Examining Changes in Internal Representations of Continual Learning Models Through Tensor Decomposition

Nishant Suresh Aswani[*,†] **Amira Guesmi**[†] **Muhammad Abdullah Hanif**[†] **Muhammad Shafique**[*,†]
[*]Department of Computer Science and Engineering, New York University Tandon, Brooklyn, USA
[†]eBrain Lab, Division of Engineering, New York University Abu Dhabi, Abu Dhabi, UAE

Abstract

Continual learning (CL) has spurred the development of several methods aimed at consolidating previous knowledge across sequential learning. Yet, the evaluations of these methods have primarily focused on the final output, such as changes in the accuracy of predicted classes, overlooking the issue of representational forgetting within the model. In this paper, we propose a novel representation-based evaluation framework for CL models. This approach involves gathering internal representations from throughout the continual learning process and formulating three-dimensional tensors. The tensors are formed by stacking representations, such as layer activations, generated from several inputs and model 'snapshots', throughout the learning process. By conducting tensor component analysis (TCA), we aim to uncover meaningful patterns about how the internal representations evolve, expecting to highlight the merits or shortcomings of examined CL strategies. We conduct our analyses across different model architectures and importance-based continual learning strategies, with a curated task selection. Often, the results of our approach mirror the difference in performance of various CL strategies on various architectures. Ultimately, however, we found that our methodology did not directly highlight specialized clusters of neurons, nor provide an immediate understanding the evolution of filters. We believe a scaled down variation of our approach will provide insight into the benefits and pitfalls of using TCA to study continual learning dynamics.

Proceedings of the 1st ContinualAI Unconference, 2023, Virtual. PMLR Volume 249.

1 INTRODUCTION

Learning is a core capability for all intelligent agents. While biological agents acquire new knowledge and adapt by building upon their prior learning experiences, artificial agents follow a more static and meticulously curated learning process. When challenged to encounter new concepts, a setting familiar to biological agents, machine learning models suffer from *catastrophic forgetting*, demonstrating poor performance when attempting to recall old patterns (Schlimmer and Fisher, 1986; Ring, 1997). As continual learning (CL) research matures in tackling this challenge, solutions will look increasingly composite, likely layering multiple mechanisms to mitigate forgetting and achieve additional goals (Kudithipudi et al., 2022). The growing complexity underscores the need for an architecture and strategy agnostic explainability tool designed to shed light on how CL methods enable models to acquire new tasks while preventing the forgetting of previously learned ones. The insights gained from studying how representations change when experiencing catastrophic forgetting has previously facilitated the advancement of new methods. Thus, better understanding how existing methods update model parameters over time holds the potential to inspire the development of even more effective learning strategies.

To execute our study, we propose a novel framework that allows us to explore internal state changes during the continual learning process. Our framework centers on creating a data representation that captures the model's internal representations across different tasks over time. By leveraging Tensor Component Analysis (TCA) (Williams et al., 2018), a technique for three-dimensional tensor decomposition, we expect to uncover patterns concerning internal representations across time. The proposed methodology, as depicted in Figure 1, provides an overview of our study in understanding how representations evolve during CL.

In summary, the contributions and the key insights that can be derived from this analysis are:

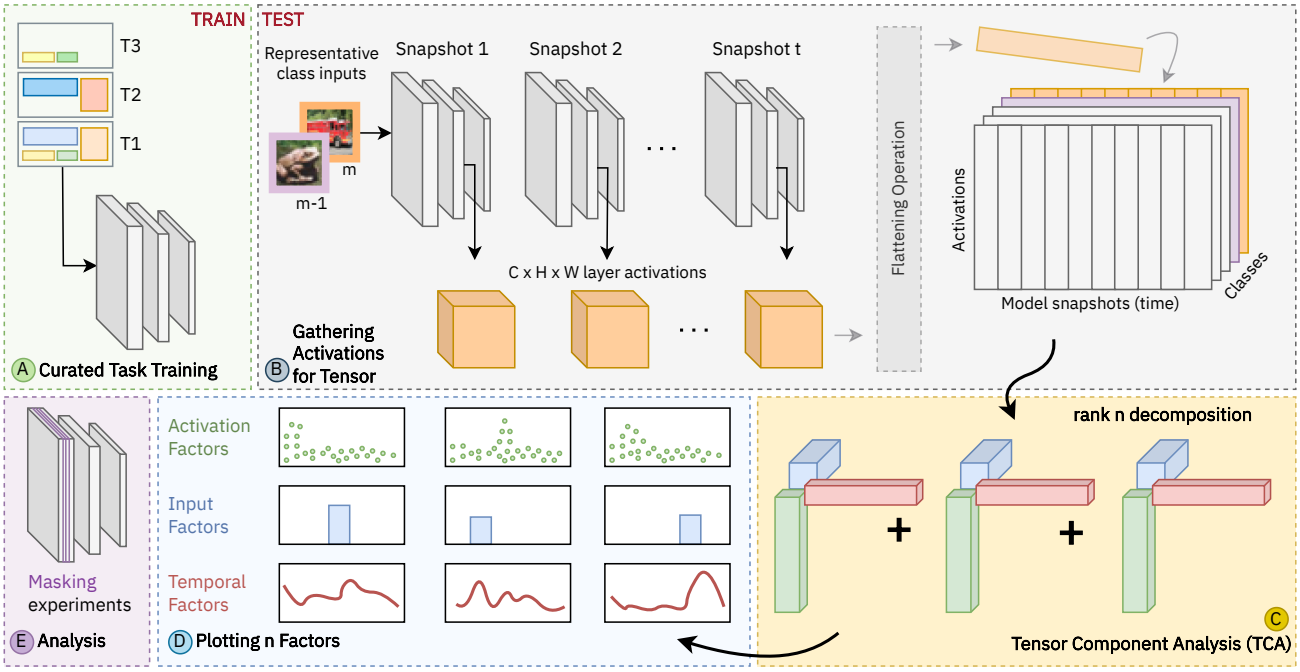

Figure 1: Overview of the proposed methodology. (A) We train a model with a CL method with curated tasks. (B) We assume access to 'snapshots' of the model taken throughout training on all tasks. At inference time, for a given dataset class, we feed its corresponding representative inputs into each of the snapshots and, for a layer of choice, gather the resulting activation tensors. We flatten the activations into a vector and stack them for all snapshots to obtain a matrix. We repeat this process for all inputs and stack the matrices to obtain a tensor. (C) We conduct tensor component analysis (TCA) with rank n to obtain n components. (D) We plot n components, each of which consists of three factors: factors selecting for certain activations (green), factors selecting for certain inputs (blue), and factors describing a temporal activity (red). (E) We conduct model masking experiments to verify our observations.

- We leverage TCA, an unsupervised method for analyzing three-dimensional tensors, to extract patterns about how model representations evolve in a continual learning setting. TCA results in a data representation that potentially captures the model's dynamics for multiple inputs over time. To the best of our knowledge, we are the first to leverage TCA in continual learning interpretability.
- We systematically compare the performance, and analyze the internal representations of several parameter-based methods (and their combinations with replay) across convolutional neural networks (CNNs), vision transformers, and a CNN-transformer hybrid architecture. In addition, we conduct these experiments with a curated set of tasks to study how internal representations change based on the feature similarities of tasks.
- We investigate neuron tracking and filter evolution which can shed light on the neural plasticity that occurs during continual learning. We plan to gain insights on which neurons or filters are more flexible and likely to change their responses over time, and which ones remain stable.

Summary of Hypotheses:

Hypothesis Set 1. Do importance-based regularization methods lead to the emergence of 'specialized' neurons for specific tasks? Does adding replay reinforce this specialization? Is there a consistent behavior across several model architectures? By tracking activations throughout training with various CL strategies, we will study the temporal patterns of these activations.

Hypothesis Set 2. Do CNN filters or transformer features within the same layer exhibit different update patterns throughout training? How do these update patterns compare by CL strategy and model architecture? By tracking what CNN filters and transformer features select, we will examine how filters/features evolve throughout the task-incremental learning.

2 BACKGROUND & RELATED WORK

2.1 Techniques for Examining Representation Quality

By examining representations, one may assess how well a model's learned features generalize across tasks and the

degree to which they preserve task-specific information. Strong representations may enable better knowledge transfer and improved adaptation to new tasks. In this section, we explore several works studying representations in the context of CL. These studies employ various metrics and experimental scenarios to evaluate the quality of learned representations during training.

Ramasesh et al. (2020) studied the impact of catastrophic forgetting on hidden representations by quantitatively comparing the layer representations before and after sequentially training on a second task. To measure the similarity between representations across multiple layers of the model, they utilized the centered kernel alignment (CKA) technique. They also investigated how the semantic similarity between tasks affected forgetting, obtaining nuanced results suggesting that forgetting is maximized when sequential tasks have intermediate rather than low or high similarity.

Davari et al. (2022a) evaluated the quality of representations using Linear Probes (LP), which involves training a linear classifier on the fixed representations for each task. The accuracy of this linear classifier on the task's test set served as a metric to assess the representation quality for that particular task. To measure forgetting, the authors measured the difference in linear probe performance before and after introducing a new task.

Zhang et al. (2022) created a synthetic dataset in which alternating (odd and even) tasks shared the same low-level features while each task simultaneously contained unique, high-level features. By having access to the 'ground-truth' features, the authors examined whether the model was making progress towards learning the features encoded in the inputs. Their findings revealed that when the shared feature was consistently encountered in all even-numbered tasks, it prevented the model from fully learning the shared feature present in the odd-numbered tasks. The same held true for learning the shared feature in even-numbered tasks.

Hess et al. (2023) introduced a task exclusion comparison, hypothesizing that if a model has trained on a task and retained knowledge related to that task, then it should exhibit a more robust representation of that task compared to a model that has never encountered the task. To test this, they compared the linear probe accuracy of models that have encountered a task with those that have had the same task deliberately excluded. Their results indicate that continually trained models typically forget task specific knowledge quickly, contrary to what was presented by Davari et al. (2022a).

Across most works studying layer representations in the context of continual learning, CKA and linear probes make a recurring appearance as tools to measure the similarity of hidden layer activation patterns in neural networks. Although these metrics offer a broad understanding of representation comparisons, the occasional conflicting results among existing literature using similar experimental setups highlights the need for a different approach to studying CL dynamics. Recent representation similarity work has even highlighted the sensitivity of CKA to outliers (Davari et al., 2022b), urging researchers to ensemble similarity measurement using several techniques.

