRL combines cross-entropy loss Eq. A.1 with a distillation loss Eq. A.2 to strengthen its knowledge retention mechanism, similar to LwF.

$$L_{\text{CL}}^{\text{iCaRL}} = -\sum_{i=1}^{N+M} y_i \log(p_i) + \lambda L_{\text{KL}} \tag{A.5}$$

Appendix A.1.3. Architecture-based Methods

Architecture-based methods take a different approach by dynamically modifying the model's backbone. When encountering a new task, these methods either create a completely new architecture (a new model) or initialize additional components. The newly added parts are then trained specifically on the new task data, enabling the model to adapt structurally to task-specific requirements and reinforce knowledge retention.

DER initializes a new backbone for each task and aggregates features from both old (frozen) and new backbones using an expanded fully connected layer. This enables the model to specialize for new tasks while preserving knowledge from earlier tasks. A key component of DER is the auxiliary loss L_{aux} in Eq. A.6 which promotes learning diverse and discriminative features for each task. Therefore, it uses temporary auxiliary classes y^{aux} and classifier by treating all old classes as one category and the new classes as another. Therefore, the complete loss function for DER can be expressed as in Eq. A.7

$$L_{\text{aux}} = -\sum_{i=1}^{N+M} y_i^{aux} \log(p_i^{aux}) \tag{A.6}$$

$$L_{\text{CL}}^{\text{DER}} = -\sum_{i=1}^{N+M} y_i \log(p_i) + L_{\text{aux}} \tag{A.7}$$

MEMO operates on the assumption that shallow network layers capture general patterns, while deeper layers specialize in task-specific concepts. To accommodate new tasks, MEMO initializes fresh deep layers or blocks for each task while preserving the shallow layers unchanged. Consequently, the model expands only the deep layers for new tasks. MEMO employs the same loss function as the DER method, enhanced by the inclusion of an additional lambda hyperparameter that controls the auxiliary loss:

$$L_{\text{CL}}^{\text{MEMO}} = -\sum_{i=1}^{N+M} y_i \log(p_i) - \lambda L_{\text{aux}} \tag{A.8}$$

FOSTER combines feature boosting and feature compression in two stages to alleviate forgetting. In the boosting stage, FOSTER adds a new feature extractor to the model when a new task arrives. The new feature extractor learns residual features, which capture the differences (residuals) between the target outputs and the predictions from the frozen old model. These residual features are then concatenated with the frozen old features, creating a combined representation.

In the boosting stage FOSTER benefits from 2 different classifiers. The first one maintains the balance between old and new classes by aligning the logits (Eq.A.9), and the second one explicitly improves the representation of old classes by using only the new feature extractor's output over all classes (Eq. A.10). Finally, similar to LwF and iCaRL, knowledge distillation is applied during this stage to align the outputs of the new model with the frozen old model to further preserve the knowledge from previous tasks. Then the total loss for FOSTER can be expressed as in Eq. A.11.

$$L_{\text{LA}} = -\sum_{i=1}^{N+M} y_i \log(p_i^{\text{aligned}}) \tag{A.9}$$

$$L_{\text{FE}} = -\sum_{i=1}^{N+M} y_i \log(p_i^{\text{enhanced}}) \tag{A.10}$$

$$L_{\text{CL}}^{\text{FOSTER}} = L_{\text{LA}} + L_{\text{FE}} + \lambda L_{\text{KL}} \tag{A.11}$$

Following the boosting stage, where a new feature extractor is added to handle residual features for new tasks, the compression process starts to address the problem of parameter growth caused by the dynamic expansion of the model. In this final stage, the dual-branch architecture (frozen old model + new feature extractor) is compressed into a single compact backbone.

Appendix A.1.4. Prompt-based Methods

Prompt-based methods represent a recently developed approach in the field of continual learning. Drawing inspiration from prompt-tuning techniques in natural language processing, these methods use task-specific prompts to guide the model's behavior. Unlike traditional approaches that modify weights or architectures, prompt-based methods largely retain the shared backbone architecture and instead focus on learning small, task-specific prompts.

