Research Report for CogAI4Sci

M Srinivasan
IIIT-Bangalore
Bangalore, India
m.srinivasan@iiitb.ac.in

***Abstract*—Class-Incremental Learning (CIL) presents a fundamental challenge in machine learning: how to accommodate new classes while preserving knowledge of previously learned ones. While Pre-Trained Models (PTMs) have demonstrated promising results in CIL scenarios, they frequently struggle with catastrophic forgetting when adapting to new tasks. This report examines the Expandable Subspace Ensemble (EASE) framework [9], which addresses these limitations through task-specific adapter modules that create distinct subspaces for different tasks. EASE provides a way to make decisions by combining information from different parts (subspaces) and uses a strategy that generates helpful feature examples for older classes. This process doesn't need access to the original training data, making it efficient and flexible. Comprehensive evaluation across seven benchmark datasets demonstrates EASE's effectiveness in mitigating catastrophic forgetting while efficiently integrating new knowledge. The codes, outputs, logs, my implementation and other related files are available in https://github.com/Srini2404/EASE**

***Index Terms*—Class-Incremental Learning, Catastrophic Forgetting, Expandable Subspace Ensemble.**

I. Introduction

This report presents my comprehensive analysis of the paper titled "Expandable Subspace Ensemble for Pre-Trained Model-Based Class-Incremental Learning" [9]. The study includes detailed examinations of their methodologies, primary research objectives, and proposed solutions. Through hands-on experimentation with the EASE framework, I evaluated its practical implementation and performance metrics. My investigation includes empirical results, their interpretation, and detailed analysis of the framework's effectiveness. The report also presents my insights on potential future improvements. While these suggestions are informed by my understanding and experimentation with the framework and may need further validation by domain experts. The analysis particularly focuses on how these works address the fundamental challenges in continual learning, supported by my experimental findings and theoretical assessments.

II. Expandable Subspace Ensemble for Pre-Trained Model-Based Class-Incremental Learning

Class incremental learning is a machine learning paradigm where a model learns new classes of data incrementally, without forgetting the previously learned classes. Learning new classes often results in overwriting previously learned classes. To address the issue the authors propose Expandable Subspace Ensemble (EASE) for PTM-based Continual Incremental Learning (CIL) by using distinct lightweight adapter modules for each new task. This method enables efficient model updating without harming former knowledge, using task-specific subspaces for joint decision-making and a semantic-guided prototype complement strategy. The authors demonstrate the superiority of their algorithm by evaluating it on seven benchmark datasets, where it achieves state-of-the-art (SoTA) performance.

Existing methods generally involve the use of non PTM based methods (expandable networks, prompt learning etc). But the issue with such methods is that they demand high resource allocation for backbone storage and require the use of exemplars for unified classifier learning or they struggle with forgetting the previous prompts.

A. Research Problem Addressed

The paper addresses two primary challenges in achieving effective Class-Incremental Learning (CIL) with Pre-Trained Models (PTMs):

1) **Constructing low-cost, task-specific subspaces**: Tuning PTMs for new tasks requires substantial computational resources. Thus, there is a need to create and store task-specific subspaces using lightweight adapter modules instead of modifying the entire backbone model.
2) **Developing a classifier for continuously expanding features**: As new classes are learned, the feature space constantly expands. Since exemplars from previous learning stages are unavailable, the classifiers for former stages become incompatible with the latest feature space. The challenge is to leverage class-wise relationships as semantic guidance to synthesize classifiers for previously learned classes.

B. Proposed Solution

The paper proposes solutions to address these challenges, allowing for efficient model updates without catastrophic forgetting. They propose Expandable Subspace Ensemble to tackle the above challenges (EASE). To reduce cross-task conflicts[1], EASE has task-specific subspaces for each incremental task, ensuring that learning new classes does not negatively affect previously learned ones. These subspaces are created by incorporating lightweight adapters into the frozen Pre-Trained Model (PTM), resulting in minimal training and memory costs. Consequently, features from the PTM can be concatenated

[1]Cross-task conflict refers to the interference between different tasks when learning new classes, which can cause previously learned tasks to be forgotten or negatively impacted.