Nevertheless, most analyses of network representations in the context of CL limit themselves to similarity measurements of network representations pre and post-task training. We believe there is an opportunity to move away from drawing conclusions about network representations from ambiguous similarity measures when attempting to understand catastrophic forgetting. Instead, we suggest that it is possible to take advantage of unsupervised tensor decomposition to study how internal representations evolve across time for several inputs.

2.2 Tensor Component Anlysis (TCA) for Exploring Learning Dynamics

Tensor component analysis (TCA), also known as canonical polyadic (CP) decomposition, is a tensor decomposition technique to identify variability across the three axes of a tensor (Carroll and Chang, 1970; Harshman et al., 1970). While it is a dimensionality reduction technique similar to Principal Component Analysis (PCA), TCA differs by extending the decomposition to an additional axis. Further, unlike PCA, the factors obtained by TCA are not necessarily orthogonal, allowing for an expression of more natural patterns. For PCA, unless the features present in the data are naturally orthogonal to each other, the components recovered cannot be interpreted as those underlying features. In addition, PCA yields multiple solutions that can be employed to reconstruct the original data, known as the rotation problem (Williams et al., 2018). In practice, TCA does not face this limitation. While TCA is not entirely immune to alternative solutions, the number of potential solutions is more constrained, and in the case of non-negative TCA, the solution is typically unique (Kruskal, 1977; Adali et al., 2022). In various contexts within neural statistics, Williams et al. (2018) demonstrated that TCA serves as an unsupervised technique capable of demixing neural data and providing interpretable results corresponding to agent behavior and learning patterns.

While Williams et al. (2018) discussed TCA as a method to improve neural data analysis, such that it describes trial-to-trial variability and avoids a trial-averaging step, McGuire et al. (2022) took advantage of TCA in a slightly different fashion. Studying the behavior of neurons in the postrhinal cortex (POR) of mice, the authors attempted to identify neuron population clusters, as well as the clusters' responses over time to various cues displayed to the mice as they sequentially learned two tasks. Rather than focusing

on trial-to-trial variability, the study focused on identifying within-stage and across-stage dynamics for multiple cues. In this context, 'stage' refers to the learning stages across two tasks. Through TCA, the researchers observed a 'division of labor' in the neural populations, showing that certain clusters of neurons are specialized to activate in response to particular cues. Moreover, they were able to capture how these clusters vary their response across time.

Dyballa et al. (2023) used TCA as part of their analyses to organize neurons into a 'manifold' to study how the neurons they measured lie with respect to each other in a lower dimensional space. Generating these manifolds for biological and artificial neurons allowed the authors to compare the primary visual cortex (V1) to a CNN by studying the differences in how neurons are encoded within their manifolds. To accomplish this, the authors first employed TCA to obtain components which formed a lower dimensional space. Then, this component space was transformed into a manifold using the IAN similarity kernel and diffusion maps. The authors claimed that their analyses allowed them to organize neurons accounting for both stimulus features and temporal response, rather than the conventional approach of solely focusing on stimulus selectivity. In essence, the authors highlighted the ability of TCA to unlock an additional dimension, as well as provide outputs that can be fed into further analyses for network comparisons.

3 PROPOSED METHODOLOGY

Through tensor component analysis (TCA), we plan to investigate how the internal representations of CL models evolve as the models train on incoming tasks. As suggested earlier, TCA unlocks an opportunity to ask questions about learning patterns that might emerge in the temporal dimension while looking at several inputs. The main questions that we will explore in this investigation are:

- Are continual learning strategies able to elicit neuron specialization as they encounter new tasks?
- How do convolutional filters and transformer features shift over time when encountering new data in the continual learning setting?

For this exploration, we plan to use combinations of several CL strategies and model architectures. We look towards insights provided by simple explainability techniques to construct an understanding of continual learning dynamics.

3.1 Overview

Our proposed framework (see Figure 1) involves sequentially training a model on a stream of curated tasks (see Section 4) and analyzing its representations. To achieve this, we adopt the following steps:

Saving Model Snapshots During the continual training process, we periodically save snapshots of the model at defined intervals. For all tasks, we save multiple snapshots of the model while it is still learning the task, in order to conduct within-task and across-task analyses.

Probing Model Snapshots Then, we probe each model snapshot. This process involves feeding data to the model and recording, for example, the activations from a designated layer within the model. Activations represent the outputs of a layer in response to the given inputs. The neural activations collected during this process are structured into a three-dimensional tensor. As we will later elaborate, our analysis extends beyond neural activations.

Tensor Component Analysis Through tensor decomposition, we hypothesize deriving interpretable components. These components can, for instance, provide descriptions of how specific neural activations evolve throughout the learning process. Such analysis holds the potential to offer valuable insights into how changes in activations over time might correspond to the model's performance.

3.2 Problem Setup

We begin with the standard setup of a CL problem, as described by Hess et al. (2023), where we assume an incoming stream of classification tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_i, ..., \mathcal{T}_t\}$. Each task consists of a set of images X_i with their corresponding class labels Y_i. In the case of a multi-head classifier, the model also has access to the unique task identifier i. Section 4 discusses how we curate our tasks to aid the analysis. For a chosen CL strategy, we train a model f_θ sequentially on these tasks. For the purposes of a temporal analysis, we save a snapshot of the model parameters $\theta_{i,e}$ at defined intervals, where i, e refers to the parameters of a model learning a task $\mathcal{T}_i$ after having completed training epoch e. For instance, if we train a model for 50 epochs on each of 3 tasks and take snapshots at a frequency of 10 epochs, we would obtain a set of snapshots $\theta = \{\theta_{1,10}, \theta_{1,20}, ..., \theta_{3,40}, \theta_{3,50}\}$.

3.3 Tensor Formulation Details

One way to probe a model snapshot $\theta_{i,e}$ is with an input m to obtain network activations $A^m_{i,e}$ from a desired model layer. We can repeatedly probe that layer with n different inputs, to obtain n different activations $A_{i,e} = \{A^1_{i,e}, A^2_{i,e}, ..., A^n_{i,e}\}$. If we flatten the neural activations, we will obtain a two-dimensional matrix of neural activations for multiple inputs. As shown in Figure 1, by repeating this procedure for all available model snapshots, we can build a three-dimensional tensor containing the neural activations across time for several inputs. Along one dimension of the tensor, one would vary the model snapshot $\{i, e\}$, and along another one

Table 1: **A Selection of Strategies Based on Parameter Importance**

Approach	Strategy	+ ER?	Ref.
Baselines	Naive	✗	
	Cumulative	✗	
Replay	Experience Replay (ER)	✗	
Importance-Based Regularization	Elastic Weight Consolidation (EWC)	✓	Kirkpatrick et al. (2017)
	Memory Aware Synapses (MAS)	✓	Aljundi et al. (2018)
Importance-Based Subnetworking	Relevance Mapping Networks (RMN)	✓	Kaushik et al. (2021)
	Winning Subnetworks (WS)	✓	Kang et al. (2022)

would vary the input m. To gather neural activations for this analysis, it is essential to feed meaningful input into the model. Future sections discuss potential strategies for selecting inputs. However, we emphasize that our analysis extends beyond neural activations.

3.4 Tensor Component Analysis

TCA approximates a three-way tensor X as a sum of rank-1 tensors, where each rank-1 tensor is an outer product of vectors. The three-way tensor can be expressed as follows (Williams et al., 2018):

$$\hat{X} \approx \sum_{r=1}^{R} (\mathbf{u^r} \otimes \mathbf{v^r} \otimes \mathbf{w^r}) \tag{1}$$

In the formulation above, the vector $\mathbf{u^r}$ might represent patterns in neural activations, $\mathbf{v^r}$ represents the inputs corresponding to this activity, and $\mathbf{w^r}$ represents the stages of task learning where this activity is present. The rank R, a hyperparameter for this technique, determines the total number of components that approximate the original tensor.

The optimization objective for TCA aims to minimize the reconstruction error between the original tensor X and its approximation $\hat{X}$, defined by the Frobenius norm:

$$\text{minimize} \quad ||X - \hat{X}||_F^2 \tag{2}$$

Additionally, the objective may be subject to non-negativity constraints on the component vectors $\mathbf{u^r}$, $\mathbf{v^r}$, and $\mathbf{w^r}$, i.e., all elements in these vectors are non-negative.

We will select a low rank R with reasonable reconstruction error, as described in Section 4.

3.5 Activation Tracking

Hypothesis 1: Does tracking neural activations throughout a continual learning setting, which constrains parameter changes based on an importance measure, reveal specialized classes of neurons?

Importance-based methods for CL, such as Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017), measure the importance of each parameter for a particular task and discourage certain parameters from significant changes throughout training or find meaningful masks for specific tasks. In our experiments, we investigate whether these strategies ultimately result in the emergence of sets of neurons that exhibit specialization for particular tasks. To explore this, we build a tensor with dimensions: [flattened neural activations] x [representative input(s) corresponding to dataset classes] x [model snapshot (time)]. We propose to conduct TCA on this tensor, thus expecting to capture patterns in neural activation of all the selected classes in a dataset across time.

In order to select the representative input(s) per class for which we will capture neural activations, we will experiment with the following approaches:

Random Sampling We will randomly select 20 images per class from the test dataset as the class representative inputs.

Maximally Activating Example We will search the test dataset for an image that maximally activates the desired class in the final model snapshot associated with the task that includes the class. For instance, if class 8 is a part of task $\mathcal{T}_t$ trained over e epochs, we would seek the test image that maximally activates the probability of class 8 in the model snapshot $\theta_{t,e}$.

Class Optimized Image We will optimize an image for a fixed number of iterations, such that the resulting image maximizes the probability of a particular class, as described by Olah et al. (2017) for convolutional neural networks

(CNNs).

When examining a TCA component, we hypothesize that specialization would be identified as a cluster of neurons characterized by greater activations. These activations would be associated with a segment of the temporal factors and one or few distinct inputs. This would indicate that certain neurons exhibited heightened activity during particular learning stages when the model was exposed to specific inputs.

3.6 Filter Evolution

Hypothesis 2: Do CNN filters or transformer features within the same layer demonstrate equal levels of activity in their evolution throughout the continual learning process?

In this experiment, we wish to understand how individual CNN filters and transformer features for a chosen layer evolve across the training regime. We ask if certain filters/features update earlier in the training regime and others update later or if filters/features are constantly changing. Specifically, do importance-based methods induce a markedly distinct updating pattern when compared to a purely replay-based approach or a baseline approach? Here, we build our tensor using dimensions: [flattened optimized image] x [filters/features] x [model snapshot (time)].