CODA-Prompt learns prompt components that are dynamically combined with input conditioned weights to create task-specific prompts. When a new task is introduced, a distinct prompt is initialized to capture task-specific information. This prompt interacts with the shared backbone model to activate the relevant representations for the current task. By localizing task-specific adaptations within the prompts, the model can effectively generalize across tasks while minimizing interference. This leads to the model's ability to retain previously acquired knowledge, as the large pre-trained backbone remains unchanged, while still efficiently learning new tasks. Formally, to achieve this, CODA-Prompt uses the prompt loss in Eq. A.12 that aims to maximize the alignment between the prompts and the task-specific features while minimizing redundancy or uninformative contributions. In this formulation, P_i represents the prompt embedding for block i, and Q_i denotes the corresponding input feature embedding for block i. The term $\|\cdot\|_2^2$ is the squared L_2-norm, which measures the difference between the prompt and input embeddings. N is the number of transformer blocks with prompts. This enforces that the prompt embeddings P_i are closely aligned with the input embeddings Q_i.

$$L_{\text{prompt}} = \frac{1}{N}\sum_{i=1}^{N} \|P_i - Q_i\|_2^2 \tag{A.12}$$

Finally, CODA-Prompt combines the standard cross-entropy loss for classification with the prompt loss summed over all transformer blocks to learn the task-specific prompts on top of pre-trained ViTs:

$$L_{\text{CL}}^{\text{CODA}} = -\sum_{i=1}^{N} y_i \log(p_i) + L_{\text{prompt}} \tag{A.13}$$

Appendix A.2. Coreset Selection Approaches

Coreset selection refers to the process of selecting a small, representative subset of data points from a given larger original dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ where x_i are the input features and y_i, such that the selected subset can approximate the performance of the full dataset for a given machine learning task.

Random selection is a straightforward approach to dataset reduction where a fixed number or proportion of data points is chosen uniformly at random from the original dataset. While this technique does not account for the importance or representativeness of individual samples, it serves as a strong and computationally efficient baseline. This method works by choosing I samples uniformly at random without replacement from the original dataset and the selected coreset C can be written as:

$$C = \{(x_i, y_i) \mid i \in I\} \tag{A.14}$$

Herding is a deterministic method for selecting a representative subset of data points, known as a coreset. It focuses on capturing the overall structure of the dataset by ensuring that the selected samples approximate the mean feature representation of the full dataset.

The method computes the mean of the features $m = \frac{1}{N}\sum_{i=1}^{N}\phi(x_i)$ where $\phi(x_i)$ maps each data point to a feature space, such as one generated by a neural network. The goal is to iteratively build a coreset C that closely approximates this mean.

The algorithm starts with an empty coreset and a residual vector $r = m$, which keeps track of the difference between the dataset mean and the cumulative contributions of the selected samples. At each step, the next sample to include in the coreset is chosen by finding the data point x_i whose feature vector $\phi(x_i)$ has the largest alignment with the residual vector r. This can be expressed as:

$$i^* = \arg\max_i r^\top \phi(x_i). \tag{A.15}$$

Once x_{i^*} is selected, it is added to the coreset, and the residual vector is updated by subtracting $\phi(x_{i^*})$. This process is repeated until the desired number of samples is selected, resulting in the coreset C. By iteratively reducing the residual, herding ensures that the selected coreset is highly representative of the original dataset. This makes it a valuable approach for tasks requiring a compact yet informative subset of data.

Uncertainty coreset selection is a method that prioritizes data points where the model exhibits the highest uncertainty in its predictions. The rationale is that these uncertain samples carry the most informative value, as they highlight areas where the model is less confident and likely to benefit from further training.

