FMNIL: FEATURE-LEVEL MODULAR NETWORK FOR CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning (CL) enables models to sequentially learn from a stream of tasks while retaining previously acquired knowledge, mitigating catastrophic forgetting (CF). Modular approaches in CL enhance flexibility by decomposing tasks into reusable modules. However, existing approaches suffer from a trade-off between accuracy and scalability, often resorting to parameter expansion to preserve accuracy at the cost of efficiency and long-term applicability. In addition, these methods that evaluate task similarity based on task accuracy are computationally expensive and limited in capturing deep representation similarity. To overcome these limitations, we propose **F**eature-level **M**odular **N**etwork for Cont**I**nual **L**earning (FMNIL), which focuses on feature-level task similarity. FMNIL dynamically constructs task-specific subnetworks, reusing modules when tasks exhibit high feature similarity and expanding the network only when necessary. This design maintains accuracy while breaking the accuracy–scalability trade-off, achieving significantly less parameter growth without sacrificing performance. Experiments on four benchmarks (CIFAR100-SC/RS/B0, ImageNet1000) show FMNIL achieves up to 5.9% higher accuracy (2.8% on average) while reducing parameter growth by 35% compared to state-of-the-art methods. FMNIL thus provides an accuracy-preserving and parameter-efficient solution that breaks the accuracy–scalability trade-off in long-term continual learning.

1 INTRODUCTION

Continual learning (CL) enables models to learn sequentially from a stream of tasks while retaining knowledge from previously learned tasks. However, traditional deep learning architectures are prone to catastrophic forgetting (CF) (Goodfellow et al., 2013; McCloskey & Cohen, 1989; McClelland et al., 1995; Zhuang et al., 2023), where new tasks overwrite previously acquired knowledge, leading to performance degradation on earlier tasks. To address this challenge, a wide range of approaches has been proposed, including regularization-based, replay-based, optimization-based, representation-based, and architecture-based methods (Wang et al., 2024).

Among these, architecture-based methods are especially appealing due to their ability to adapt model structures dynamically. While other strategies rely on a shared parameter set across tasks—leading to inter-task interference—architecture-based methods allocate task-specific parameters, which directly reduces such interference (Kang et al., 2022; Mallya et al., 2018; Serra et al., 2018; Jin & Kim, 2022; Zhuang et al., 2022). This motivates our focus on architecture-based approaches in this work. There are two types of methods in architecture-based methods, fixed-capacity and capacity-increasing. Fixed-capacity methods maintain a fixed parameter budget across tasks, prioritizing resource efficiency and low structural complexity. However, they risk capacity saturation, limiting performance on complex new tasks. Conversely, capacity-increasing methods dynamically expand the model to ensure high flexibility. For instance, PackNet (Mallya & Lazebnik, 2018) uses iterative pruning and parameter reallocation, while DyTox (Douillard et al., 2022) dynamically expands transformer token representations for efficient task adaptation.

Within capacity-increasing methods, *modular networks* have garnered significant attention for their ability to decompose tasks into reusable modules, facilitating interpretability, flexibility, and reusability. For example, PathNet (Fernando et al., 2017) uses genetic algorithms to select reusable components, while PICLE (Valkov et al., 2023) incorporates probabilistic models for module combination.

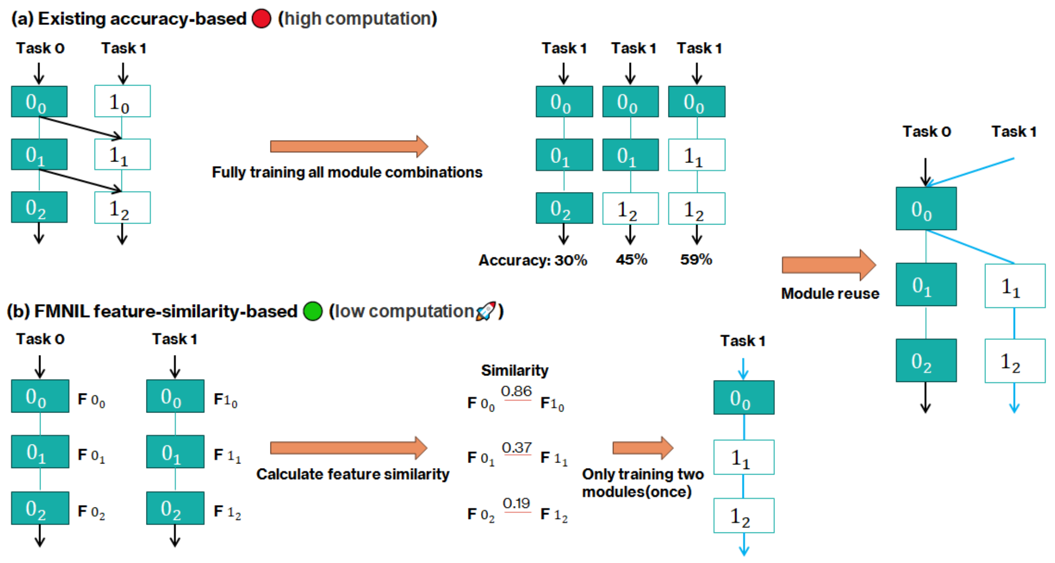
(a) Existing accuracy-based (high computation)
Task 0
Task 1
Fully training all module combinations
Accuracy: 30%
45%
59%
Module reuse
(b) FMNIL feature-similarity-based (low computation)
Calculate feature similarity
Similarity
0.86
0.37
0.19
Only training two modules(once)

Figure 1: An illustration highlighting the differences between existing modular network methods and FMNIL: (a) Existing methods rely on fully training all module combinations and then selecting the best-performing one. This accuracy-driven evaluation typically requires full model training and multiple evaluations to identify reusable components. (b) FMNIL identifies reusable components through feature-similarity-based evaluation. Note that the **colored modules** represent frozen modules from Task 0, while only the **hollow modules** from the new task (Task 1) are updated during training. Assuming that the best architecture for Task 1 at the previous step is $(0_0, 1_1, 1_2)$, module 0_0 is reused for Task 1, while modules 1_1 and 1_2 are added to the current library of modules.

However, these methods struggle with the large search space of module combinations, high computational costs, and difficulties in balancing module reuse with task-specific adaptation.

Although modular networks show great potential in mitigating CF, most prior methods suffer from a trade-off between accuracy and scalability, typically relying on parameter expansion to maintain accuracy, which hinders long-term scalability. In addition, a critical limitation in current modular approaches is their reliance on task accuracy as the primary metric for evaluating task similarity. This accuracy-driven evaluation typically requires full model training and multiple evaluations to identify reusable components, significantly increasing computational cost. Moreover, task-level accuracy provides only a coarse signal and may fail to capture underlying feature-level similarities between tasks, which are crucial for effective module reuse in architecture-based CL frameworks.

In contrast to accuracy-based comparison strategies, we propose using feature-level similarity as a more efficient and informative proxy for evaluating task relatedness. By computing cosine similarity between task-level feature representations, reusable modules can be dynamically identified without exhaustive retraining, thereby reducing computational overhead and enhancing scalability. This perspective is illustrated in Figure 1, which contrasts existing modular approaches with our proposed design. Building on this motivation, we introduce **F**eature-level **M**odular **N**etwork for Cont**I**nual **L**earning (FMNIL), a modular continual learning architecture that integrates feature similarity directly into the module selection process, reusing modules for similar tasks and expanding only when necessary. This similarity-driven mechanism supports task-specific subnetwork construction, mitigates catastrophic forgetting, and avoids uncontrolled parameter growth. By leveraging feature-level similarity for module reuse, FMNIL provides an accuracy-preserving and parameter-efficient solution for long-term continual learning. The contributions of this paper are as follows:

- We introduce FMNIL, a modular architecture that leverages feature similarity to guide module reuse, providing a novel mechanism within the architecture-based CL framework.