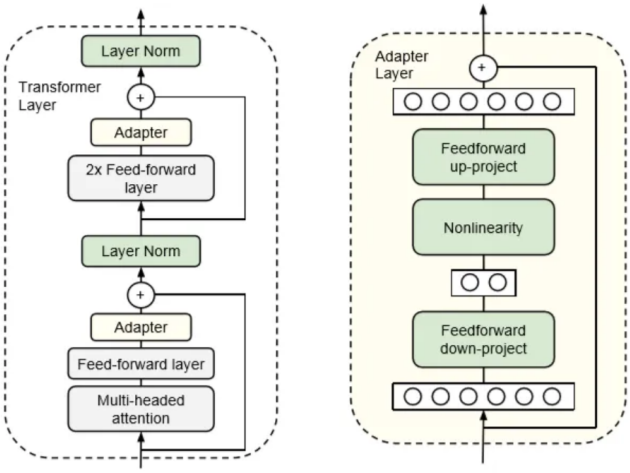

Fig. 1. The standard Transformer is used with an additional adapter layer, added after each sub-layer and before adding the skip connection back. The output of the adapter layer is then forwarded to the layer Normalization. Image taken from Medium article [3].

with each adapter (refer Figure 1) to efficiently aggregate the information.

The following section provides background on the key concepts relevant to my study. I will discuss class-incremental learning (CIL), describing how it helps AI models learn new classes over time. I will then explain pre-trained models and their role in CIL, as well as current baseline approaches and their limitations. Additionally, I will review existing CIL methods and identify their main challenges and shortcomings. This background will help establish the context needed to understand why new approaches like EASE are important for addressing current limitations in the field.

Class-Incremental Learning

Class-Incremental Learning (CIL) is a learning scenario in which a model continuously learns new classes and integrates them into a unified classifier. Consider a sequence of N training sets, denoted as $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N$, where each training set $\mathcal{D}_b = \{(x_i, y_i)\}_{i=1}^{n_b}$ consists of n_b instances. Each instance $x_i \in \mathbb{R}^D$ belongs to a class $y_i \in Y_b$, where Y_b represents the label space of task b. The label spaces for different tasks are non-overlapping, i.e., $Y_b \cap Y_{b'} = \emptyset$ for $b \neq b'$.

We follow the exemplar-free setting described in [4, 6, 7], where no exemplars from previous tasks are stored. During the i-th incremental stage, only the current task's data $\mathcal{D}_i$ is available for training. The goal of CIL is to build a unified classifier for all observed classes $\mathcal{Y}_b = Y_1 \cup \cdots \cup Y_b$ as the data evolves. Specifically, we aim to find a model $f(x) : X \to \mathcal{Y}_b$ that minimizes the expected risk:

$$f^* = \arg\min_{f \in H} \mathbb{E}_{(x,y)\sim \mathcal{D}_1^t \cup \cdots \cup \mathcal{D}_b^t} I(y \neq f(x)) \tag{1}$$

where H is the hypothesis space, $I(\cdot)$ denotes the indicator function, and $\mathcal{D}_b^t$ represents the data distribution for task b.

In Class-Incremental Learning (CIL), we follow the typical practice of leveraging a pre-trained model (PTM) as the initialization for learning new classes [4, 6, 7]. For instance, a Vision Transformer (ViT) [2] can be used as a PTM. The advantage of using a PTM is that it provides a strong foundation for feature extraction, which helps to reduce the training burden for new tasks.

Decoupling the PTM:

- **Feature Embedding Function** ($\phi(\cdot) : \mathbb{R}^D \to \mathbb{R}^d$): This function maps the input data, $x_i \in \mathbb{R}^D$, to a lower-dimensional feature space, $z_i \in \mathbb{R}^d$. In the case of the Vision Transformer (ViT) [2], the embedding function $\phi(\cdot)$ refers to the output of the special [CLS] token, which aggregates information from the entire image.
- **Linear Classifier** ($W \in \mathbb{R}^{d \times |Y_b|}$): The linear classifier takes the feature embeddings from $\phi(x)$ and maps them to the predicted class. The model output is defined as:

$$f(x) = W^\top \phi(x) \tag{2}$$

where $W^\top$ is the transpose of the classifier matrix W. The dimensions of W are $d \times |Y_b|$, where d is the feature dimension and $|Y_b|$ is the number of classes for the current task b.

Breaking Down the Classifier

The classifier matrix W can be further decomposed into individual weight vectors for each class:

$$W = [w_1, w_2, \ldots, w_{|Y_b|}] \tag{3}$$

where each w_j is the weight vector corresponding to class j. These weight vectors are used to decide whether an input belongs to class j based on the feature embedding $\phi(x)$. Thus, the output prediction is a weighted combination of the feature embeddings produced by the pre-trained model.