For a model $f(x)$ that outputs a probability distribution over classes, the uncertainty of a sample x_i can be quantified using measures such as entropy. The entropy for a prediction is computed as $H(x_i) = -\sum_{c\in C} p_c(x_i)\log p_c(x_i)$ where $p_c(x_i)$ is the predicted probability for class c, and C is the set of all possible classes. The uncertainty selection process involves computing $H(x_i)$ for all samples in the dataset and ranking them by their uncertainty scores. The top k samples with the highest entropy are chosen to form the coreset:

$$C = \{(x_i, y_i) \mid x_i \text{ ranks among the top } k \text{ in } H(x_i)\}. \tag{A.16}$$

By selecting the most uncertain samples, this method focuses on the regions of the data space where the model requires additional learning, ensuring an informative and compact coreset, particularly effective when resources for training are limited.

Forgetting coreset selection identifies data points that the model struggles to consistently classify correctly during training. These are known as "forgotten examples" because their predictions frequently change from correct to incorrect. By focusing on such challenging samples, this method selects a subset of data that is highly informative for improving the model's robustness.

During training, the model keeps track of whether it correctly predicts each sample at every training step. Let's denote the prediction correctness for a sample x_i at a given step as a binary value; 1 if the prediction is correct and 0 if the prediction is incorrect.

A *forgetting event* occurs when the model's prediction for a sample changes from correct to incorrect as training progresses. The forgetting score for a sample is simply the total number of forgetting events it experiences during training. For example, if the prediction for x_i flips from correct to incorrect three times, its forgetting score would be 3. Samples with higher forgetting scores are more challenging for the model to learn and retain. To form the coreset, the method ranks all samples by their forgetting scores and selects the top k samples with the highest scores:

$$C = \{(x_i, y_i) \mid x_i \text{ ranks among the top } k \text{ in high forgetting score}\} \tag{A.17}$$

This approach ensures the coreset contains the most challenging and informative examples, which can help the model learn and retain knowledge more effectively.

GraphCut coreset selection models the dataset as a graph to identify a subset of representative data points by leveraging the relationships among samples. In this approach, each data points x_i is represented as a node $v_i \in V$, and each edge $e_{ij} \in E$ between nodes v_i and v_j represent weighted by a similarity metric $s(x_i, x_j)$ in a graph $G = (V, E)$. A similarity metric $s(x_i, x_j)$ with a scaling factor σ can be defined as:

$$s(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{2\sigma^2}\right), \tag{A.18}$$

The goal is to form a coreset C by selecting a subset of nodes such that the cut value of the partition is minimized while preserving representativeness. The cut value measures the total similarity between the coreset and the remaining dataset, or is defined formally as the sum of the weights of the edges crossing between the selected subset C and the remaining nodes $V \setminus C$ as in Eq. A.19. A lower cut value ensures that the selected coreset is less redundant and has minimal overlap with the rest of the dataset. Therefore, the final coreset C can be obtained by using Eq. A.20 where k is the desired size of the coreset.

$$\mathrm{Cut}(C, V \setminus C) = \sum_{i\in C, j\in V\setminus C} s(x_i, x_j). \tag{A.19}$$

$$C = \arg\min_{|C|\le k} \mathrm{Cut}(C, V \setminus C), \tag{A.20}$$

Appendix A.3. Implementation Details

Backbones. To offer a more comprehensive evaluation, we test both from scratch and pre-trained models across two architectures: ResNet18 [39] and Vision Transformer (ViT) [40]. In **ResNet18** trained from scratch, we observe how well it can learn task-specific features directly from the dataset. In contrast, the pre-trained models **pre-trained-ResNet18** and **pre-trained-ViT** are initialized with ImageNet weights, giving them prior knowledge of visual patterns and structures, which helps them start with a robust foundation for CL.

Metrics. We utilize average accuracy (ACC) which measures the final accuracy averaged over all tasks and can be formulated as $ACC = \frac{1}{T}\sum_{i=1}^{T} A_{T,i}$ where $A_{T,i}$ represents the testing accuracy of task T after learning task i. To observe learning-forgetting dynamics more in detail, we utilize heatmaps that show the accuracy of each task after every learning session instead of sharing a single numerical value.