- We demonstrate that feature-level similarity offers an efficient and effective alternative to accuracy-based task similarity evaluation, reducing computational overhead and enabling scalable modular reuse.
- We provide a theoretical analysis of FMNIL's advantages in terms of parameter efficiency, storage optimization, and computational complexity.
- We validate FMNIL on benchmark datasets (CIFAR100-SC, CIFAR100-RS, CIFAR100-B0, and ImageNet1000), showcasing its superior performance in terms of average accuracy and Top-5 accuracy compared to state-of-the-art methods.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces the problem formulation. Section 4 describes the proposed FMNIL framework. Section 5 reports experimental details and results, followed by the conclusion in 6.

2 Related work

This section reviews related work in two key areas: continual learning (CL) methods, particularly architecture-based strategies, and modular network designs that support flexible adaptation across tasks.

2.1 Continual learning

CL methods can be categorized into five main families of approaches: regularization-based, replay-based, optimization-based, representation-based, and architecture-based (Wang et al., 2024). Each category tackles CL challenges uniquely, leveraging its underlying mechanisms to mitigate catastrophic forgetting (CF), enhance adaptation, and improve resource efficiency.

Architecture-based methods can be categorized into *fixed-capacity* and *capacity-increasing* approaches based on the evolution of their model parameters as the number of tasks increases in the early work. The first fixed-capacity method is the *mask-based approach*, which uses binary masks to selectively activate or deactivate parts of the neural network. WSN (Kang et al., 2022) and HAT (Serra et al., 2018) employ binary masks and attention mechanisms to protect important weights and dynamically adjust task-specific subnetworks. The second can be collectively referred to as *parameter reallocation*, which contrasts with the mask-based methods by focusing on dynamically identifying and reallocating parameters based on their importance and activity. PackNet (Mallya & Lazebnik, 2018) employs iterative pruning and re-training to free up redundant parameters, optimizing for new tasks without affecting the performance on older tasks through a weight-magnitude-based pruning method.

However, fixed-capacity methods in CL face several additional limitations. In addition to the issues of parameter saturation and the need for sparse constraints, these include the potential for degraded model performance as the network's capacity is exhausted, leading to compromised learning on new tasks. Another drawback is the challenge of efficiently reusing and reallocating network resources, which can result in suboptimal task performance and increased computational overhead. To address these challenges, network architectures can dynamically expand their capacity when insufficient for mastering new tasks. The first *capacity-increasing* method is *parameter segregation*. BNS (Qin et al., 2021) dynamically constructs the network structure for each new task by adding nodes and layers as needed, which are determined by a controller using reinforcement learning to balance network capacity and task-specific needs. The second is *model decomposition*, which separates a model into task-sharing and task-specific components, with the latter often being expandable. DyTox (Douillard et al., 2022) further exemplifies innovation by applying transformer architectures to CL. It dynamically expands token representations while maintaining computational efficiency. The third type is *modular network*, which utilize parallel sub-networks or sub-modules to learn incremental tasks in a differentiated manner compared to model decomposition without predefined task-shared or task-specific components. PICLE (Valkov et al., 2023) employs a probabilistic model to estimate the fitness of module combinations for achieving perceptual, few-sample, and latent transfer learning. PathNet (Fernando et al., 2017) utilizes agents embedded within the network to identify reusable network components. It selects paths for replication and mutation through genetic algorithms. Our approach introduces a new perspective on modular networks.

2.2 Modular Learning

Modular learning decomposes models into functional components to enhance flexibility, interpretability, and reusability—properties particularly useful in CL settings. A broader comparison between modular and non-modular paradigms, including their respective strengths and limitations, is provided in Appendix A.2. In our comparison with state-of-the-art methods, the term *non-modular approaches* specifically refers to regularization- and replay-based CL methods.

2.3 Prototype Learning

Prototype-based methods classify samples by comparing them to class or task prototypes, typically defined as the mean feature vectors of exemplars. A more detailed discussion of prototype methods is provided in Appendix A.3. Our FMNIL can also be regarded as a lightweight prototype CL method.

3 Preliminaries and Problem Formulation

Continual learning (CL) aims to enable deep learning models to learn sequentially from a stream of tasks while retaining previously acquired knowledge. In a typical CL setup, a model is exposed to a sequence of tasks, where each task t is defined by a dataset $\mathcal{D}_t = \{(x_{i,t}, y_{i,t})\}_{i=1}^{n_t}$, consisting of n_t input-output pairs. In continual learning, the model learns task t using only the current dataset $\mathcal{D}_t$, without access to previous datasets $\mathcal{D}1, \ldots, \mathcal{D}t-1$. This restricted access setting introduces challenges such as *catastrophic forgetting*.

The objective of CL is to optimize a shared set of model parameters θ to minimize the cumulative loss across all tasks observed so far. Formally, given T tasks, the optimization objective is defined as:

$$\min_{\theta} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim\mathcal{D}_t} \left[\mathcal{L}(f(x;\theta), y)\right],$$

where $f(x;\theta)$ denotes the model's prediction, and $\mathcal{L}(\cdot)$ is the task-specific loss function, such as cross-entropy loss.

To effectively address the challenges of CL, the model parameters θ are often decomposed into *shared components*, which capture knowledge transferable across tasks, and *task-specific components*, which adapt to the unique characteristics of each task.

In summary, this formulation provides the foundation for designing continual learning algorithms that balance *stability* (retaining knowledge) and *plasticity* (adapting to new tasks), enabling the model to generalize effectively across a sequence of tasks.

4 Methodology

This section first motivates our FMNIL method and then details its key components.

4.1 Motivation and Framework Overview

Traditional modular continual learning methods often evaluate task similarity through task accuracy. However, this approach is computationally expensive and limited in capturing deep representational semantics. To address this issue, FMNIL leverages feature-level similarity to decide whether to reuse existing modules or expand with new ones, enabling efficient adaptation across tasks.

Figure 2 illustrates the FMNIL pipeline. It consists of two key components: (a) **similarity-driven module selection**, and (b) **subnetwork construction**. Specifically, when a new task arrives, FMNIL computes cosine similarities between the task's average feature representations and those of previously learned tasks. If the similarity exceeds a learned threshold, corresponding modules from a shared library are reused; otherwise, new modules are instantiated. This strategy enables efficient transfer of knowledge while minimizing unnecessary expansion. To ensure robust performance across task streams, FMNIL employs a *genetic algorithm* to adaptively determine the similarity threshold, balancing stability and plasticity. This dynamic thresholding helps avoid both brittle reuse and excessive growth, achieving scalable continual learning with reduced parameter overhead.