This approach decouples the **feature extraction** (handled by $\phi(x)$) from the **classification** (handled by W), allowing the pre-trained model to remain largely unchanged while the classifier is updated to accommodate new tasks.

Standard Methods of CIL

- **Learning with PTMs**: One of the standard procedure is to freeze most of the weights of a PTM and then only train a small, learnable prompt[2]. This *Pool* influences the self-attention process of the model, helping the PTM incorporate new task-specific information. The target is written as

$$\min_{\text{Pool},U,W} \sum_{(x,y)\in D^b} \ell(W^\top \phi(x; \text{Pool}), y) + \mathcal{L}_{\text{Pool}}, \tag{4}$$

[2]Prompts are special tokens that are learnable (i.e., they are updated during training). Here they have the same dimensions as the image patch embeddings used by models like Vision Transformers. In ViT [2], an image is split into patches, each of which is represented by an embedding vector. The prompts are added to these patch embeddings during the self-attention process to encode task-specific information pool (denoted as *Pool*)

where $\ell(\dots)$ is the cross-entropy loss that measures the discrepancy between prediction and ground truth. $\mathcal{L}_{\text{Pool}}$ denotes the prompt selection or regularization term for prompt training. Optimizing Equation 4 encodes the task information into these prompts, enabling the PTM to capture more class-specific information as data evolves.

- **Learning with expandable backbones:** In continual learning scenarios, training on new classes can lead to conflicts with previously learned classes, causing the model to forget its earlier knowledge. To mitigate this issue, methods incorporating model expansion have been proposed [5, 8]. Specifically, when a new task arises, the existing backbone $\bar{\phi}_{\text{old}}$ is frozen and retained in memory while a new backbone ϕ_{new} is initialized. The model then combines the embedding functions $[\bar{\phi}_{\text{old}}(\cdot), \phi_{\text{new}}(\cdot)]$ and initializes a larger fully-connected layer $W_E \in \mathbb{R}^{2d\times|Y_b|}$. The training objective optimizes the cross-entropy loss over the combined dataset:

$$\min_{\phi_{\text{new}}\cup W_E} \sum_{(x,y)\in D^b\cup\mathcal{E}} \ell\left(W_E^\top \left[\bar{\phi}_{\text{old}}(x), \phi_{\text{new}}(x)\right], y\right) \quad (5)$$

  where $\mathcal{E}$ denotes the exemplar set containing instances of previous classes, which may not be accessible in the current scenario.
  For instance, if the first task includes 'cats', the old embedding is specialized in capturing features such as patterns and textures due to model capacity constraints. However, when encountering a new task that includes 'birds', rather than overwriting the established features in $\bar{\phi}_{\text{old}}$, we utilize the new backbone ϕ_{new} to learn distinctive features such as beaks and feathers. The combined feature representation allows the model to acquire new information while preserving existing knowledge, calibrating its classifier with the exemplars across all observed classes.

But these methods are costly and aren't memory efficient and additionally, since we do not have any exemplars $\mathcal{E}$, optimizing also fails to achieve a well calibrated classifier for all seen classes. This lead the authors to think if there is a possibility of using lightweight subspaces and do it without the use of exemplars as they are difficult to obtain for tasks.

EASE: The authors came up with the idea of using expandable subspaces of sequential tasks. But since the authors were aiming at providing the solution without using exemplars, the authors had to create and complete the expanding classifier and calibrate the predictions among different tasks without using previous instances. Equation 4, requires large computational cost and memory budget to fine-tune and save the model. Hence we use the adapter module for fine-tuning. The architecture is shown Fig 1.

Let L represent the number of transformer blocks within the pre-trained model, where each block consists of a self-attention module and a multi-layer perceptron (MLP). In line with the approach outlined in [1], we introduce an adapter module as a supplementary branch associated with the MLP. Specifically, an adapter functions as a bottleneck structure, incorporating a down-projection layer $W_{\text{down}} \in \mathbb{R}^{d\times r}$, a non-linear activation function σ, and an up-projection layer $W_{\text{up}} \in \mathbb{R}^{r\times d}$. The output of the MLP is modified as follows:

$$x_o = \sigma(x_i W_{\text{down}})W_{\text{up}} + \text{MLP}(x_i), \quad (6)$$

where x_i and x_o denote the input and output of the MLP, respectively. Equation 6 captures the task-specific information by appending the residual component to the original output.