Appendix A.4. More Results and Analysis on Stability-Plasticity

We have discussed in the main paper that the increase in incremental performance is due to achieving a better stability-plasticity trade-off. In Section 5.2 our analysis on Split-CIFAR10 demonstrates that models trained with coreset samples tend to have better knowledge retention(stability) for previous tasks. Here we provide additional results on Split-CIFAR100 in Figure B.

Notably, the results show a pattern of improved accuracy when coresets are used, which aligns with observations made in the Split-CIFAR10 experiments. This accuracy boost can primarily be attributed to reduced forgetting, as training on a selected subset allows the model to retain important task information with less interference from previous tasks. Minimizing redundancy and focusing on coreset samples, provides a more targeted training approach and enhances overall model performance in class-incremental scenarios.

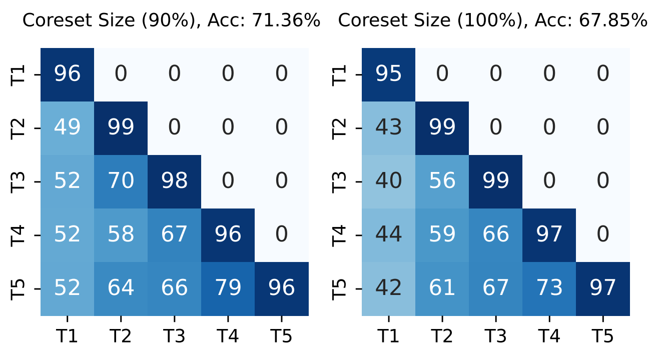
We provide further analysis for CIFAR-10 with a pre-trained ResNet18 (Figure A.a) and CIFAR-100 with a pre-trained ViT in Figure A.b) and observe improvements in stability without degrading plasticity. This suggests that pre-trained models, when fine-tuned on carefully selected coreset samples, are particularly well-suited for incremental learning.

Table A: Accuracy [%] of CL models across various coreset fractions and selections on **Split-CIFAR100** with **pre-trained ViT**. Underlined results outperform training with all samples and the best results are highlighted in bold.