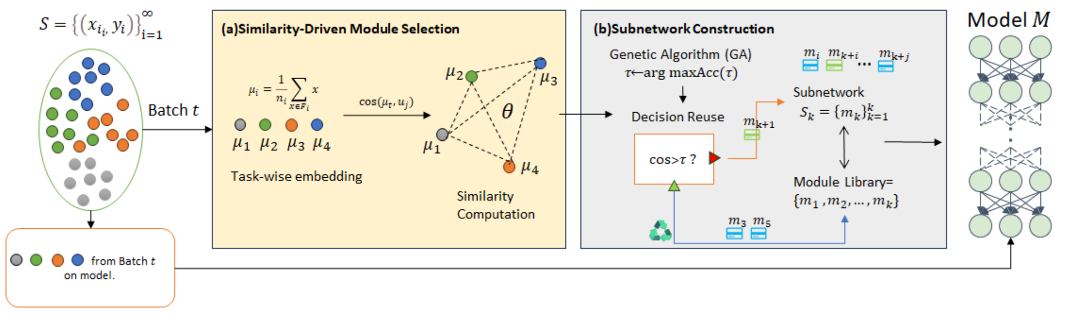
$S = \{(x_{i_i}, y_i)\}_{i=1}^{\infty}$
Batch t
from Batch t on model.
(a)Similarity-Driven Module Selection
$\mu_i = \frac{1}{n_i}\sum_{x \in F_i} x$
μ_1 μ_2 μ_3 μ_4
Task-wise embedding
$\cos(\mu_t, u_j)$
θ
Similarity Computation
(b)Subnetwork Construction
Genetic Algorithm (GA)
$\tau \leftarrow \arg\max \mathrm{Acc}(\tau)$
Decision Reuse
$\cos > \tau$?
m_i m_{k+i} … m_{k+j}
m_{k+1}
Subnetwork $S_k = \{m_k\}_{k=1}^{k}$
Module Library= $\{m_1, m_2, \ldots, m_k\}$
m_3 m_5
Model M

Figure 2: Overview of our FMNIL method, (a) similarity-driven module selection, where task-wise embeddings are computed and compared via cosine similarity, and (b) subnetwork construction by reusing or expanding modules from a shared library based on a learned threshold.

This design is inspired by recent findings from representational similarity analysis (Raghu et al., 2017; Morcos et al., 2018) and contrastive learning (Azizi et al., 2023; Chen et al., 2020; Wang & Isola, 2020), which demonstrate that internal feature distributions encode transferable task semantics. FMNIL leverages these findings in a modular architecture that supports dynamic expansion and principled module selection, as described in the following sections.

4.2 FMNIL Architecture

FMNIL is implemented as a backbone-agnostic modular framework. In this work, we instantiate it with ResNet18 and adopt a modular design by partitioning the backbone into distinct functional modules, each responsible for a specific stage of feature extraction (He et al., 2016). The modularized ResNet18 architecture, illustrated in Figure 5 (Appendix A.10), comprises four residual modules; each contains two residual blocks, and each block consists of two 3×3 convolutional layers.

The model incrementally constructs a sequence of task-specific subnetworks, which are dynamically assembled and integrated into the FMNIL architecture. For the initial task (Task 0), FMNIL uses the entire ResNet18 to construct SubNetwork 0 and stores its feature representations in a buffer for later reuse.

For each subsequent task T_i $(i > 0)$, FMNIL extracts features using previously constructed subnetworks and compares them with stored representations at corresponding modular levels. Modules exhibiting high cosine similarity are reused, while dissimilar ones are replaced with newly initialized modules, yielding a task-specific subnetwork S_i.

This modular and similarity-guided strategy enables FMNIL to transfer knowledge across tasks efficiently while minimizing interference, thus supporting continual adaptation without catastrophic forgetting (CF). A visualization of subnetwork cooperation during inference is provided in Figure 6 (Appendix A.10). The full construction and training procedure is detailed in Algorithm 1, with additional implementation information in Appendix A.4.

4.3 Dynamic Network Expansion

Dynamic expansion is a key component of FMNIL, enabling structural adaptation based on task-specific requirements. When a new task T_i exhibits low feature similarity with previous tasks, FMNIL instantiates new modules to accommodate its distinct representation needs. These newly added modules inherit architectural design from existing ones, ensuring consistency across the framework.

To maintain balance between reuse and growth, FMNIL adjusts its parameter allocation during the expansion process. This selective augmentation prevents unnecessary duplication while preserving model flexibility. In contrast to naive capacity scaling, which leads to uncontrolled parameter growth, FMNIL expands only when task dissimilarity justifies structural extension.

4.4 Choice of Mean Vectors

Classical Gaussian-based continual learning methods characterize task representations with both the mean and covariance of feature distributions, which leads to $O(d^2)$ storage and computational complexity for feature dimension d. This quadratic cost quickly becomes prohibitive on modern high-dimensional backbones.

FMNIL instead adopts a lightweight prototype-style representation. For each task, we store only *mean feature vectors*, which reduces the overhead to $O(d)$ and thus scales gracefully to large datasets and long task sequences. Specifically, one mean vector is extracted from each residual module of the backbone, resulting in four vectors for ResNet-18. This design aligns directly with the modular decomposition of the architecture: early modules capture low-level edges and textures, middle modules encode local patterns and parts, and later modules represent high-level semantic abstractions. By maintaining one mean per module, FMNIL achieves broad semantic coverage while avoiding redundancy, ensuring that transferable features are preserved without excessive memory cost. This one-to-one alignment between residual modules and mean vectors is not arbitrary: each module forms a semantically coherent subnetwork, making its mean feature a natural summary representation. This design ensures that transferable features at different levels are preserved without redundancy.

Empirical support. We compared different granularities of mean statistics. A single mean discards transferable features and reduces accuracy, while eight means bring only marginal gains (<0.5%) but nearly double the overhead. Four means thus provide the best trade-off between accuracy and efficiency (see Appendix A.5 for full results).

Generality across architectures. Although we instantiate FMNIL on ResNet-18 for clarity, the design naturally extends to other backbones. In DenseNet, one can treat each dense block as a module and extract a mean vector accordingly; in vision transformers, transformer blocks or even heads can serve as modules. The general principle is to align the number of stored mean vectors with the natural modular decomposition of the backbone, thereby ensuring semantic coverage across depth while maintaining scalability.

4.5 Measuring Task Relatedness

A core challenge in modular continual learning lies in determining when and how to reuse modules across tasks. Prior approaches typically rely on task-level validation accuracy as a proxy for similarity, which requires full training and lacks sensitivity to representation-level alignment. In contrast, FMNIL leverages cosine similarity between mean feature representations to achieve efficient and semantically meaningful module selection.

To operationalize this idea, FMNIL determines whether to reuse existing modules or instantiate new ones based on the feature-level similarity between tasks. For each task T_i, we compute the cosine similarity between its mean feature vector and those of prior tasks. Specifically, given a feature matrix $F_i \in \mathbb{R}^{n_i \times d}$ for task T_i, the mean feature vector μ_i is defined as:

$$\mu_i = \frac{1}{n_i} \sum_{x \in F_i} x.$$

The similarity between two tasks T_i and T_j is then calculated as:

$$\text{sim}(i, j) = \frac{\langle \mu_i, \mu_j \rangle}{\|\mu_i\| \|\mu_j\|}.$$

If $\text{sim}(i, j) > \tau$, the modules from T_j are reused in constructing T_i's subnetwork; otherwise, new modules are created.