In contrast to the previous tensor formulation, this analysis will not be conducted for any specific class. Instead of examining multiple classes, we will focus on studying all the filters or features within a selected layer of the model. Additionally, rather than relying on neural activations, our approach involves utilizing optimized images specifically designed for each filter (in the case of CNNs) or feature (in the case of transformers) within a given layer. This optimization process is akin to the method of generating images that maximally activate a target class. In our case, we optimize images to achieve maximum activation for a chosen filter or feature within a model layer. This optimization procedure is inspired by the work of Olah et al. (2017) for CNN architectures and Ghiasi et al. (2022) for transformer architectures.

In this experiment, we may observe a scenario where a specific component exhibits the following characteristics: it selects one particular filter in the filter factors, corresponds to a particular segment in the temporal factors, and encompasses a subset of pixels within the 'flattened image' factors. Such a result could imply that this specific filter became actively engaged after learning a certain task. The temporal factors would indicate when it became active, the filter factors would specify which filter was active, and the flattened image factors would reveal which region was active during this process.

4 EXPERIMENTAL PROTOCOL

Datasets We propose to run our experiments on the following classification datasets: (i) SplitMNIST, (ii) SplitCIFAR10, (iii) SplitCIFAR100, and (iv) twenty CIFAR100 superclasses (Ramasesh et al., 2020). The proposed settings cover a variety of task complexities. Across all comparable experiments, the dataset splits and task orders will be consistent to ensure a fair comparison.

Task Generation and Order Influenced by the approach of Ramasesh et al. (2020) and Zhang et al. (2022), we wish to apply our framework in a controlled setting by curating the order of tasks. Since we are interested in studying how learned features evolve, we propose that the initial task be large enough to learn rich features. We hypothesize that smaller initial tasks lead to poor initial model representations, making it difficult to draw meaningful conclusions. For instance, using Split-CIFAR10, the initial task may consist of the following four classes: airplane, automobile, cat, and horse, allowing the model to 'generalize.' We can then curate the next task to either be a two-way classification between deer/dog (animal) or truck/ship (vehicle). We can conduct a similar task curation for SplitMNIST and SplitCIFAR. In order to meaningfully curate these tasks, we propose generating tSimCNE embeddings of the datasets, providing us with a coarse understanding of which classes might share features. Then, for instance, we can select dissimilar classes to consist the initial task and experiment with the remaining classes for the next tasks.

Strategies To study how various CL strategies affect model weights, this work will focus on the strategies outlined in Table 1. The selected strategies all exploit parameter importance to improve incremental learning, likely being an interesting class of strategies to study for our hypotheses. The final column describes whether there will be an additional variant of the strategy where it is combined with experience replay. We expect to finalize precise hyperparameters through a rigorous grid search.

Model Architectures We aim to study three architectures, selecting variants with a similar number of parameters: (i) ResNet-50 (23M parameters) (He et al., 2016), (ii) DeiTSmall (22M parameters) (Dosovitskiy et al., 2020), and (iii) Convolutional vision Transformer CvT13 (20M parameters) (Wu et al., 2021). For the experiments outlined in Sections 3.5 and 3.6, we will use multi-head models and provide task identities during the inference phase. We will train all models with the SGD optimizer using a batch size of 256 images and a fixed learning rate, which will be determined for each architecture by tracking the most accurate model on the validation sets of the scenarios. We expect to train

models for a varying number of epochs depending on the architecture and/or the dataset (e.g. 40 epochs for Split-CIFAR-10). We will ensure that certain model training parameters (e.g. number of epochs) are constant across strategies when comparing experimental results. We will equivalently seed models to ensure they are initialized with the same weights.

Hyperparameter Selection Experiments We plan to conduct extensive grid search experiments for hyperparameter selection, similar to those conducted by van de Ven et al. (2022), to ensure that the selected hyperparameters for the main experiments result in models with comparable performance across tasks. The primary focus of this work is to explore how internal representations are affected by the choice of continual learning strategy. Therefore, we will minimize explorations on the effects of various hyperparameters, instead aiming to pick a fair set of hyperparameters that will ensure reasonable and comparable model performance for a given architecture and dataset across continual learning strategies.

TCA Model Optimization We will explore fitting our TCA models with the nonnegative hierarchical alternating least squares (HALS) (Cichocki et al., 2007) and nonnegative block coordinate descent (BCD) algorithms (Lee and Seung, 2000), with the expectation to select the optimization algorithm that obtains the lowest minima solution for each experiment. We will remain consistent on our selection for a given architecture and dataset. We expect the nonnegative variant of TCA to provide the most interpretable results.

TCA Rank Selection Adopting from how McGuire et al. (2022) utilized TCA, for each experiment, we plan to fit TCA models across a range of ranks (e.g., 1-20) and plot the errors. We will empirically determine a narrower range of ranks where the "elbow" in the plot lies, essentially looking for the lowest number of ranks that provide reasonable error. For each rank within the determined "elbow-range", we will fit 10 TCA models and compare the similarities between the components returned. For the final results, we will select the lowest rank that consistently returns a high similarity (above 0.8) (Williams et al., 2018).

Evaluation and Metrics For all experiments, we plan to report the average of the final classification accuracy. We recognize that our framework is skewed towards an empirical analysis, as we expect to extract activation and filter update patterns from plotting the resulting components after a tensor decomposition. Given that our hypotheses seek some level of specialization within the model, we propose to run masking experiments and measure the change in final accuracy to validate our findings from the TCA models. To elaborate, we can randomly mask certain filters in a CNN layer and measure the change in output accuracy. If our analysis reveals that specific filters hold significance for a particular class, the subsequent masking experiment could quantify the practical usefulness of our observation.

To compare the components from two TCA models, we will utilize the similarity score proposed in (Williams et al., 2018), which first solves an assignment problem to match components between two models and finds the 'optimal' permutation of components. Then, it computes the dot product between the matched components and returns the mean as a similarity score.

5 CHANGES TO THE PROTOCOL

Strategy Choice. We initially proposed studying the AGS-CL and SI strategies with our method. With regards to the former, we learned that the AGS-CL strategy is not compatible with architectures that include residual streams. To maintain consistency in our approach, we decided to remove this strategy from our study, as it could not be applied to the architectures outlined in Section 4. For the latter, given that we studied the very similar EWC and MAS strategies, we found the inclusion of SI to be redundant and uninformative.

Dataset. While the initial proposal included using the Split-MNIST dataset, we decided against its inclusion, due to the dataset's overly simple nature for our selected architectures.

Representative Input Selection. Our experiments selected representative inputs by finding maximally activating examples, as outlined in Section 3.5. Originally, we proposed additional methods for selecting representative inputs, which involved randomly selecting images from a class as well as optimizing noise to maximize the output logits for a particular class, originally described by Olah et al. (2017). We did not pursue these methods. We discovered that the latter was an especially computationally intensive procedure and determined that it was not the focus of our study.

6 RESULTS AND DISCUSSIONS

In this section, we present the results of our experiments along with analyses and discussion.

6.1 Experimental Setup

Hardware Configuration. Our models were trained on a machine equipped with NVIDIA RTX A6000 GPUs, featuring 48GB of GPU memory.

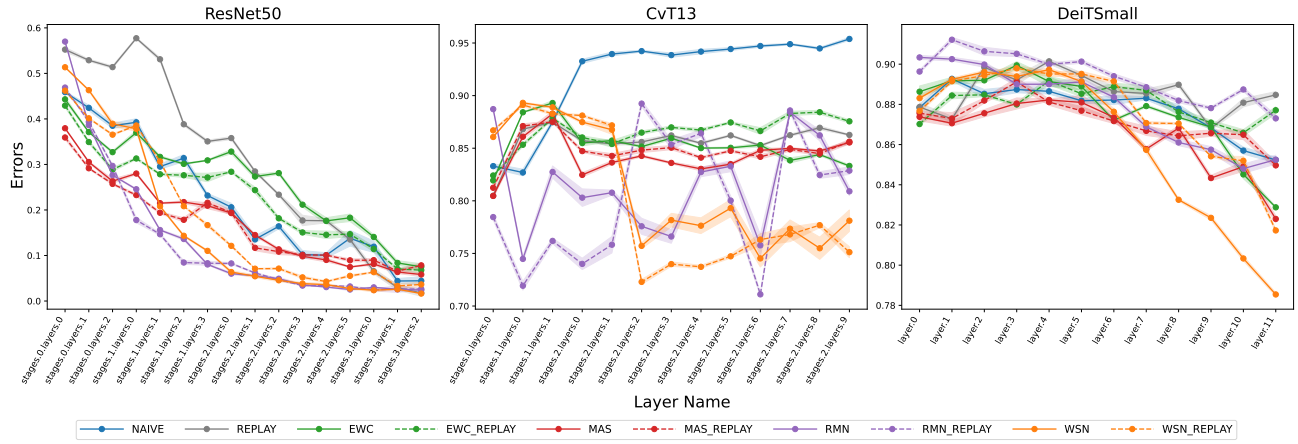

Figure 2: Layer-wise reconstruction errors with shaded standard deviations for ResNet50 (*left*), CvT13 (*middle*), and DeiTSmall (*right*) models. **Note:** The y-axis scaling varies between plots for clearer visualization.