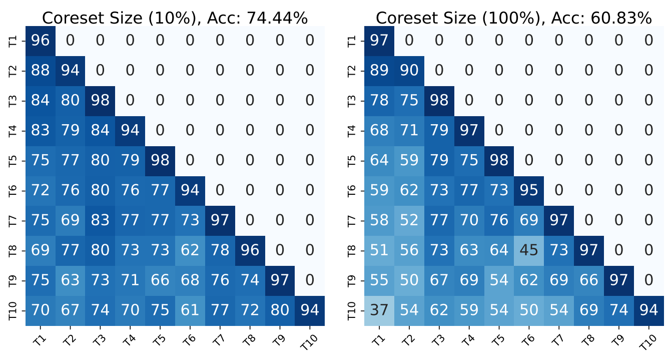
	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER	60.83 ± 1.93	Random	61.51 ± 0.36	61.88 ± 1.00	64.39 ± 0.78	63.12 ± 0.02	64.10 ± 1.22
		Herding	68.25 ± 1.44	69.26 ± 1.15	70.07 ± 0.15	68.58 ± 1.03	68.88 ± 1.92
		Uncertainty	**74.44 ± 0.37**	71.30 ± 0.42	69.68 ± 0.16	68.92 ± 0.37	70.28 ± 0.18
		Forgetting	70.70 ± 2.70	73.10 ± 0.55	69.92 ± 1.23	68.15 ± 0.63	68.05 ± 0.09
		GraphCut	72.58 ± 0.27	72.29 ± 0.03	69.70 ± 1.58	68.88 ± 1.47	68.37 ± 2.21
FOSTER	**86.74 ± 0.30**	Random	72.51 ± 2.67	81.41 ± 0.67	84.97 ± 0.56	85.91 ± 0.28	86.35 ± 0.42
		Herding	68.84 ± 0.01	78.87 ± 0.34	83.68 ± 0.23	85.41 ± 0.34	85.58 ± 0.27
		Uncertainty	77.10 ± 0.59	82.68 ± 0.26	85.17 ± 0.23	86.03 ± 0.12	85.83 ± 0.19
		Forgetting	77.00 ± 1.53	82.61 ± 0.30	84.90 ± 0.39	85.74 ± 0.33	86.03 ± 0.24
		GraphCut	74.64 ± 0.79	79.72 ± 0.42	84.14 ± 0.08	85.09 ± 0.13	85.68 ± 0.41
MEMO	**36.12 ± 0.16**	Random	14.84 ± 0.20	17.87 ± 0.90	23.74 ± 5.85	27.24 ± 5.37	30.07 ± 7.65
		Herding	27.79 ± 1.15	24.68 ± 1.79	28.22 ± 2.03	31.02 ± 0.66	30.07 ± 0.45
		Uncertainty	29.21 ± 1.47	29.34 ± 1.07	32.13 ± 0.76	31.88 ± 2.99	30.95 ± 0.10
		Forgetting	35.14 ± 1.79	31.72 ± 0.71	29.29 ± 1.46	31.47 ± 2.11	31.00 ± 2.94
		GraphCut	33.74 ± 1.66	32.46 ± 2.07	33.45 ± 3.05	30.67 ± 3.23	28.38 ± 2.63
iCaRL	66.03 ± 0.61	Random	71.24 ± 1.50	71.79 ± 2.62	70.62 ± 1.56	68.30 ± 1.72	68.79 ± 2.38
		Herding	68.34 ± 0.21	69.85 ± 0.25	71.11 ± 0.48	70.72 ± 0.41	69.09 ± 0.59