The similarity threshold τ critically affects reuse decisions: a low τ risks over-reuse and degraded task performance, while a high τ may trigger unnecessary expansion. To adaptively tune this parameter, FMNIL employs a Genetic Algorithm (GA) that evolves a population of threshold candidates via crossover and mutation. The threshold maximizing average task accuracy is selected:

$$\tau^* = \arg \max_{\tau \in [0,1]} \text{Accuracy}_{\text{avg}}(\tau).$$

This approach enables FMNIL to automatically balance generalization and adaptability across diverse task distributions, ensuring efficient module selection without exhaustive evaluation. The module selection procedure is summarized in Algorithm 2, and pseudocode is provided in Appendix A.4. Detailed derivations of parameter efficiency and computational complexity are deferred to Appendix A.6.

5 Experiments

In this section, we extensively evaluate FMNIL on visual classification tasks. All experiments were conducted on a server equipped with NVIDIA A100-SXM4-80GB GPUs. All models and algorithms are implemented using the PyTorch (Paszke et al., 2019) library. We will release our code upon acceptance of our paper for reproduction of the results.

5.1 Datasets

We evaluate FMNIL on four widely used continual learning benchmarks: CIFAR100 (with SC, RS, and B0 splits), ImageNet100, and ImageNet1000. These benchmarks cover small-scale, medium-scale, and large-scale scenarios, allowing us to assess both accuracy and scalability across diverse settings.

CIFAR100. CIFAR100 (Krizhevsky et al., 2009) contains 60,000 images across 100 classes, and serves as a widely used benchmark for continual learning. Following prior works (Zhuang et al., 2023; 2024b;a), we adopt three different task split protocols: (i) **CIFAR100-SC** (Yoon et al., 2019), which groups classes into 20 semantically coherent superclasses and samples tasks while preserving semantic structure. We evaluate this split under 10- and 20-phase settings (but not 50-phase, which would fragment the superclass structure). (ii) **CIFAR100-RS**, where classes are randomly shuffled and divided into tasks without semantic constraints. We evaluate under 10-, 20-, and 50-phase settings (Wang et al., 2022). (iii) **CIFAR100-B0**, which assigns classes sequentially by index, following (Rebuffi et al., 2017). This protocol is also tested under 10-, 20-, and 50-phase settings. Together, these three splits provide diverse small-scale CL benchmarks with different levels of task coherence.

ImageNet100. Following prior work in continual learning (Douillard et al., 2020; 2022), we include ImageNet100, a 100-class subset of ImageNet-1K constructed by sampling 100 classes and splitting them into 10 sequential tasks (10 classes each). This benchmark offers a medium-scale setting, striking a balance between CIFAR100 (small-scale) and ImageNet1000 (large-scale), while retaining higher visual and semantic diversity than CIFAR100.

ImageNet1000. Finally, we evaluate on the large-scale ImageNet-1K dataset (Deng et al., 2009), consisting of over 1.2 million training images across 1,000 classes. Following common CL protocols (Douillard et al., 2022), we divide the dataset into 10 sequential tasks of 100 classes each. This benchmark is the most challenging and realistic, assessing both the scalability and transferability of continual learning methods.

5.2 Comparison with Baselines

We evaluate our approach against nine established continual learning (CL) baselines. These comparisons include an upper bound, a representative regularization-based method, a replay-based approach, and six architecture-based strategies. Specifically, we compare against:

Bound: Joint training, an upper-target in continual learning, involves training a single network on all tasks simultaneously. **SS-IL** (Ahn et al., 2021), a *non-modular* regularization-based method that mitigates prediction bias between old and new classes through separated softmax and task-wise knowledge distillation. **MIR** (Aljundi et al., 2019), a *non-modular* replay-based method that selects the most informative samples for replay based on interference potential. **DyTox** (Douillard et al., 2022), a fixed-capacity, transformer-based architecture with shared encoders and dynamically allocated task-specific tokens. **MARK** (Hurtado et al., 2021), which leverages meta-learning to construct and maintain a shared knowledge base with trainable masks. **HAT** (Serra et al., 2018), which employs task-specific hard attention masks. **MNTDP** (Veniat et al., 2020), a capacity-increasing approach using modular neural networks with task-driven priors for dynamic module selection and sharing. **DER** (Yan et al., 2021), which dynamically expands feature representations by freezing previously learned features and adding new ones for each task. **DER (w/o P)** (Yan et al., 2021), a variant of DER that omits the pruning step during feature expansion to retain more information from novel concepts.

5.3 Training Settings

We mainly focus on the task-incremental setting used in prior work (Wang et al., 2022; Serra et al., 2018) and follow their implementation for most experiments unless specified otherwise. We use ResNet18 (He et al., 2016) as the backbone architecture in our framework. Each task is trained for 100 epochs with a batch size of 128, using the Adam optimizer with an initial learning rate of 1×10^{-3} and weight decay of 1×10^{-5}. The learning rate is decayed via a cosine annealing schedule over each task. To optimize the cosine similarity threshold for module reuse, we employ a Genetic Algorithm (GA), which dynamically adjusts the threshold to prevent excessive reuse or unnecessary expansion. Full training configurations and optimization details are provided in Appendix A.7.

5.4 Metrics

In this study, we report the two key evaluation metrics. **Average Accuracy (AA)** measures the mean accuracy across all tasks after the model has completed the entire task sequence, offering a holistic view of overall performance. **Top-5 Accuracy** evaluates whether the true class label appears within the top five predictions, providing a more tolerant assessment for large-scale classification tasks. This metric is particularly relevant for datasets like ImageNet, where inter-class similarity is high and precise top-1 accuracy may underrepresent model capability. Formal definitions and computational details of these metrics are provided in Appendix A.8.

5.5 Results

The main experimental results are reported in this section along with a discussion. Additional results are presented in the Appendix.

5.5.1 Comparative Analysis of Methods

We evaluate FMNIL against state-of-the-art continual learning approaches, including regularization-based, replay-based, and architecture-based (fixed-capacity and capacity-increasing strategies). The performance comparison is conducted on three benchmark datasets: CIFAR-100-SC, CIFAR-100-RS, and CIFAR-100-B0, across multiple phases (10-phase, 20-phase, and 50-phase). The results, including average accuracy and parameter count, are reported in Table 1. The rationale behind CIFAR-100-SC being evaluated without 50 phases is explained in Section 5.1.

Table 1: The table compares the parameter count and average accuracy of various CL methods across different datasets (CIFAR-100-SC, CIFAR-100-RS, and CIFAR100-B0) under different task phases (10-phase, 20-phase, and 50-phase). #P means the average number of parameters used during inference over phases, which is counted by millions.