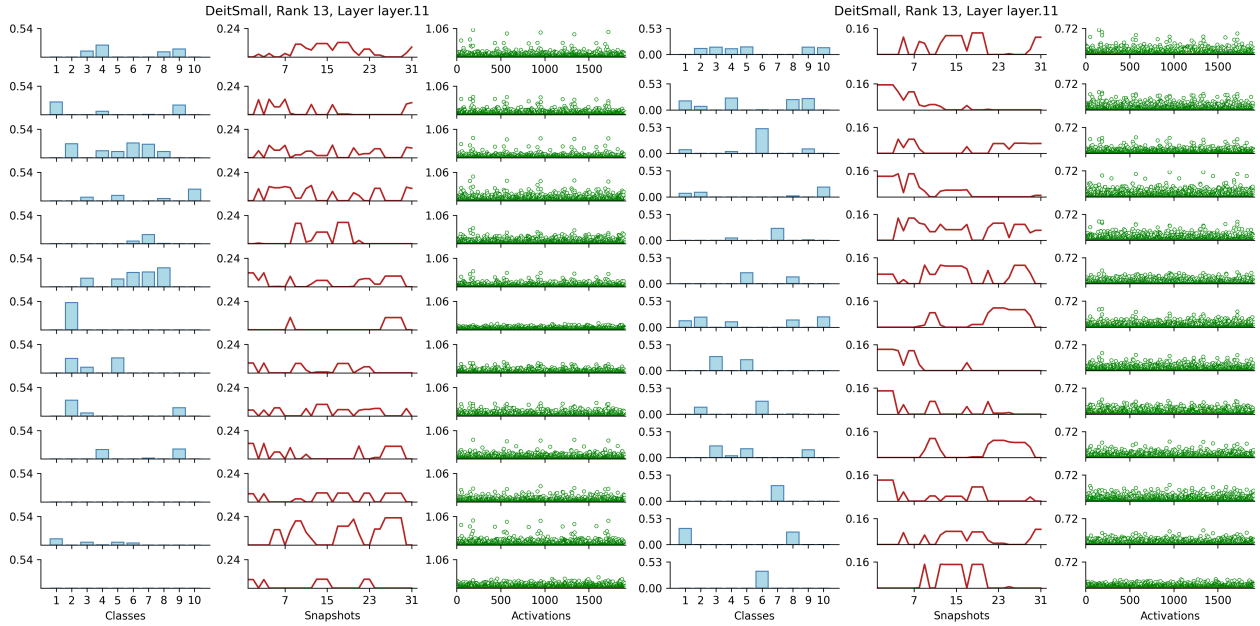

Figure 3: Example of tensor component analysis on an activations tensor: two rank-13 TCA plots of activation tensors from DeitSmall trained on Split-CIFAR-10, permuted to order factors that are best aligned. The similarity score between the two is 0.58. *Left*-TCA plot of activations from the MAS strategy, with a reconstruction error of 0.82. *Right*-TCA plot of activation from the MAS Replay strategy, with a reconstruction error of 0.85. Despite the similar performance of the TCA models, as demonstrated in Figure 2, the TCA models do not have a high similarity.

Tasks Generation. To curate the first task (i.e., Task 1) for each dataset, we used tSimCNE (Böhm et al., 2023) embeddings in conjunction with the QuickHull (Barber et al., 1996) algorithm to select a specific number of classes. A detailed description of this process is provided in Appendix A.1. For the Split-CIFAR-10 benchmark, we selected 4 classes: automobile, airplane, frog, and dog. For Split-CIFAR-100, we selected 10 classes: sea, cloud, aquarium fish, sunflower, lion, raccoon, girl, pickup truck, wardrobe, and chair. For Split-CIFAR-100 superclasses, we selected 4 classes: large outdoor natural scenes, large carnivores, people, and household furniture. The remaining classes were evenly and randomly distributed among the remaining tasks.

Model Training and Hyperparameter Optimization. Training sessions extended to 120 epochs for both the Split-CIFAR-100 and Split-CIFAR-100-Super datasets, while Split-CIFAR-10 training was for 40 epochs. Hyperparameters were determined through random sweeps across two tasks, using SGD with a momentum of 0.9. Further details on this process can be found in Appendix

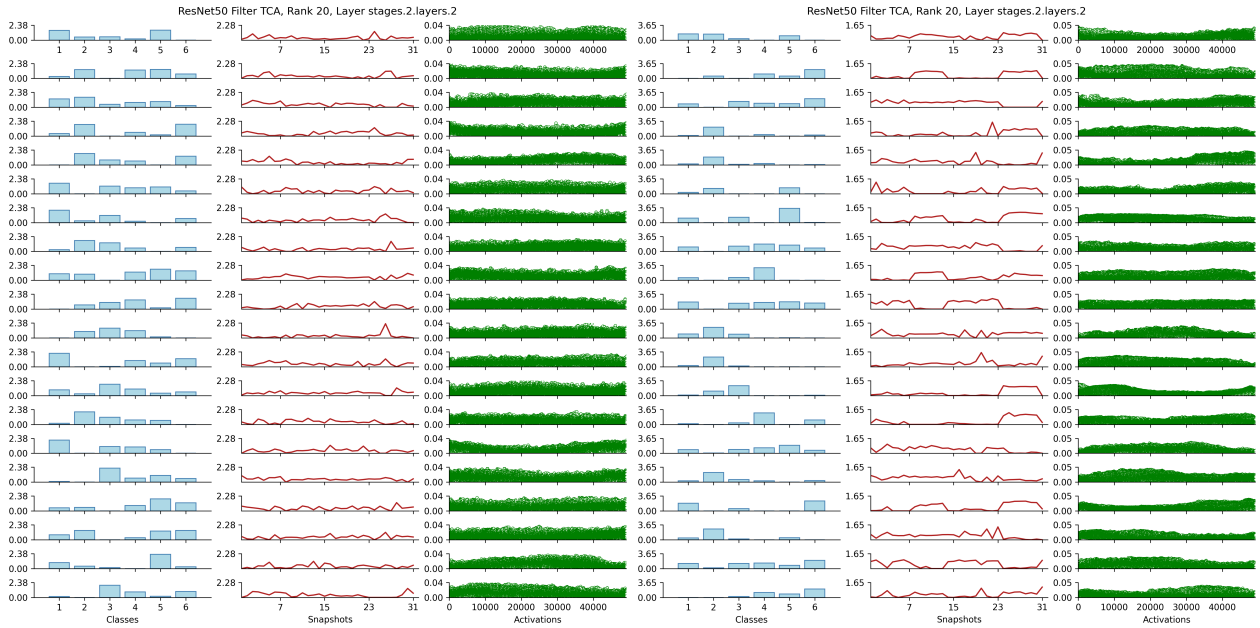

Figure 4: Example of tensor component analysis on an optimized images for filters tensor: two rank-20 TCA plots of optimized images for filters tensors from ResNet50 trained on Split-CIFAR-10. *Left*-TCA plot of filter images from the Naive strategy, with a reconstruction error of 0.24. *Right*-TCA plot of activation from the WSN strategy, with a reconstruction error of 0.19.

A.2. Sweeps were conducted on epochs, batch size, learning rate for different strategies and across different datasets. Table A.6 provides a breakdown of the final hyperparameters selected for each model and dataset combination.

6.2 TCA reconstruction errors from activations are significantly varied between architectures.

To develop an understanding of which layers are useful to study across architectures, we computed the reconstruction errors across all the layers for each architecture. Figure 2 plots the errors for each of the strategies using activations from Split-CIFAR-10. Surprisingly, we see that the reconstruction error for the ResNet50 architecture is significantly lower than the other architectures. And, there is a clear pattern where later layers in the model tend to have lower reconstruction error. The trend of reconstruction errors across the layers of CvT13, unlike ResNet50 and DeiTSmall, seem to be strategy dependent. There are noticeable variations in reconstruction errors across layers for the subnetworking strategies (WSN and RMN), whereas the reconstruction errors remain largely stable. Overall, there is a trend of reconstruction errors following strategy performance (see Figure C.11); the subnetworking strategies have lower reconstruction errors than the regularization strategeies. In addition, following our methodology, Figure B.6 plots the reconstruction errors of decompositions from activations of two layers, across ranks 10-24 for all the models.

6.3 H1: Activation tracking does not clearly highlight specialized classes of neurons.

As outlined in Section 3.5, we conduct TCA on activation tensors. We found that our approach to activation tracking produces factors that are difficult to interpret across different architectures, and the interpretability of these factors does not correlate with the performance of the strategy. Our method does not readily identify specialized clusters of neurons across architectures. Initially, we hypothesized that TCA would cleanly select an input, highlight a cluster of activations, and identify a relevant region on the snapshot (temporal) component. Figure 3 shows a sample result from applying our TCA methodology to activation tensors for a selected layer in DeiTSmall, trained on Split-CIFAR-10. Despite both strategies performing very well with the DeiTSmall model, and having similar reconstruction errors, the decompositions are not similar, scoring only 0.58 on the similarity metric proposed byWilliams et al. (2018). The authors of that study suggested that, for their tasks, similarity values of at least 0.6 indicate empirically similar factors.

Nevertheless, our decompositions on activation tensors are able to isolate individual classes and regions on the temporal axis corresponding to specific tasks. This is demonstrated across all our activation decomposition plots

(see Figures 3,B.7, B.8). For example, in Figure 3, the third component on the right plot highlights snapshots from the initial and final task and singles out a class on the inputs axis. In the Split-CIFAR-10 task, class 6 falls in the initial task, but the component highlights snapshots in the initial and the final task. Although it might be tempting to interpret such a component as indicating something about the model's learning dynamics, the decompositions tend to include conflicting and difficult to interpret selections. For instance, the first and most aligned component in both plots selects several classes and snapshots, along with a few strong activations. We believe it would be naive to attempt and directly assign meaning to such a component. While activation TCA plots for DeiTSmall tend to strongly select for certain activations, this quality is not present in the other activation decomposition plots. On the other hand, activation TCA plots for the other architectures seem to better select for individual classes (see Figure B.8) or snapshots on the temporal axis that belong to the same task (see Figure B.7). Appendix B.3 features two other activation tensor decomposition with different strategies for CvT13 and ResNet50.

6.4 H2: Filter decompositions produce smoother and more pronounced outputs, but remain difficult to interpret.

Following our tensor construction from Section 3.6, we also produce TCA plots to study the filters. Figure 4 shows two filter decomposition plots for channels $= [0, 100, 300, 500, 700, 1000]$. The left decomposition is for the Naive strategy, while the right decomposition is for the WSN strategy. From an empirical analysis, we see that the decomposition on the tensor related to WSN has more striking selections, especially in the temporal axis. Comparing the decomposition for the Naive strategy to figures B.9 and B.10, we also see that the filter selections in the CL strategies, aside from Replay, tend to be more sparse (leftmost columns; blue), while there is a more noticeable selection in the image pixels space (rightmost columns; green). Unlike the activation tensor decompositions, the filter decomposition plots produce smoother curves on the temporal axis. On the other hand, it is difficult to try and intuit what a selected filter might represent; whereas, in the activation decompositions, there are clearer meanings to be assigned when a class is selected for a certain selection on the temporal axis.

Our methodology does not immediately clarify whether these filters demonstrate similar activity in their evolution. However, the selection of multiple filters for a certain region of pixels might suggest that these filters are working together. Ultimately, while these plots are still difficult to interpret, we believe that the empirical differences in the plots between the Naive/Replay and the other CL strategies suggests that this approach is a more promising avenue than the previous to study internal representations.

Overall, we found this to be a computationally challenging approach, as it required optimizing images for filters in a chosen layer across all snapshots. Hence, we limited these plots to certain filters and for the CIFAR-10. In addition, we limited these plots to the ResNet50 architecture, as we found it challenging to adapt the "channel" optimization procedure for the other architectures in a meaningful way. Appendix B.4 features two other activation tensor decomposition with different strategies.