The collection of adapters across all L transformer blocks is defined as $\mathcal{A}$, leading to an adapted embedding function with the adapter denoted by $\phi(x;\mathcal{A})$. Consequently, when confronted with a new incremental task, we can freeze the pre-trained weights of the model and optimize only the adapter by solving the following optimization problem:

$$\min_{\mathcal{A}\cup W} \sum_{(x,y)\in\mathcal{D}^b} \ell\left(W^\top \bar{\phi}(x;\mathcal{A}), y\right). \quad (7)$$

where $\mathcal{D}_b$ represents the dataset associated with the new task and ℓ signifies the loss function.

Accordingly, the frozen pre-trained backbone is shared while expandable adapters are learned for each new task. During the learning phase of task i, a new adapter $\mathcal{A}_i$ is initialized, and Equation 7 is optimized to learn task-specific subspaces. This results in a collection of i adapters: $\{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_i\}$. Consequently, the concatenated features across all subspaces can be obtained by combining the pre-trained backbone with each adapter:

$$\Phi(x) = [\phi(x;\mathcal{A}_1), \phi(x;\mathcal{A}_2), \dots, \phi(x;\mathcal{A}_i)] \in \mathbb{R}^{id}. \quad (8)$$

Figure 2 (left and middle) illustrates the process of adapter expansion. Since we fine-tune only the task-specific adapter for each corresponding task, training on a new task does not interfere with previously learned knowledge (i.e., the earlier adapters remain unaffected). In addition, as shown in Equation 8, we combine the pre-trained embedding with different task-specific adapters to produce the final representation. This embedding encapsulates all task-specific information across various subspaces, which can be integrated for a comprehensive prediction. Furthermore, because adapters are lightweight branches, they require significantly fewer parameters and resources compared to fully fine-tuning the backbone. The parameter overhead for storing these adapters is given by $B \times L \times 2dr$, where B is the number of tasks, L is the number of transformer blocks, and $2dr$ represents the number of parameters for each adapter (i.e., linear projections).

We then use a prototype-based classifier for prediction, after the training process of each incremental stage, we extract the class prototype of the i-th class in adapter $\mathcal{A}_b$'s subspace.

$$\hat{p}_{i,b} = \frac{1}{N}\sum_{j=1}^{|\mathcal{D}^b|} \mathbb{I}(y_j = i)\phi(\mathbf{x}_j;\mathcal{A}_b) \quad (9)$$

Equation 9 is used to extract the most representative pattern of the class in corresponding embedding space. This can

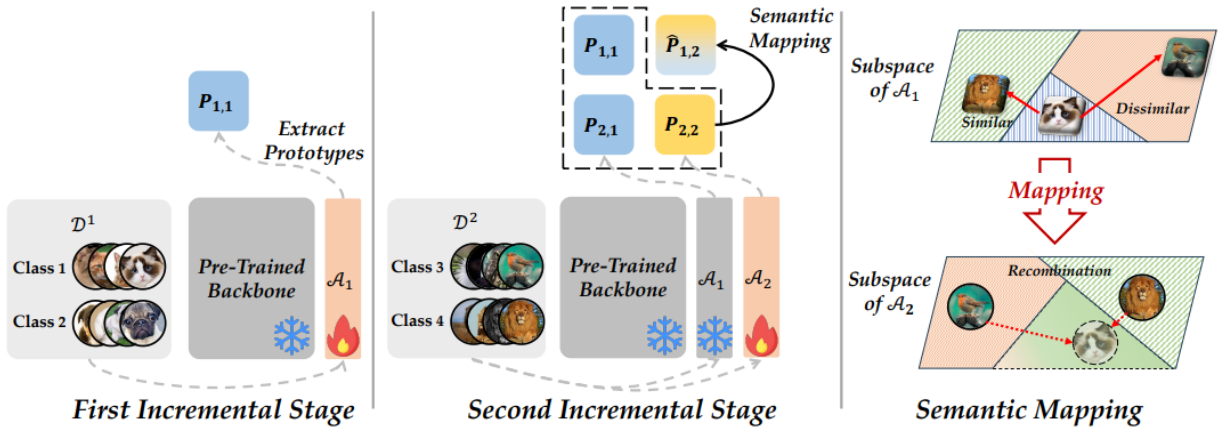

Fig. 2. Illustration of EASE. Left: In the first task, we learn an adapter A_1 to encode task specific features, and extract class prototypes $P_{1,1}$. Middle: In the second task, we initialize a new adapter A_2 to encode new features, and extract prototypes $P_{2,1}$ and $P_{2,2}$. Without exemplars, we need to synthesize $P_{1,2}$ (old class prototypes in the new subspace) for prediction. Right: Semantic mapping process. We extract class-wise similarity in the co-occurrence subspace and utilize it to synthesize old class prototypes in the target space. Image taken from the original paper [9].