	CIFAR-100-B0						**CIFAR-100-SC**				**CIFAR-100-RS**					
Method	**# P**	**10-phase**	**# P**	**20-phase**	**# P**	**50-phase**	**# P**	**10-phase**	**# P**	**20-phase**	**# P**	**10-phase**	**# P**	**20-phase**	**# P**	**50-phase**
Bound	11.2	80.41	11.2	81.49	11.2	81.74	11.2	80.41	11.2	81.49	11.2	80.41	11.2	81.49	11.2	81.74
SS-IL (Ahn et al., 2021)	11.7	63.15 ±1.01	11.7	61.08 ±0.83	11.7	-	11.7	64.20 ±4.21	11.7	62.48 ±2.78	11.7	66.27 ±1.12	11.7	67.02 ±1.83	11.7	61.33 ±0.86
MIR (Aljundi et al., 2019)	11.2	62.21 ±0.92	11.2	60.95 ±0.89	11.2	-	11.2	62.83 ±4.92	11.2	61.83 ±3.76	11.2	56.09 ±1.06	11.2	55.52 ±1.46	11.2	49.38 ±0.25
Dytox (Douillard et al., 2022)	10.7	65.42 ±0.34	10.7	62.12 ±0.59	10.7	58.15 ±0.42	10.7	69.52 ±3.34	10.7	66.39 ±1.70	10.7	70.62 ±0.34	10.7	65.53 ±1.77	10.7	63.58 ±0.74
HAT (Serra et al., 2018)	6.8	69.31 ±1.01	6.8	65.20 ±0.95	6.8	59.87 ±0.85	6.8	62.95 ±0.82	-	-	6.8	68.57 ±0.24	6.8	66.96 ±1.32	6.8	50.29 ±0.89
MARK (Hurtado et al., 2021)	4.7	69.87 ±0.78	4.7	64.92 ±0.91	4.7	59.38 ±1.03	4.7	64.84 ±0.76	-	-	4.7	69.31 ±0.89	-	-	-	-
MNTDP (Veniat et al., 2020)	102	63.05 ±0.95	-	-	-	-	102	63.90 ±4.87	-	-	102	69.23 ±1.62	-	-	-	-
DER (w/o P) (Yan et al., 2021)	61.6	67.60 ±0.31	117.6 (+90.9%)	66.35 ±0.40	285.6 (+142.8%)	65.29 ±1.29	61.6	70.41 ±3.76	117.6 (+90.9%)	68.45 ±1.03	61.6	70.58 ±1.57	117.6 (+90.9%)	70.04 ±0.19	285.6 (+142.8%)	68.98 ±0.21
DER (Yan et al., 2021)	5	67.12 ±1.43	7.2 (+44%)	66.00 ±1.13	10.2 (+41.67%)	64.37 ±2.06	5	69.02 ±3.52	7.2 (+44%)	68.39 ±3.12	5	70.01 ±1.05	7.2 (+44%)	69.41 ±0.67	10.2 (+41.67%)	67.27 ±0.28
FMNIL (Ours)	38.5	**74.90** ±0.58	52.1 (+35.2%)	**70.12** ±0.85	72.4 (+38.96%)	**67.05** ±0.95	38.5	**71.30** ±0.55	52.1 (+35.2%)	**72.50** ±2.81	38.5	**74.12** ±0.61	52.1 (+35.2%)	**73.08** ±0.37	72.4 (+38.96%)	**69.35** ±0.21

Note: The data for *Bound* is obtained from (Douillard et al., 2022).

Regularization-Based and Replay-Based Methods

Regularization-based methods, such as SS-IL (Ahn et al., 2021), and replay-based methods, such as MIR (Aljundi et al., 2019), demonstrate relatively stable performance but suffer from limited adaptability due to their fixed-capacity nature. SS-IL achieves moderate accuracy, reaching 66.27% on CIFAR-100-RS (10-phase), but its performance declines to 61.33% in the 50-phase setting on CIFAR-100-B0. MIR, which relies on memory replay, exhibits weaker performance on CIFAR-100-B0, dropping to 49.38% at 50-phase.

Architecture-Based: Fixed-Capacity Strategies

Fixed-capacity models, including DyTox (Douillard et al., 2022), HAT (Serra et al., 2018), and MARK (Hurtado et al., 2021), perform well in early phases but struggle as task complexity increases due to their constrained parameter space.

DyTox achieves 69.52% on CIFAR-100-SC (10-phase) but declines to 58.15% at 50-phase on CIFAR-100-B0. HAT maintains strong performance initially, reaching 69.31% on CIFAR-100-B0 (10-phase), but its accuracy drops to 59.87% at 50-phase. MARK, which has the smallest parameter count (4.7M), achieves competitive early-phase results but experiences accuracy degradation over longer task sequences.

Architecture-Based: Capacity-Increasing Strategies

Capacity-increasing approaches dynamically adjust network parameters to improve adaptability. Among them, MNTDP (Veniat et al., 2020), DER (Yan et al., 2021), and FMNIL demonstrate distinct parameter growth patterns:

- *MNTDP*'s substantial 102M parameters achieve only 63.05% accuracy on CIFAR-100-B0 (10-phase), revealing diminishing returns from brute-force scaling.
- *DER (w/o P)* exhibits aggressive expansion from 61.6M to 285.6M parameters across phases (90.9%-142.8% growth rates), yet delivers suboptimal accuracy (68.98% on CIFAR-100-SC at 50-phase).
- Standard *DER* shows moderated growth from 5M to 10.2M parameters (44.0%-41.67% increases) with 67.27% accuracy on CIFAR-100-SC (50-phase).
- *FMNIL* demonstrates controlled scaling from 38.5M to 72.4M parameters (35.3%-38.96% growth rates), significantly slower than both DER variants.

To further illustrate the parameter scaling trends across different CL methods, we visualize the parameter count progression over increasing task phases in Figure 3.

As shown in Figure 3, DER (w/o P) undergoes excessive parameter growth, reaching 4.64× its initial size by the 50-phase setting, significantly increasing computational and memory overhead. In contrast, standard DER scales at a moderate rate (2.04×), yet still exhibits higher parameter inflation compared to FMNIL. Our FMNIL method achieves a balanced trade-off, maintaining a controlled scaling ratio of 1.88× while achieving competitive accuracy (see Table 1). This demonstrates that FMNIL effectively mitigates the drawbacks of uncontrolled capacity expansion while preserving learning performance.

Figure 3 shows that while FMNIL follows a linear scaling pattern, traditional modular architectures like DER (w/o P) experience quadratic-like growth. This difference is crucial for real-world deployment, where excessive parameter expansion can lead to prohibitive memory and computational costs.

Compared to DER (w/o P), which scales aggressively (from 61.6M to 285.6M parameters), FMNIL constrains growth to a significantly lower range (from 38.5M to 72.4M), making it a more scalable and resource-efficient solution for long-horizon continual learning.

FMNIL: A Balanced Trade-Off

Unlike conventional **modular networks** that incur quadratic parameter growth, our FMNIL introduces feature reuse mechanisms achieving linear scaling. This design yields:

- Accuracy superiority: 69.35% on CIFAR-100-RS (50-phase) vs. 67.27% (*DER*) and 68.98% (*DER w/o P*).

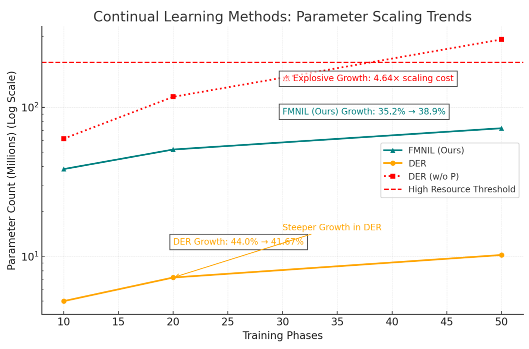

Figure 3: Comparison of parameter scaling trends across continual learning methods.

- Controlled scaling: 1.88× total parameter growth (10→50 phases) vs. 2.04× (*DER*) and 4.64× (*DER w/o P*).
- Empirical efficiency: 67.05% on CIFAR-100-B0 (50-phase) outperforms *DER* (64.37%) despite requiring 7.1× fewer parameters than *DER w/o P*.

The results validate FMNIL's dual advantage: maintaining accuracy leadership while imposing minimal parameter overhead, establishing a new Pareto frontier for continual learning systems.