7 FUTURE WORK

Despite the unclear nature of our results, we believe this approach still holds merit. Tensor decompositions are a widely used tool throughout multiple domains and often help provide interpretable insight to high dimensional data. Hence, we believe that our approach would benefit from refinement. In particular, we believe adopting a mechanistic interpretability approach would be more beneficial in understanding the failure cases of using TCA to study continual learning dynamics. To elaborate, we expect that conducting decompositions on activations and filter tensors from significantly smaller toy models and controlled toy datasets would highlight the benefits and pitfalls of this approach.

8 CONCLUSION

Our deliberate choice to work with importance-based strategies stems from their explicit goal to encourage specialization, such that certain model parameters establish greater importance for certain tasks. Our analysis is centered on the determining whether we are able to track this type of specialization to understand if importance-based methods operate in the intended way. We believe specialization in CL is crucial because it encourages an efficient use of predefined resources as the model learns to allocate parameters to accommodate new tasks. By identifying meaningful behavior of existing methods through this analysis, we wish to pave the way to improve importance-based methods or discover the need to seek a different angle of attack to elicit specialization.

Ultimately we found that our methodology did not successfully highlight specialized clusters of neurons or aid in understanding the evolution of filters. Nevertheless, glimpses in our results still demonstrate that this approach has merit and could be refined and better understood for smaller datasets and architectures. By building an intuition from the ground up of what TCA shows us in this context, we believe a refined version of our approach could be a beneficial tool for study.

9 ACKNOWLEDGEMENTS

We thank Böhm et al. (2022) for sharing the CIFAR embeddings generated by tSimCNE.

This work was supported in parts by the NYUAD Center for Cyber Security (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104.

References

Adali, T., Kantar, F., Akhonda, M. A. B. S., Strother, S., Calhoun, V. D., and Acar, E. (2022). Reproducibility in matrix and tensor decompositions: Focus on model match, interpretability, and uniqueness. *IEEE Signal Processing Magazine*, 39(4):8–24.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483.

Böhm, J. N., Berens, P., and Kobak, D. (2022). Unsupervised visualization of image datasets using contrastive learning. *arXiv preprint arXiv:2210.09879*.

Böhm, J. N., Berens, P., and Kobak, D. (2023). Unsupervised visualization of image datasets using contrastive learning. In *International Conference on Learning Representations*.

Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319.

Carta, A., Pellegrini, L., Cossu, A., Hemati, H., and Lomonaco, V. (2023). Avalanche: A pytorch library for deep continual learning. *Journal of Machine Learning Research*, 24(363):1–6.

Cichocki, A., Zdunek, R., and Amari, S.-i. (2007). Hierarchical als algorithms for nonnegative matrix and 3d tensor factorization. In *International conference on independent component analysis and signal separation*, pages 169–176. Springer.

Davari, M., Asadi, N., Mudur, S., Aljundi, R., and Belilovsky, E. (2022a). Probing representation forgetting in supervised and unsupervised continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16712–16721.

Davari, M., Horoi, S., Natik, A., Lajoie, G., Wolf, G., and Belilovsky, E. (2022b). Reliability of cka as a similarity measure in deep learning. *arXiv preprint arXiv:2210.16156*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dyballa, L., Rudzite, A. M., Hoseini, M. S., Thapa, M., Stryker, M. P., Field, G. D., and Zucker, S. W. (2023). Population encoding of stimulus features along the visual hierarchy. *bioRxiv*, pages 2023–06.

Ghiasi, A., Kazemi, H., Borgnia, E., Reich, S., Shu, M., Goldblum, M., Wilson, A. G., and Goldstein, T. (2022). What do vision transformers learn? a visual exploration. *arXiv preprint arXiv:2212.06727*.

Harshman, R. A. et al. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics*, 16(1):84.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hess, T., Verwimp, E., van de Ven, G. M., and Tuytelaars, T. (2023). Knowledge accumulation in continually learned representations and the issue of feature forgetting. *arXiv preprint arXiv:2304.00933*.

Kang, H., Mina, R. J. L., Madjid, S. R. H., Yoon, J., Hasegawa-Johnson, M., Hwang, S. J., and Yoo, C. D. (2022). Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pages 10734–10750. PMLR.

Kaushik, P., Gain, A., Kortylewski, A., and Yuille, A. (2021). Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. *arXiv preprint arXiv:2102.11343*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Kruskal, J. B. (1977). Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138.

Kudithipudi, D., Aguilar-Simon, M., Babb, J., Bazhenov, M., Blackiston, D., Bongard, J., Brna, A. P., Chakravarthi Raja, S., Cheney, N., Clune, J., et al. (2022). Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4(3):196–210.

Lee, D. and Seung, H. S. (2000). Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13.

McGuire, K. L., Amsalem, O., Sugden, A. U., Ramesh, R. N., Fernando, J., Burgess, C. R., and Andermann, M. L. (2022). Visual association cortex links cues with conjunctions of reward and locomotor contexts. *Current Biology*, 32(7):1563–1576.

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

Ramasesh, V. V., Dyer, E., and Raghu, M. (2020). Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*.

Ring, M. B. (1997). Child: A first step towards continual learning. *Machine Learning*, 28:77–104.

Schlimmer, J. C. and Fisher, D. (1986). A case study of incremental concept induction. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, pages 496–501.

van de Ven, G. M., Tuytelaars, T., and Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197.

Williams, A. H., Kim, T. H., Wang, F., Vyas, S., Ryu, S. I., Shenoy, K. V., Schnitzer, M., Kolda, T. G., and Ganguli, S. (2018). Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis. *Neuron*, 98(6):1099–1115.

Wu, H., Xiao, B., Codella, N., Liu, M., Dai, X., Yuan, L., and Zhang, L. (2021). Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22–31.

Zhang, X., Dou, D., and Wu, J. (2022). Feature forgetting in continual representation learning.

A MODEL METHODS

A.1 Task Generation and Order

This section describes the methodology employed to achieve the curated task generation outlined in Section 4. In summary, we leverage the tSimCNE embeddings of the CIFAR datasets, as introduced by Böhm et al. (2023), in conjunction with the QuickHull algorithm (Barber et al., 1996) to delineate the classes for constructing the first task. Figure A.5 visualizes the tSimCNE embeddings, the N computed centroids for each class (denoted by crosses), and the convex hull defined by the subset $M \in N$ centroids (denoted by the blue boundary).

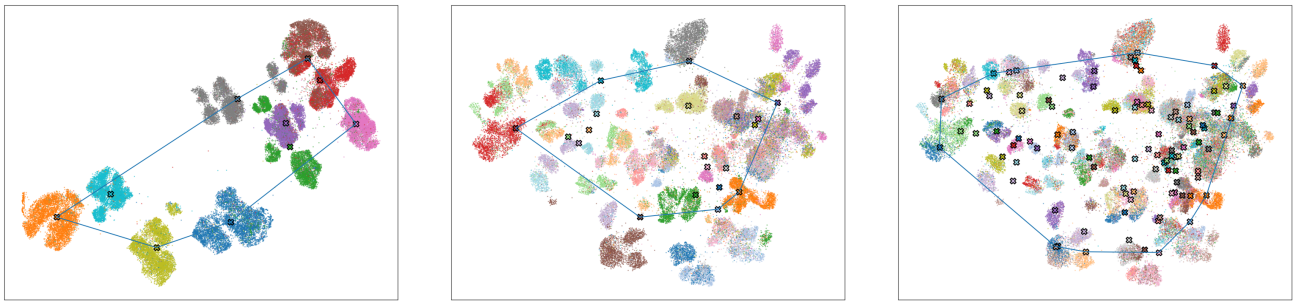

Figure A.5: Visualization of tSimCNE embeddings, the computed centroids for each class (marked by crosses), and the convex hull constructed using QuickHull algorithm for CIFAR-10 (*left*), CIFAR-100 superclasses (*middle*), and CIFAR-100 (*right*) datasets.

tSimCNE employs contrastive learning, a technique that enhances an image sample by creating two augmented versions. It then optimizes a CNN to minimize the distance between the latent representations of both augmented versions of the sample. Böhm et al. (2023) propose to optimize the following loss function:

$$\ell_{\text{t-SimCNE}}(i,j) = -\log \frac{1/(1+\|z_i - z_j\|^2)}{\sum_{k\neq i}^{2b} 1/(1+\|z_i - z_k\|^2)}$$

where z_i, z_j refer to two augmentations of the same image, in the latent space. When minimizing the loss, the numerator enforces a similarity between the same sample, while the denominator acts to repel different samples. For CIFAR-10 and CIFAR-100, we use the 2D embeddings obtained by the authors' experiments, made available at `https://github.com/berenslab/t-simcne`.

To summarize a class, we first compute the mean x and y coordinates of all samples per class, resulting in a representative centroid for each of the N classes. Following this, we employ the QuickHull algorithm to identify a subset of M centroids, where $M < N$, that constitute the convex hull encompassing all class centroids. For the QuickHull algorithm, we utilize the implementation provided at `https://github.com/zifanw/ConvexHull2D`.

The QuickHull algorithm identified $M = 6$ centroids for CIFAR-10, from which we randomly selected 4 classes to form Task 1. For CIFAR-100 (superclasses), the algorithm identified $M = 11$ centroids ($M = 7$ for CIFAR-100 superclasses), from which we randomly selected 10 classes (4 for CIFAR-100 superclasses). The remaining classes were evenly distributed among the remaining tasks. The tasks are listed in Table A.2.

Dataset	$\lvert\mathcal{T}_1\rvert$	Classes
CIFAR-10	4	automobile, airplane, frog, dog
CIFAR-100 superclasses	4	large outdoor natural scenes, large carnivores, people, household furniture
CIFAR-100	10	sea, cloud, aquarium fish, sunflower, lion, raccoon, girl, pickup truck, wardrobe, chair

Table A.2: Number of selected classes for the initial task ($\mathcal{T}_1$) and their corresponding labels across datasets.

A.2 Hyperparameter Sweeps

Our hyperparameter selections were guided by random hyperparameter sweeps for two tasks using SGD with momentum set to 0.9. For each of the three architectures introduced in Section 4, initial sweeps were conducted on epochs, batch size, and learning rate using the Split-CIFAR-10 and Split-CIFAR-100 datasets, employing the Naive strategy. Upon determining the appropriate epochs and batch size for each architecture and dataset combination, additional sweeps were conducted on learning rate and strategy-specific hyperparameters for the contrastive learning (CL) approaches on Split-CIFAR-100. The selected hyperparameters for model training are discussed in Section A.3.