		Uncertainty	74.88 ± 0.41	74.11 ± 0.14	70.61 ± 0.13	70.99 ± 0.34	69.20 ± 0.45
		Forgetting	73.21 ± 0.58	73.51 ± 0.40	71.91 ± 0.76	70.74 ± 0.23	70.61 ± 1.03
		GraphCut	72.74 ± 4.08	**73.68 ± 1.78**	71.72 ± 0.52	73.05 ± 2.42	73.59 ± 2.28
ER	67.95 ± 0.86	Random	69.52 ± 2.83	73.54 ± 1.81	73.59 ± 0.10	73.36 ± 0.16	72.39 ± 0.52
		Herding	67.47 ± 1.53	70.57 ± 0.20	71.43 ± 1.43	72.65 ± 0.60	72.24 ± 0.16
		Uncertainty	73.97 ± 0.25	72.71 ± 1.94	71.68 ± 0.38	72.68 ± 0.84	70.31 ± 0.37
		Forgetting	71.32 ± 0.73	71.31 ± 0.24	71.50 ± 1.06	72.00 ± 0.45	72.09 ± 0.27
		GraphCut	**76.59 ± 0.35**	76.39 ± 1.68	74.87 ± 0.46	70.09 ± 0.25	70.69 ± 0.66
LwF	**66.63 ± 1.41**	Random	52.76 ± 2.27	60.26 ± 2.62	64.73 ± 1.56	65.71 ± 0.70	65.35 ± 0.85
		Herding	22.99 ± 0.13	24.44 ± 0.13	27.57 ± 0.49	29.46 ± 0.67	31.10 ± 0.40
		Uncertainty	25.17 ± 0.60	26.27 ± 0.31	28.78 ± 0.26	30.19 ± 0.64	30.31 ± 0.04
		Forgetting	24.99 ± 0.29	26.50 ± 0.18	27.63 ± 0.74	31.22 ± 0.77	30.52 ± 0.44
		GraphCut	23.32 ± 1.24	25.84 ± 0.98	29.53 ±1.47	29.66 ± 0.45	31.82 ± 0.75
CODA-Prompt	85.37 ± 0.79	Random	78.99 ± 1.42	81.62 ± 1.89	84.01 ± 0.11	84.64 ± 0.38	85.45 ± 0.44
		Herding	73.21 ± 1.23	74.31 ± 1.19	83.21 ± 0.72	85.51 ± 0.65	85.73 ± 0.01
		Uncertainty	78.48 ± 1.02	82.32 ± 1.01	85.20 ± 0.16	85.64 ± 0.37	85.57 ± 0.92
		Forgetting	78.30 ± 1.81	82.48 ± 1.19	84.73 ± 0.33	85.73 ± 0.98	86.33 ± 0.81
		GraphCut	80.55 ± 1.28	83.33 ± 1.16	84.31 ± 0.35	85.26 ± 0.38	**86.34 ± 0.26**