5.5.2 Top-5 Accuracy on ImageNet1000

We evaluate Top-5 accuracy on ImageNet1000 to assess the model's robustness in large-scale continual learning. As shown in Figure 4 in Appendix A.9, FMNIL demonstrates strong performance throughout the task stream, especially in later stages. Compared to baselines such as DyTox, DER, and WA, FMNIL maintains superior Top-5 accuracy while using fewer parameters. Detailed stage-wise comparisons and analysis are provided in Appendix A.9.

6 Conclusion

We proposed FMNIL, a modular continual learning framework that evaluates task similarity based on feature representations instead of task accuracy. FMNIL dynamically reuses modules when tasks exhibit high similarity and expands only when necessary. This design improves scalability and reduces parameter growth while preserving knowledge. Our experimental results on four benchmarks demonstrate FMNIL's superior trade-off between accuracy and efficiency. Compared to state-of-the-art methods, FMNIL consistently improves accuracy by 2.8% to 5.9% across benchmarks with more than 3.5× fewer parameters. In future work, we plan to explore selective forgetting and hierarchical module structures to further improve scalability in large-scale task streams.

Limitations

FMNIL relies on a similarity threshold τ to guide module reuse, which may require tuning when applied to domains with different feature distributions. Furthermore, our approach assumes that the base network architecture can be decomposed into reusable modules—a structural constraint that may not generalize to all architectures or tasks.

Broader Impact

This work focuses on methodological improvements in modular continual learning. While FMNIL is designed for general-purpose machine learning tasks, it does not involve sensitive data, decision-making in critical domains, or applications with direct social implications. Nonetheless, potential impacts in real-world deployment, such as improved model efficiency for edge devices, merit further exploration in future work.

Ethics Statement

This study did not involve human subjects or personally identifiable data. All datasets are publicly available. The work complies with the ICLR Code of Ethics.

Reproducibility Statement

We have taken several measures to ensure the reproducibility of our results. All datasets and benchmark splits used (CIFAR100-SC/RS/B0, ImageNet100, and ImageNet1000) are publicly available and described in Section 5.1. Detailed training configurations, hyperparameters, and evaluation metrics are reported in Section 5. Algorithmic details, theoretical derivations, and ablation studies are provided in the Appendix A. We will release the full implementation, including code and scripts to reproduce all experiments, upon acceptance of this paper, to support transparency and community reuse.

References

Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 844–853, 2021.

Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in Neural Information Processing Systems*, 32, 2019.

Mohammed Amer and Tomás Maul. A review of modularization techniques in artificial neural networks. *Artificial Intelligence Review*, 52:527–561, 2019.

Siamak Azizi et al. Robust contrastive learning using negative samples with semantics. *ICML*, 2023.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

X Chen, X Liang, L Li, and H Lin. Conpl: Contrastive prototype learning for continual relation extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. Modular networks for compositional instruction following. *arXiv preprint arXiv:2010.12764*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. Ieee, 2009.

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, pp. 86–102. Springer, 2020.

Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Julio Hurtado, Alain Raymond, and Alvaro Soto. Optimizing reusable knowledge for continual learning via metalearning. *Advances in Neural Information Processing Systems*, 34:14150–14162, 2021.

Hyundong Jin and Eunwoo Kim. Helpful or harmful: Inter-task association in continual learning. In *European Conference on Computer Vision*, pp. 519–535. Springer, 2022.

Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D Yoo. Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pp. 10734–10750. PMLR, 2022.

Apoorv Khandelwal, Ellie Pavlick, and Chen Sun. Analyzing modular approaches for visual question decomposition. *arXiv preprint arXiv:2311.06411*, 2023.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Xiang Li, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. Adaptive prototype learning and allocation for few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8334–8343, 2021.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82, 2018.

James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Qing Miao, Ying He, Peng Zhang, Fei Wu, Han Yu, Changjun Chen, Yong Xu, and Zhenghua Song. Few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12183–12192, 2021.

Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Qi Qin, Wenpeng Hu, Han Peng, Dongyan Zhao, and Bing Liu. Bns: Building network structures dynamically for continual learning. *Advances in Neural Information Processing Systems*, 34: 20608–20620, 2021.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep understanding and model comparison. In *NeurIPS*, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Johnson Teen, Vishal Patel, et al. Training-free initialization for incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Lazar Valkov, Akash Srivastava, Swarat Chaudhuri, and Charles Sutton. A probabilistic framework for modular continual learning. *arXiv preprint arXiv:2306.06545*, 2023.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

Liyuan Wang, Xingxing Zhang, Qian Li, Jun Zhu, and Yi Zhong. Coscl: Cooperation of small continual learners is stronger than a big one. In *European Conference on Computer Vision*, pp. 254–271. Springer, 2022.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*, 2020.

Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3014–3023, 2021.

Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations*, 2019.

Zhixiang Zhang, Zhongqi Wang, Di Lin, Mingli Song, and Hanxiao Xie. Few-shot incremental learning with continually evolved classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12455–12464, 2021.

Hanbin Zhao, Hao Zeng, Xin Qin, Yongjian Fu, Hui Wang, Bourahla Omar, and Xi Li. What and where: Learn to plug adapters via nas for multidomain learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6532–6544, 2021.

Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. Forward compatible few-shot class-incremental learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Huiping Zhuang, Zhenyu Weng, Hongxin Wei, Renchunzi Xie, Kar-Ann Toh, and Zhiping Lin. ACIL: Analytic class-incremental learning with absolute memorization and privacy protection. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 11602–11614. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/4b74a42fc81fc7ee252f6bcb6e26c8be-Paper-Conference.pdf`.

Huiping Zhuang, Zhenyu Weng, Run He, Zhiping Lin, and Ziqian Zeng. GKEAL: Gaussian kernel embedded analytic learning for few-shot class incremental task. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7746–7755, June 2023. doi: 10.1109/CVPR52729.2023.00748.

Huiping Zhuang, Yizhu Chen, Di Fang, Run He, Kai Tong, Hongxin Wei, Ziqian Zeng, and Cen Chen. GACL: Exemplar-free generalized analytic continual learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., December 2024a.

Huiping Zhuang, Run He, Kai Tong, Ziqian Zeng, Cen Chen, and Zhiping Lin. DS-AL: A dual-stream analytic learning for exemplar-free class-incremental learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(15):17237–17244, March 2024b. doi: 10.1609/aaai.v38i15.29670. URL `https://ojs.aaai.org/index.php/AAAI/Article/view/29670`.

A APPENDIX

This appendix provides supplementary materials that complement the main text. It includes detailed algorithm descriptions, theoretical derivations, dataset partition protocols, and extended experimental results. These materials aim to ensure reproducibility and provide deeper insight into the proposed FMNIL framework.

A.1 LARGE LANGUAGE MODEL USAGE

This appendix provides supplementary materials that complement the main text. It includes detailed algorithm descriptions, theoretical derivations, dataset partition protocols, and extended experimental results. These materials aim to ensure reproducibility and provide deeper insight into the proposed FMNIL framework.

A.2 MODULAR VS. NON-MODULAR APPROACHES

Modular and non-modular approaches represent two distinct paradigms in deep learning system design. Modular learning decomposes models into functionally independent components, enhancing reusability, interpretability, and adaptability (Amer & Maul, 2019; Corona et al., 2020). This paradigm is particularly effective for tasks involving compositional reasoning or cross-domain generalization, such as visual question answering (VQA) and multidomain learning (Zhao et al., 2021). However, modular architectures also introduce design overhead, requiring efficient module selection strategies and clear interface integration (Khandelwal et al., 2023).