Tables A.3, A.4, and A.5 detail the sweep configurations and selected hyperparameters. The results of our sweeps are documented at: `https://wandb.ai/nishantaswani/cl_decomp/sweeps`. We recommend sorting by the first column for ease of navigation.

Strategy	Dataset	Epochs	Batch Size	Learning Rate	Random Samples	Memory Size
Naive	Split-CIFAR-10	30	LogUniform(64,256)	Uniform(1×10^{-1} , 5×10^{-2})	20	-
	Split-CIFAR-100	{60,90}	{64,128,256}	Uniform(1×10^{-2} , 4×10^{-2})	50	-
Replay	Split-CIFAR-100	50	90	LogUniform(3×10^{-3} ,3×10^{-2})	50	QLogUniform(20,500,q=10)

Table A.3: Sweep configurations for the hyperparameters used by the Naive and Replay strategies across various datasets.

The sweeps corresponding to Table A.4 and A.5 were conducted 50 times each, with a fixed SGD momentum of 0.9, spanning 90 epochs, and employing a batch size of 256.

Strategy	Learning Rate	λ Parameter	α Parameter
EWC	LogUniform(1×10^{-3} ,3×10^{-2})	QLogUniform(1,1×10^{2} ,q=1)	-
MAS	LogUniform(1×10^{-3} ,3×10^{-2})	QLogUniform(1,1×10^{2} ,q=1)	Uniform(1×10^{-1} , 1)

Table A.4: Sweep configurations for the hyperparameters used by the EWC and MAS strategies on Split-CIFAR-100 dataset.

Strategy	Learning Rate	Capacity param_c	Prune Epoch	Pruning Threshold (wt_param)
WSN	LogUniform(1×10^{-3} ,3×10^{-2})	QUniform(1×10^{-1} , 1 , q=0.1)	-	-
RMN	LogUniform(1×10^{-3} ,3×10^{-2})	-	QUniform(30,80,q=5)	LogUniform(1×10^{-2} ,1×10^{-1})

Table A.5: Sweep configurations for the hyperparameters used by the WSN and RMN strategies on Split-CIFAR-100 dataset.

A.3 Model Training

We used existing implementations from the Avalanche framework (Carta et al., 2023) for EWC, MAS, and Replay. Due to our use of architectures not originally outlined in the RMN and WSN papers, we had to develop flexible implementations of RMN and WSN. We developed these implementations guided by details from the original papers and publicly available code repositories provided by the respective authors. We will contribute our Avalanche-based implementations of RMN and WSN to the Avalanche community. Table A.6 details the final hyperparameters used for each model and dataset combination.

B TENSOR COMPONENT ANALYSIS (TCA)

We fit the tensor component analysis (TCA) with the `tensortools` package (Williams et al., 2018) in Python. From empirical results, we found minimal difference between the nonnegative HALS and nonnegative BCD algorithms for fitting our TCA models. Hence, we selected the nonnegative BCD algorithm to fit our models.

B.1 Reconstruction Error per Layer

We fit rank $= 15$ TCA models across all the layers of each model, for each strategy. Each reconstruction error was averaged across 5 TCA models. We plot the means (circle) and their standard deviations (shading) in Figure 2.

Strategy	Architecture	Epochs	Batch Size	Learning Rate	Memory Size	Other Hyperparameters
Naive	CvT13 DeiTSmall ResNet50	40 for SC10; 120 for SC100 and SC100-Super	256	2.50×10^{-2} 2.00×10^{-2} 1.00×10^{-2}	-	-
Cumulative	CvT13 DeiTSmall ResNet50			2.50×10^{-2} 2.00×10^{-2} 1.00×10^{-2}	-	-
Replay	CvT13 DeiTSmall ResNet50			2.00×10^{-2} 2.00×10^{-2} 1.50×10^{-2}	200	-
EWC [Replay]	CvT13 DeiTSmall ResNet50			2.00×10^{-2} 1.50×10^{-2} 5.00×10^{-3}	[200]	$\lambda = 30$
MAS [Replay]	CvT13 DeiTSmall ResNet50			1.50×10^{-2} 1.50×10^{-2} 5.00×10^{-3}	[200]	$\lambda = 3, \alpha = 0.5$
RMN [Replay]	CvT13 DeiTSmall ResNet50			2.50×10^{-2} 2.00×10^{-2} 3.00×10^{-3}	[200]	prune epoch = 15 for SC10; 50 for SC100 and SC100-Super; wt_param = 1e-2
WSN [Replay]	CvT13 DeiTSmall ResNet50			2.00×10^{-2} 2.00×10^{-2} 1.00×10^{-2}	[200]	capacity (param_c) = 0.5

Table A.6: List of final hyperparameters used for training the different architectures on the selected strategies. SC10 refers to the Split-CIFAR-10 dataset. SC100 and SC100-Super refer to the Split-CIFAR-100 and Split-CIFAR-100-Superclasses datasets, respectively.

B.2 Rank Search

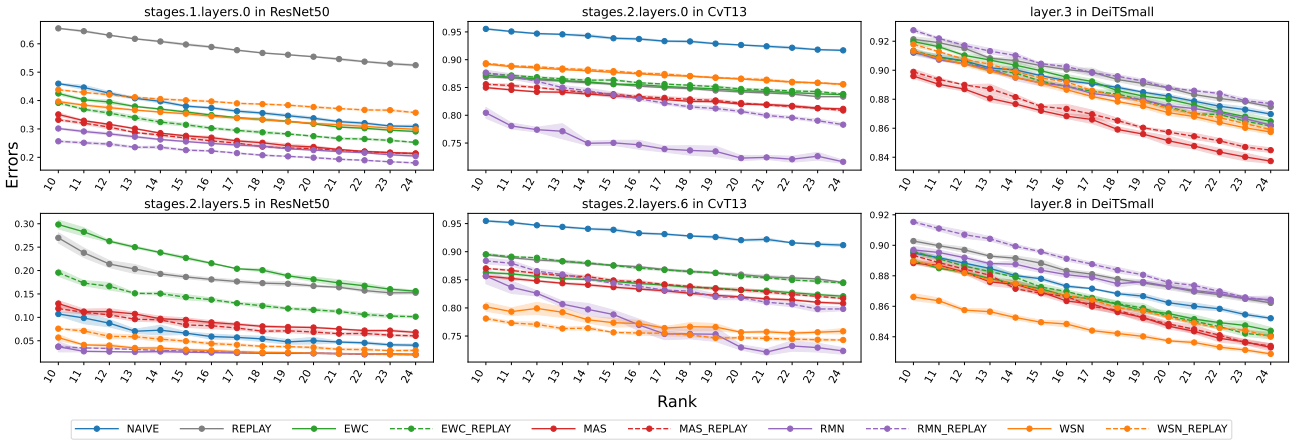

Figure B.6: Reconstruction errors across ranks with standard deviations for ResNet50 (*left*), CvT13 (*middle*), and DeiTSmall (*right*) models studied on two layers from their respective models. In the top row, we selected an early layer and in the bottom row, a relatively later layer in the architecture. **Note:** The y-axis scaling varies between plots for clearer visualization.

As outlined in Section 4, we fit TCA models across ranks 10-24 to study the effect of rank selection on the performance of our TCA models. Figure B.6 shows our resultson CIFAR-10, for two layers per model, based on tensors created from layer activations. Similar to the findings from Figure 2, the TCA models are a lot more performant in the case of ResNet50. Aside from EWC and Replay on ResNet50, and RMN on CvT13, the TCA models do not improve drastically when increasing the rank of the models.

B.3 Additional Activation Tensor Decomposition Plots

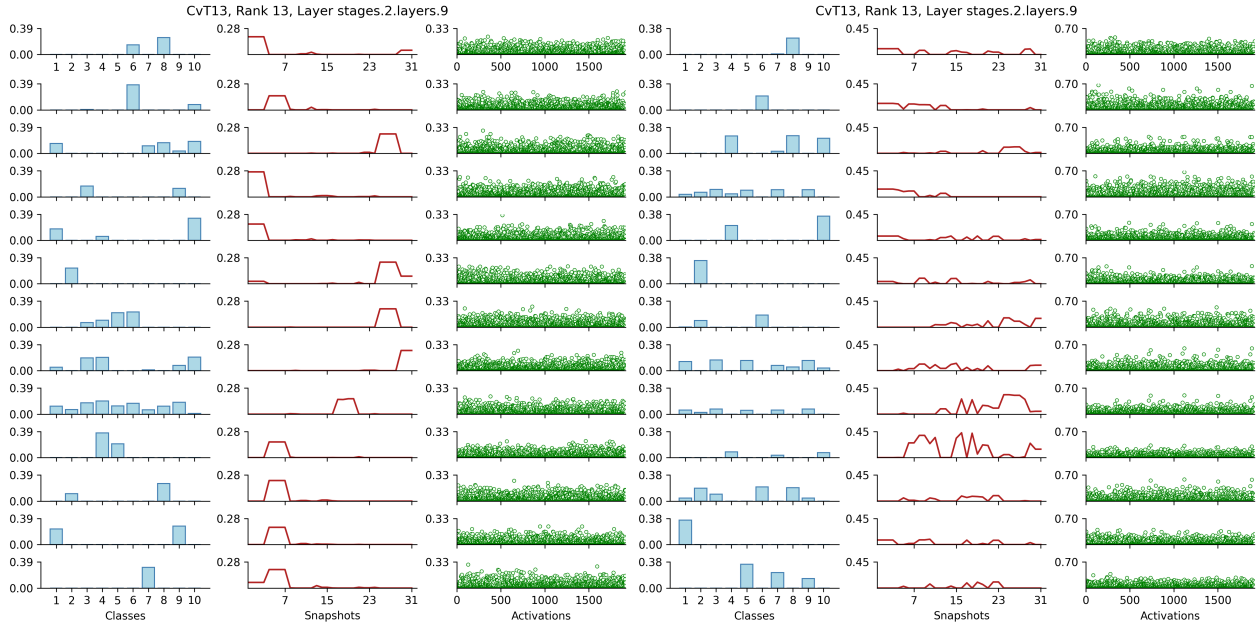

Figure B.7: Example of tensor component analysis on an activations tensor: two rank-13 TCA plots of activation tensors from CvT13 trained on Split-CIFAR-10, permuted to order factors that are best aligned. The similarity score between the two is 0.49. *Left*-TCA plot of activations from the Naive strategy, with a reconstruction error of 0.94. *Right*-TCA plot of activation from the Replay strategy, with a reconstruction error of 0.87.