Table B: Accuracy [%] of CIL models across various coreset fractions and selections on **Split-ImageNet100**. Learning from coreset samples enhances the performance.

	w/o Coreset	Fraction	10%	20%	50%	80%	90%
DER [9]	55.03 ± 1.2	Random	19.89 ± 2.3	32.70 ± 1.5	42.45 ± 0.6	52.61 ± 1.8	53.12 ± 1.0
		Herding	18.30 ± 1.2	29.83 ± 0.6	44.77 ± 0.8	53.59 ± 0.3	55.52 ± 0.1
		Uncertainty	27.08 ± 0.5	36.92 ± 0.9	49.84 ± 0.4	55.10 ± 0.2	**56.46 ± 0.6**
		Forgetting	32.69 ± 2.1	40.21 ± 1.3	50.27 ± 0.9	55.15 ± 0.7	55.60 ± 0.8
		GraphCut	32.91 ± 0.7	38.90 ± 0.4	50.12 ± 0.8	54.71 ± 0.3	55.81 ± 0.1
FOSTER [10]	**52.06 ± 0.4**	Random	17.59 ± 1.3	22.68 ± 0.8	34.20 ± 3.8	46.90 ± 4.1	48.64 ± 4.2
		Herding	8.67 ± 0.1	13.42 ± 0.2	30.63 ± 1.7	45.85 ± 1.0	48.89 ± 0.1
		Uncertainty	8.14 ± 0.1	15.91 ± 0.5	35.40 ± 0.5	46.39 ± 0.5	48.37 ± 0.5
		Forgetting	11.62 ± 0.5	18.71 ± 0.4	35.26 ± 0.3	46.95 ± 0.9	49.45 ± 0.4
		GraphCut	16.74 ± 0.5	22.99 ± 0.1	37.42 ± 0.4	47.22 ± 0.4	49.95 ± 0.9
MEMO [11]	46.36 ± 1.0	Random	18.79 ± 0.1	27.29 ± 0.2	40.02 ± 1.7	44.48 ± 0.2	47.80 ± 1.9
		Herding	18.15 ± 1.1	26.08 ± 0.4	37.71 ± 3.1	46.76 ± 2.3	47.94 ± 1.1
		Uncertainty	20.22 ± 0.8	26.94 ± 2.2	39.39 ± 1.1	45.90 ± 0.4	**48.54 ± 0.2**
		Forgetting	24.40 ± 1.5	33.16 ± 1.0	41.86 ± 0.5	45.57 ± 0.5	47.19 ± 0.9
		GraphCut	29.76 ± 1.8	35.73 ± 1.1	42.80 ± 1.9	45.98 ± 2.8	48.50 ± 1.3
iCaRL [21]	33.05 ± 1.8	Random	21.93 ± 0.7	27.29 ± 0.5	30.21 ± 3.7	29.12 ± 1.9	30.30 ± 1.6
		Herding	20.80 ± 1.8	24.29 ± 2.3	30.92 ± 0.2	33.23 ± 0.9	34.04 ± 0.2
		Uncertainty	22.52 ± 0.3	22.37 ± 0.9	32.67 ± 1.6	33.03 ± 0.1	34.76 ± 0.9
		Forgetting	26.38 ± 0.1	28.35 ± 0.8	31.85 ± 0.7	33.80 ± 0.6	34.77 ± 2.7
		GraphCut	33.04 ± 0.6	35.10 ± 0.6	34.87 ± 1.1	**35.19 ± 0.2**	34.29 ± 0.3
ER [22]	34.23 ± 4.2	Random	20.19 ± 0.1	25.84 ± 2.7	30.47 ± 2.0	29.14 ± 1.0	30.81 ± 0.6
		Herding	20.21 ± 0.1	24.56 ± 0.8	29.81 ± 1.1	31.92 ± 0.4	33.68 ± 0.7
		Uncertainty	20.82 ± 0.6	23.08 ± 0.6	29.23 ± 0.5	29.35 ± 1.1	30.74 ± 1.4
		Forgetting	24.85 ± 0.6	28.32 ± 1.4	29.03 ± 0.2	32.85 ± 0.4	31.74 ± 2.1
		GraphCut	30.13 ± 1.0	30.52 ± 0.2	**34.83 ± 0.6**	32.05 ± 1.5	32.16 ± 0.5
LwF [3]	16.46 ± 1.8	Random	9.25 ± 0.1	11.22 ± 0.7	15.88 ± 0.8	16.27 ± 1.1	**16.52 ± 0.5**
		Herding	5.70 ± 0.5	7.65 ± 1.1	10.70 ± 0.1	11.33 ± 0.2	11.64 ± 0.2
		Uncertainty	7.84 ± 0.1	8.07 ± 0.1	11.27 ± 0.2	11.41 ± 0.1	11.51 ± 0.3
		Forgetting	7.38 ± 0.2	10.01 ± 0.1	11.60 ± 0.2	12.15 ± 0.1	12.57 ± 0.3
		GraphCut	7.41 ± 0.2	9.29 ± 0.8	10.77 ± 0.5	12.06 ± 0.2	12.88 ± 0.1