In contrast, non-modular (monolithic) models optimize the entire architecture end-to-end. These approaches are well-suited for data-rich scenarios with narrowly defined tasks, where simplicity and raw performance are prioritized, as seen in image classification and language generation tasks (Khandelwal et al., 2023). While easier to train, monolithic models typically lack flexibility, transparency, and robustness to task shifts or out-of-distribution data.

FMNIL builds on the modular learning paradigm by leveraging its strengths in task decomposition and adaptability, while addressing limitations in scalability and module selection through feature-level similarity and dynamic reuse mechanisms.

A.3 PROTOTYPE-BASED METHODS

Prototype-based continual learning methods represent each class by prototypes, defined as the mean feature vectors of selected exemplars, and classify test samples by nearest-prototype matching in the latent space. A representative method is iCaRL (Rebuffi et al., 2017), which integrates exemplar replay and nearest-class-mean classification, a mechanism proven critical in class-incremental learning (CIL). Other related works include DER++ (Buzzega et al., 2020), which combines prototype regularization with rehearsal to maintain class separation in the embedding space.

Recent advancements extend prototype-based learning to diverse continual learning scenarios. For example, CEC (Zhang et al., 2021) evolves class prototypes dynamically through graph-based message passing, enabling adaptive classifier refinement in few-shot class-incremental learning (FSCIL). TEEN (Teen et al., 2023) introduces a training-free prototype calibration strategy that aligns feature and classifier representations without updating model weights, offering a lightweight solution for resource-constrained settings. FACT (Zhou et al., 2022) improves forward compatibility by pre-allocating virtual prototypes in the embedding space, facilitating the accommodation of novel classes without retraining.

In the FSCIL domain, FSLL (Miao et al., 2021) employs shared sparse activation structures guided by prototypes, thus maintaining compact representations without large backbones or extensive replay. ConPL (Chen et al., 2023) emphasizes semantic stability across tasks by enforcing prototype consistency, supporting knowledge retention in continual relation extraction. Finally, APLA (Li et al., 2021) highlights the versatility of prototype-based learning in dense prediction by dynamically allocating and optimizing prototypes for pixel-wise segmentation in few-shot class-incremental settings.

These methods collectively underscore the adaptability and effectiveness of prototype-based classification across modalities (image, relation, segmentation) and regimes (few-shot, training-free, lifelong), solidifying prototypes as a foundational mechanism in continual learning.

A.4 Algorithmic Details

We provide the detailed procedures of FMNIL's modular construction and module selection process, as shown in Algorithm 1 and Algorithm 2.

Algorithm 1 FMNIL Model Construction

```
Require: Task sequence $\{Task_0, \ldots, Task_N\}$
Ensure: Trained FMNIL with subnetworks $\{SubNet_0, \ldots, SubNet_N\}$
 1: Initialize buffer with Task 0 features
 2: Train SubNet$_0$ on Task$_0$
 3: for $i = 1$ to $N$ do
 4:     Select modules via SelectModules($Task_i$, Buffer)
 5:     Construct SubNet$_i$
 6:     Train SubNet$_i$ on Task$_i$
 7:     Update buffer with Task $i$ features
 8: end for
 9: return All trained subnetworks
```

Algorithm 2 SelectModules via Feature Similarity

```
Require: Current task $T_i$, feature buffer from previous tasks
Ensure: Selected modules for $T_i$
 1: for each prior task $T_j$, where $j < i$ do
 2:     Extract feature matrix $\mathcal{F}_i$ using SubNet$_j$
 3:     Compute mean vector $\mu_i = \frac{1}{n_i} \sum_{x \in \mathcal{F}_i} x$
 4:     Retrieve $\mu_j$ from buffer
 5:     Compute cosine similarity between $\mu_i$ and $\mu_j$
 6:     if similarity $> \tau$ then
 7:         Reuse module(s) from SubNet$_j$
 8:     else
 9:         Create new module(s) for $T_i$
10:     end if
11: end for
```

A.5 Ablation on the Number of Mean Vectors

We report detailed ablation results for the number of mean vectors stored per task in Table 2. As discussed in Section 4.4, storing too few vectors loses transferable features, while too many increases overhead without clear benefit.

Using only a single mean vector (from the final module) led to a drop of approximately 2–3% accuracy on CIFAR100 benchmarks, since transferable low- and mid-level features were discarded. Using eight mean vectors (two per module) yielded only marginal improvement (<0.5%) compared to four, while roughly doubling the storage and computation cost.

These results show that four mean vectors achieve the best trade-off between accuracy and efficiency: a single mean discards useful low/mid-level features, while eight means offer negligible accuracy gains (<0.5%) but nearly double the overhead compared to four.

Table 2: Ablation on the number of mean vectors per task. Memory is reported per task (KB).

Setting	CIFAR100 Acc. (%)	ImageNet-100 Acc. (%)	Memory / Task (KB)
1 mean	73.2	62.5	0.5
4 means (ours)	76.0	64.8	2.0
8 means	76.3	65.1	4.0

A.6 Theoretical Analysis

We provide full derivations for parameter growth analysis and computational complexity as discussed in Section 4.

Parameter growth. In conventional modular CL methods (e.g., DER (Yan et al., 2021), PathNet (Fernando et al., 2017)), the network expands with each task, leading to total parameter growth of $\mathcal{O}(T \cdot m)$, where T is the number of tasks and m is the average module size. FMNIL reuses modules across tasks by measuring feature similarity, reducing unnecessary expansion. Let $\rho \in (0, 1)$ denote the average reuse ratio. Then the expected parameter growth becomes:

$$\text{Params}(T) = \rho \cdot T \cdot m, \quad \text{with } \rho \ll 1$$

This sub-linear growth is supported by empirical results in Section 5 (see Figure 3).

Module selection complexity. Accuracy-based methods must retrain and evaluate each candidate module for new tasks, leading to per-task selection cost of:

$$\mathcal{C}_{\text{acc}} = T \cdot E, \quad \text{where } E \text{ is epochs per evaluation}$$

In contrast, FMNIL computes cosine similarity between task-level feature means (see Section 4.5), yielding selection complexity of:

$$\mathcal{C}_{\text{FMNIL}} = T$$

This reduces the selection cost by a factor of E and eliminates the need for full model evaluation.

Threshold optimization. FMNIL further introduces a Genetic Algorithm (GA) to automatically determine the optimal similarity threshold τ^*. The total cost of GA optimization is $\mathcal{O}(k \cdot g \cdot T)$, where k is the population size and g the number of generations. As k and g are small constants, the optimization overhead is negligible and amortized over the task stream.

Together, these properties demonstrate FMNIL's scalability and efficiency in long-horizon continual learning settings.

A.7 Training Configuration

Optimizer and Learning Schedule. We use the Adam optimizer with an initial learning rate of 1×10^{-3} and weight decay of 1×10^{-5}. The learning rate is decayed using a cosine annealing schedule over each task. Each task is trained for 100 epochs with a batch size of 128.

Architecture and Initialization. Our base model is a modularized ResNet18, which is dynamically assembled per task based on feature similarity. All modules are randomly initialized and trained from scratch for each task, unless reused.

Task Sequence. We evaluate FMNIL on three CIFAR100 variants (SC, RS, B0) and ImageNet1000 under 10, 20, and 50 incremental phases, consistent with prior benchmarks.