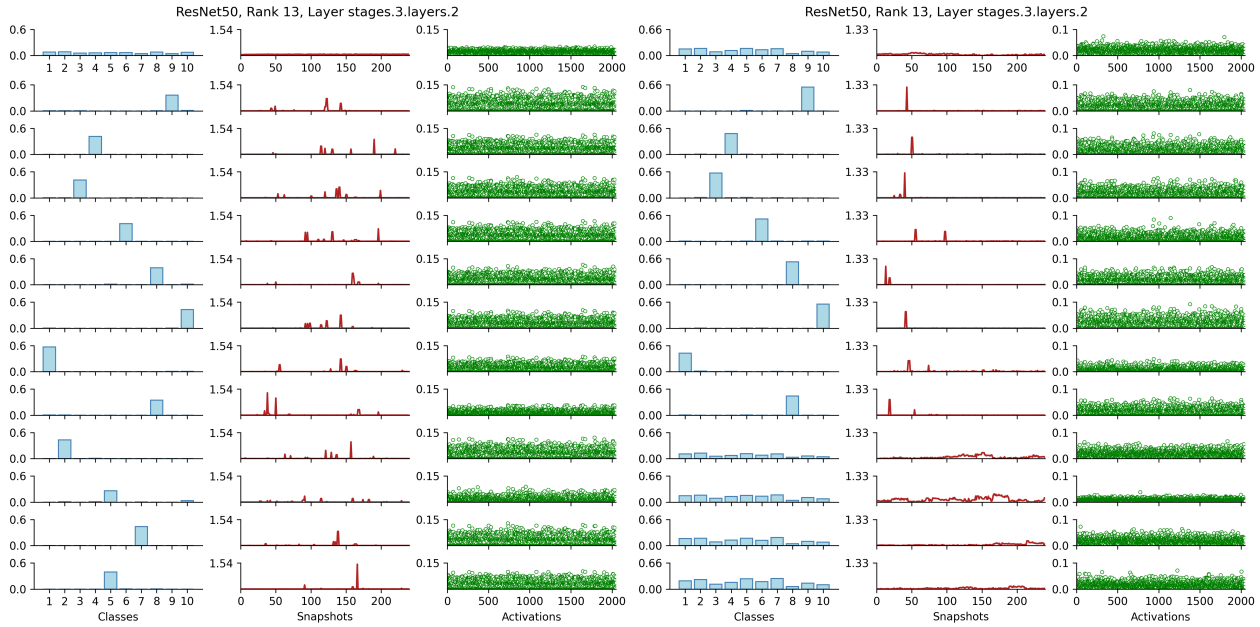

Figure B.8: Example of tensor component analysis on an activations tensor: two rank-13 TCA plots of activation tensors from ResNet50 trained on Split-CIFAR-100, permuted to order factors that are best aligned. The similarity score between the two is 0.53. *Left*-TCA plot of activations from the MAS strategy, with a reconstruction error of 0.17. Out of the 100 classes, we selected 10, where each class belongs to a distinct task. *Right*-TCA plot of activation from the WSN strategy, with a reconstruction error of 0.58.

B.4 Additional Filter Tensor Decomposition Plots

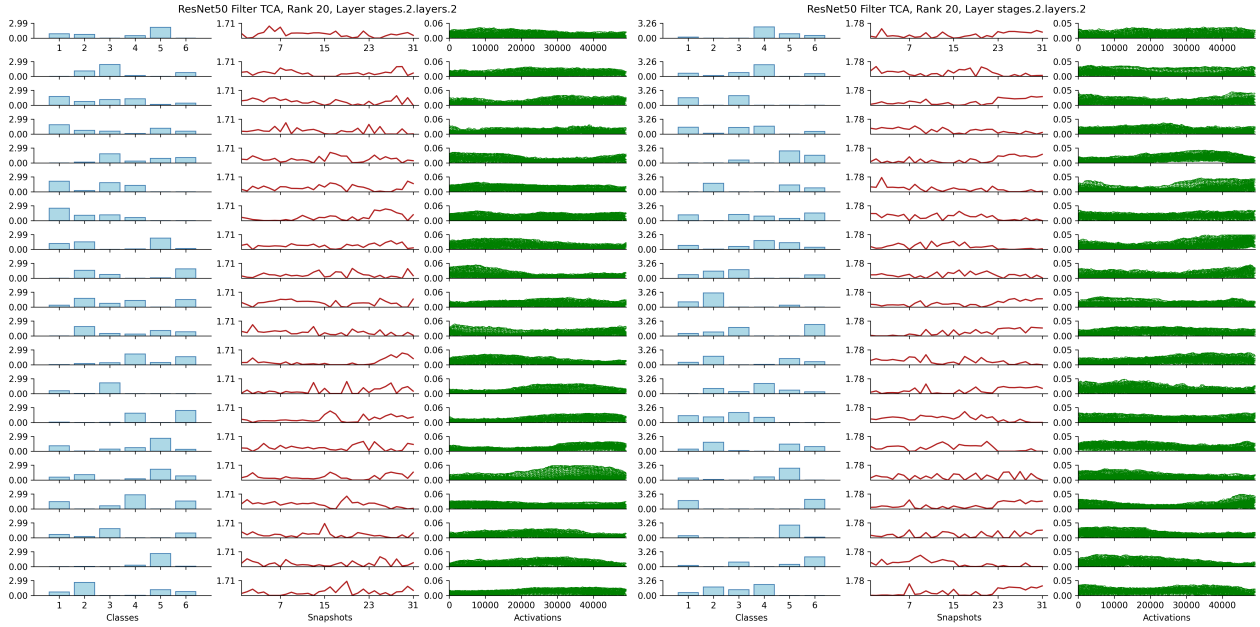

Figure B.9: Example of tensor component analysis on an optimized images for filters tensor: two rank-20 TCA plots of optimized images for filters tensors from ResNet50 trained on Split-CIFAR-10. *Left*-TCA plot of filter images from the EWC strategy, with a reconstruction error of 0.22. *Right*-TCA plot of activation from the MAS strategy, with a reconstruction error of 0.22.

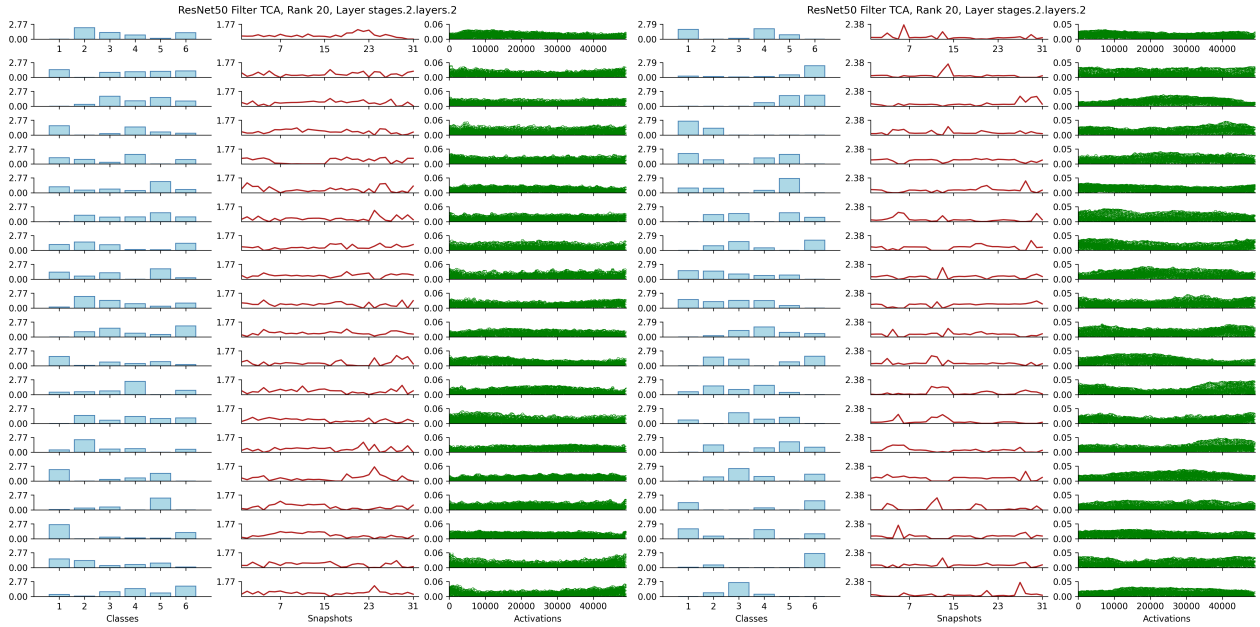

Figure B.10: Example of tensor component analysis on an optimized images for filters tensor: two rank-20 TCA plots of optimized images for filters tensors from ResNet50 trained on Split-CIFAR-10. *Left*-TCA plot of filter images from the Replay strategy, with a reconstruction error of 0.20. *Right*-TCA plot of activation from the RMN strategy, with a reconstruction error of 0.24.