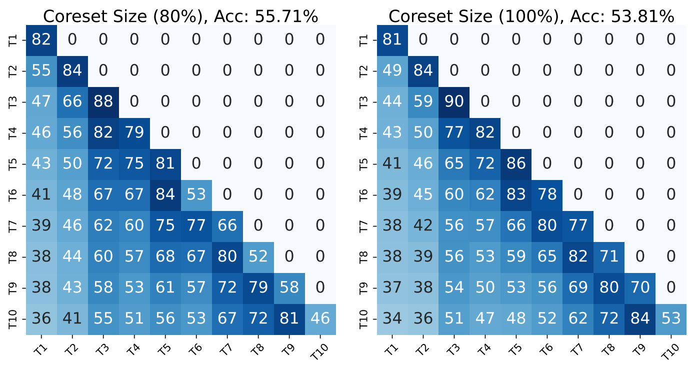


(a) DER with Herding on Split-CIFAR10

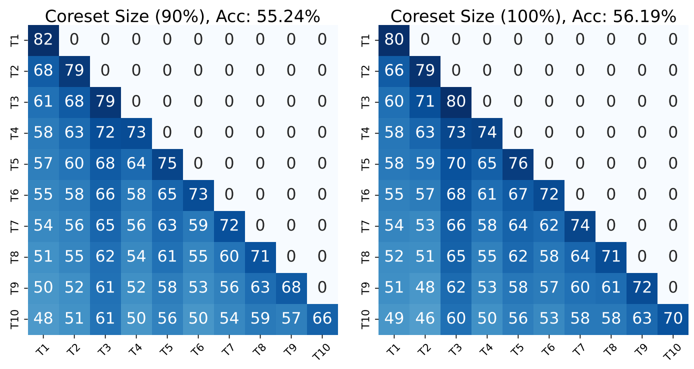


(b) DER with Uncertainty on Split-CIFAR100

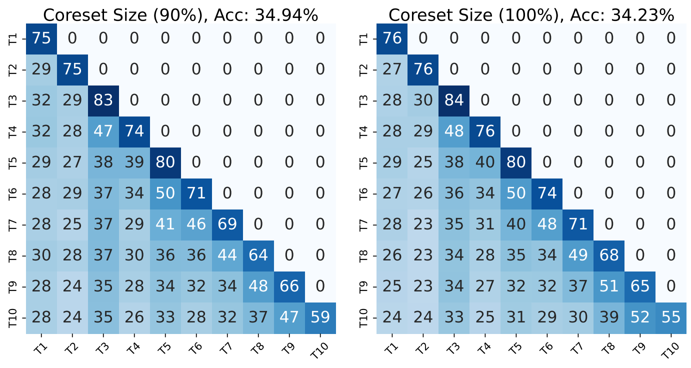
Figure A: Accuracy [%] of each task after every learning session on the DER method with pre-trained ResNet18 on Split-CIFAR10 and pre-trained ViT on Split-CIFAR100. The results indicate that incremental performance improves due to better stability without compromising plasticity.



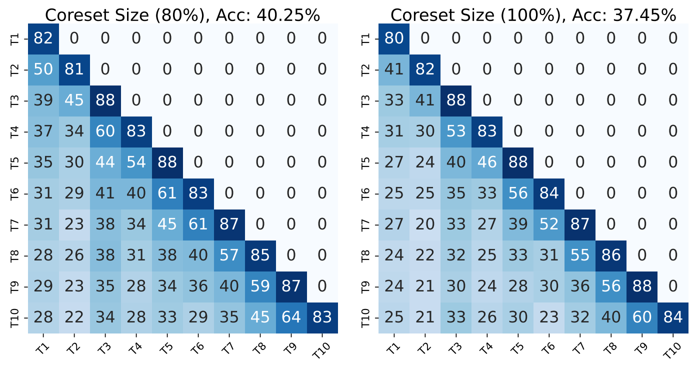
(a) DER



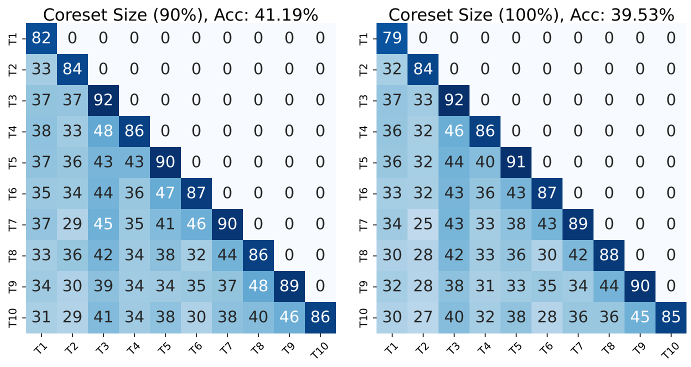
(b) FOSTER



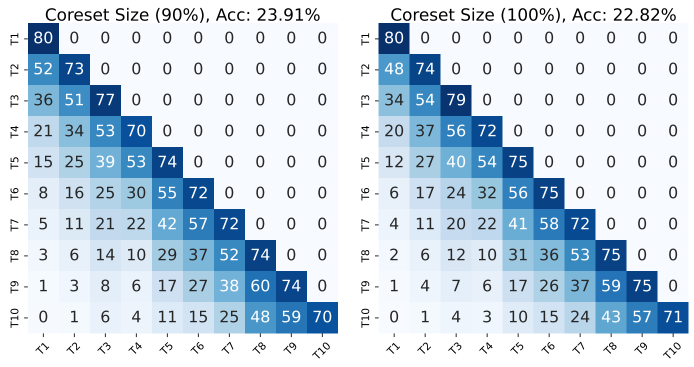
(c) MEMO



(d) iCaRL



(e) ER



(f) LwF

Figure B: Accuracy [%] of each task after every learning session on different class-incremental learning methods with Split-CIFAR100. Its results align with Split-CIFAR10 and again incremental performance improves due to better knowledge retention.