Hyperparameter Tuning. Our method involves a critical hyperparameter: the cosine similarity threshold for module reuse. We employ a Genetic Algorithm (GA) to search for the optimal threshold. GA evolves a population of candidate values across generations by evaluating model performance, enabling dynamic adjustment based on dataset characteristics. This prevents both excessive module reuse (which can impair adaptability) and overly conservative reuse (which causes unnecessary expansion). All reported results correspond to the best-performing configuration determined by GA.

A.8 Evaluation Metric Details

Average Accuracy (AA). We define $a_{k,j} \in [0, 1]$ as the accuracy on the test set of task j after training the network on task 1 through k. The Average Accuracy (AA) is computed as:

$$\text{AA}_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j} \tag{1}$$

Top-5 Accuracy. Top-5 Accuracy assesses the proportion of test samples where the true class label is among the top five predicted classes. It is defined as:

$$\text{Top-5 Accuracy} = \frac{N_c}{N_t} \tag{2}$$

where N_c denotes the number of correctly predicted samples (i.e., the ground-truth label is within the top five predictions), and N_t represents the total number of test samples.

This metric offers a relaxed yet effective performance assessment for large-scale classification, especially when classes are fine-grained or overlapping. It accounts for uncertainty in prediction ranking and avoids penalizing the model harshly for near-correct outputs. In datasets such as ImageNet, where label ambiguity is common, Top-5 Accuracy serves as a more comprehensive measure of classification reliability. We report the mean and standard deviation over three independent runs with different random seeds for all quantitative results presented in the main paper.

A.9 Additional Results on ImageNet1000

Figure 4 shows the top-5 accuracy across different phases on ImageNet1000, further supporting the robustness of FMNIL.

Top-5 Accuracy Trends. Figure 4 presents a detailed Top-5 accuracy comparison across continual learning tasks on ImageNet1000.

Early Stages (100–300 classes): FMNIL performs slightly below DyTox (by at most 1%), remaining competitive with other baselines such as WA and DER w/o P.

Later Stages (500–1000 classes): FMNIL surpasses DyTox and maintains a Top-5 accuracy of 85.50%, while DyTox drops to 84.49%. Compared to DER w/o P (82.85%), WA (81.10%), Simple-DER (80.76%), and BiC (73.20%), FMNIL consistently demonstrates superior performance.

This trend highlights FMNIL's ability to retain learned knowledge more effectively as task complexity increases, while systematically mitigating catastrophic forgetting (CF). Unlike fixed-capacity or continuously growing architectures, FMNIL leverages modular feature reuse to achieve higher efficiency in balancing accuracy and memory constraints.

A.10 Additional Visualizations

Subnetwork Architecture. Figure 5 illustrates the modular ResNet18 architecture used as the base subnetwork in FMNIL. Each task-specific subnetwork is composed of selected or newly instantiated modules drawn from this structure, enabling flexible reuse across tasks.

Subnetwork Cooperation During Testing. Figure 6 shows how subnetworks in FMNIL cooperate during inference by reusing modules across tasks.

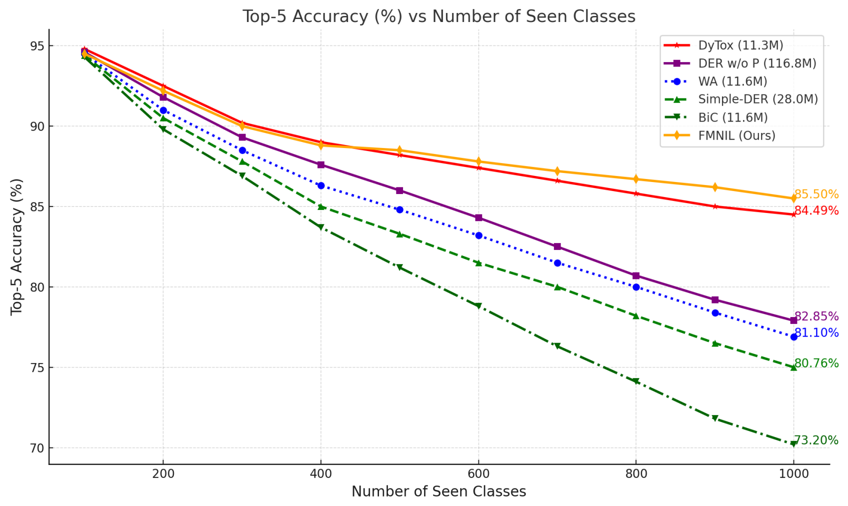

Figure 4: Top-5 Accuracy of different methods on continual learning tasks with ImageNet1000. Baseline results are taken from (Douillard et al., 2022). Each task introduces 100 new classes. Previously learned classes are not fully accessible but while previous tasks must still be retained. FMNIL (in orange) achieves competitive performance compared to state-of-the-art methods like DyTox (in red), while reducing catastrophic forgetting over time. It consistently outperforms other baselines such as DER w/o P (purple), WA (blue), Simple-DER (dashed green), and BiC (dark green), while maintaining a more compact model.

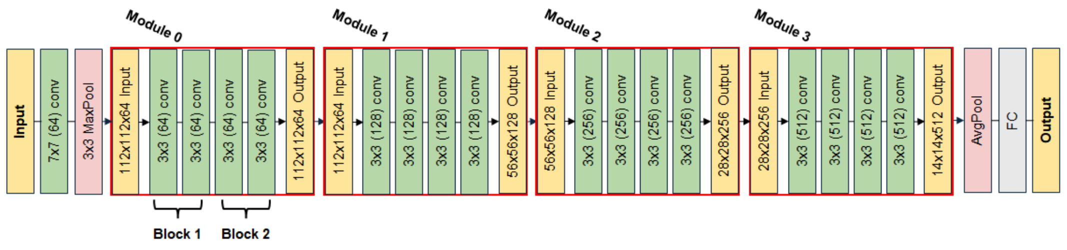

Figure 5: SubNetwork in FMNIL.

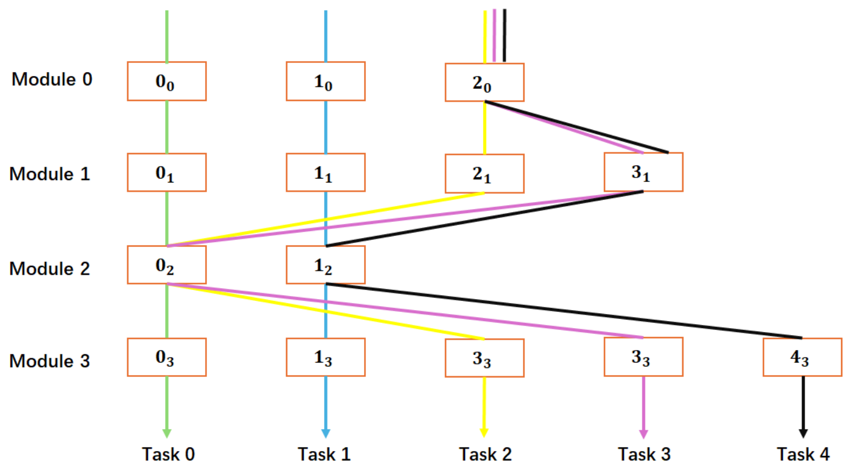

Figure 6: Visualization of subnetwork cooperation during testing in FMNIL. Each task (Task 0–Task 4) is color-coded: green (Task 0), blue (Task 1), yellow (Task 2), pink (Task 3), and black (Task 4). Each node m_n represents a module m used in Task n. Arrows indicate *module reuse*—e.g., Module 2_0 is shared by Task 2, Task 3, and Task 4.