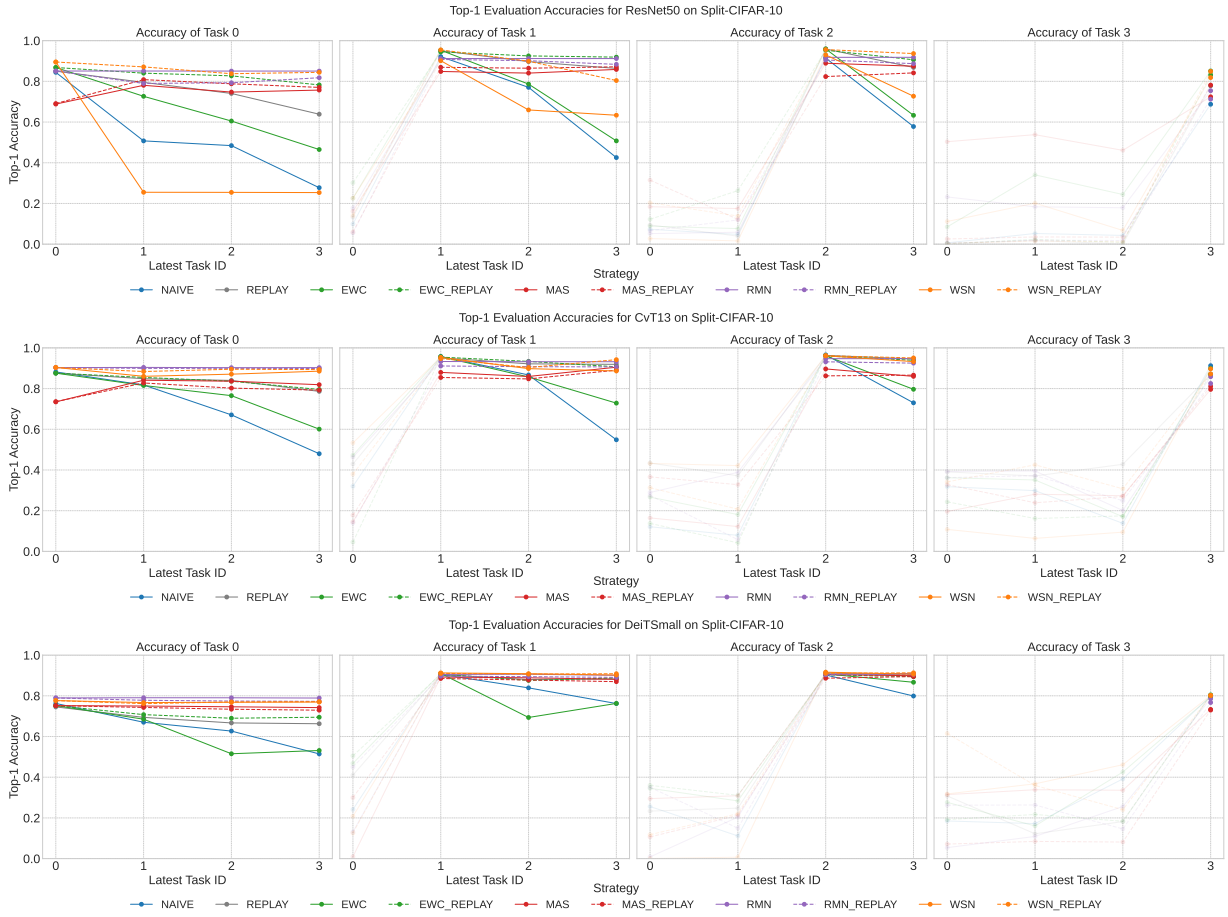

Figure C.11: Results on the Split-CIFAR-10 dataset. *Upper Row* - ResNet50, *Middle Row* - CvT13, *Bottom Row* - DeiTSmall.

C TRAINING RESULTS

Figures C.11, C.12, and C.13 display top-1 evaluation accuracies across different architectures and datasets using the CL strategies from Table 1. We highlight the evaluation accuracy on a task only after the model has encountered the task.

For a better visualization of our training results, we encourage readers to view the results on our dashboard at: `https://wandb.ai/nishantaswani/cl_decomp/reportlist`

D CODE

All of our project code is documented and publicly available at `https://github.com/niniack/CLDecomp`.

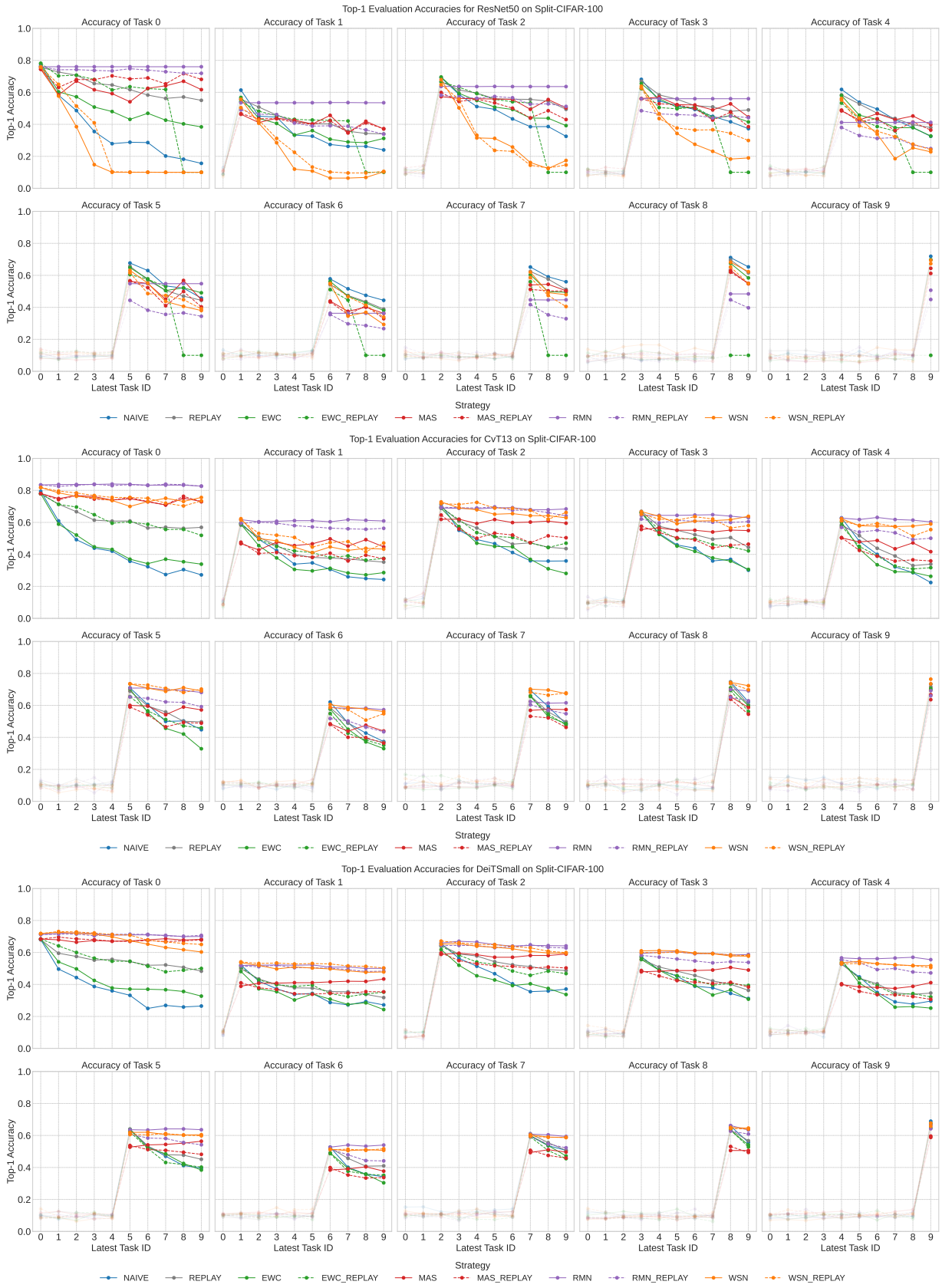

Figure C.12: Results on the Split-CIFAR-100 dataset. *Upper Row* - ResNet50, *Middle Row* - CvT13, *Bottom Row* - DeiTSmall.

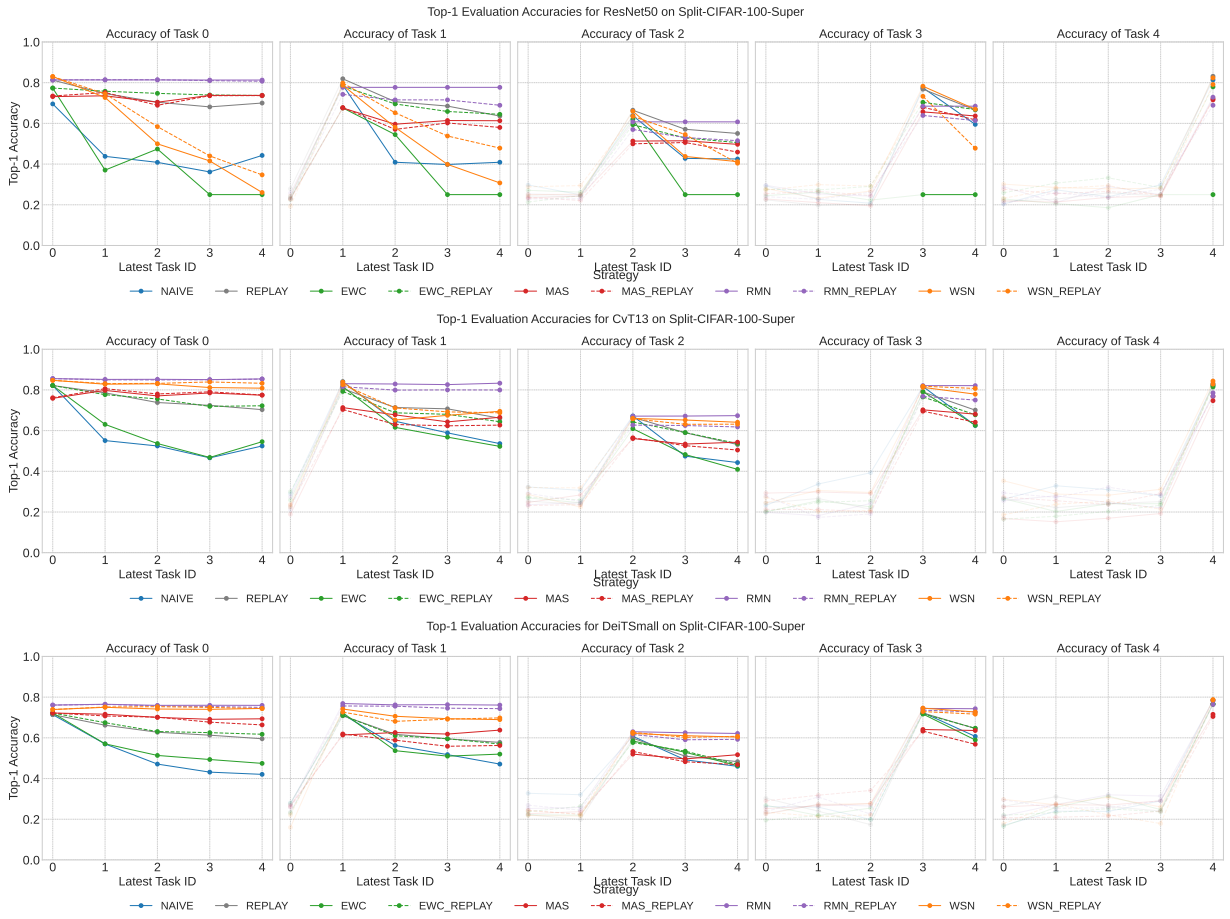

Figure C.13: Results on the Split-CIFAR-100-Super dataset. *Upper Row* - ResNet50, *Middle Row* - CvT13, *Bottom Row* - DeiTSmall.