further be used in concatenation of the prototypes from all the adapters' embedding spaces.

$$\mathcal{P}_i = [p_{i,1}, p_{i,2}, \ldots, p_{i,b}] \in \mathbb{R}^{bd} \tag{10}$$

Equation 10 denotes the classifier for i's classifier. The classification embedding $\Phi(x)$ and then concatenated prototype i.e.

$$p(y|x) \propto \text{sim}(P_y, \Phi(x)) \tag{11}$$

.

For the similarity we will be using the *cosine similarity*.

Equation 9 constructs classifiers using representative prototypes. However, when a new task arrives, a new subspace needs to be learned with a corresponding adapter. This requires recalculating all class prototypes in the updated subspace to align them with the newly expanded embeddings. The challenge arises from the lack of exemplars for old classes, making it difficult to estimate their prototypes in the new subspace.

For instance, in the first stage, we train the adapter $\mathcal{A}_1$ with the first dataset $\mathcal{D}^1$ and extract prototypes for the classes in $\mathcal{D}^1$, represented as:

$$\mathcal{P}_{1,1} = \text{Concat}[p_{1,1}, \ldots, p_{|\mathcal{Y}_1|,1}] \in \mathbb{R}^{|\mathcal{Y}_1| \times d}. \tag{12}$$

Here, the first subscript in $\mathcal{P}_{1,1}$ denotes the task index, while the second subscript indicates the subspace.

In the subsequent stage, we introduce an adapter $\mathcal{A}_2$ for the second dataset $\mathcal{D}^2$. Since only $\mathcal{D}^2$ is available, we can only compute the prototypes of classes in $\mathcal{D}^2$ within both the subspaces of $\mathcal{A}_1$ and $\mathcal{A}_2$, i.e., $\mathcal{P}_{2,1}$ and $\mathcal{P}_{2,2}$. *We can't calculate the prototypes of old classes in the new embedding space directly i.e.* $\mathcal{P}_{1,2}$.

To calculate the prototypes of old classes in the new embedding space, we take help from the similar classes scores. We especially consider semantic information in the co-occurrence space and restore the prototypes by combining the related prototypes. We measure the similarity between old and new classes in the old subspace and utilize it to calculate the similarity in the new embedding space.

$$\text{Sim}_{i,j} = \frac{\mathbf{P}_{o,o}[i] \cdot \mathbf{P}_{n,o}[j]^\top}{\|\mathbf{P}_{o,o}[i]\|_2 \|\mathbf{P}_{n,o}[j]\|_2}, \tag{13}$$

Equation 13 is used to calculate the class-wise similarity among class is calculated via prototypes in the co-occurrence subspace. Then we normalize this value via softmax. Index i and j in Equation 13 correspond to ith and jth class respectively. Using Equation 13 we can construct the similarity matrix for all of the classes and then the prototype for the old class in the new subspace can be calculated by weighted combination of new class (given by equation 14).

$$\hat{\mathbf{P}}_{o,n}[i] = \sum_j \text{Sim}_{i,j} \times \mathbf{P}_{n,n}[j]. \tag{14}$$

After performing adapter expansion and prototype complement, the full classifier (prototype matrix) can be represented as:

$$\begin{bmatrix} \mathbf{P}_{1,1} & \hat{\mathbf{P}}_{1,2} & \cdots & \hat{\mathbf{P}}_{1,B} \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \cdots & \hat{\mathbf{P}}_{2,B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{B,1} & \mathbf{P}_{B,2} & \cdots & \mathbf{P}_{B,B} \end{bmatrix} \tag{15}$$

where items above the main diagonal are estimated using equation 14. During inference, the logit for task b is computed as:

$$[\mathbf{P}_{b,1}, \mathbf{P}_{b,2}, \ldots, \mathbf{P}_{b,B}]^\top \Phi(x) = \sum_i \mathbf{P}_{b,i}^\top \phi(x; \mathcal{A}_i) \tag{16}$$

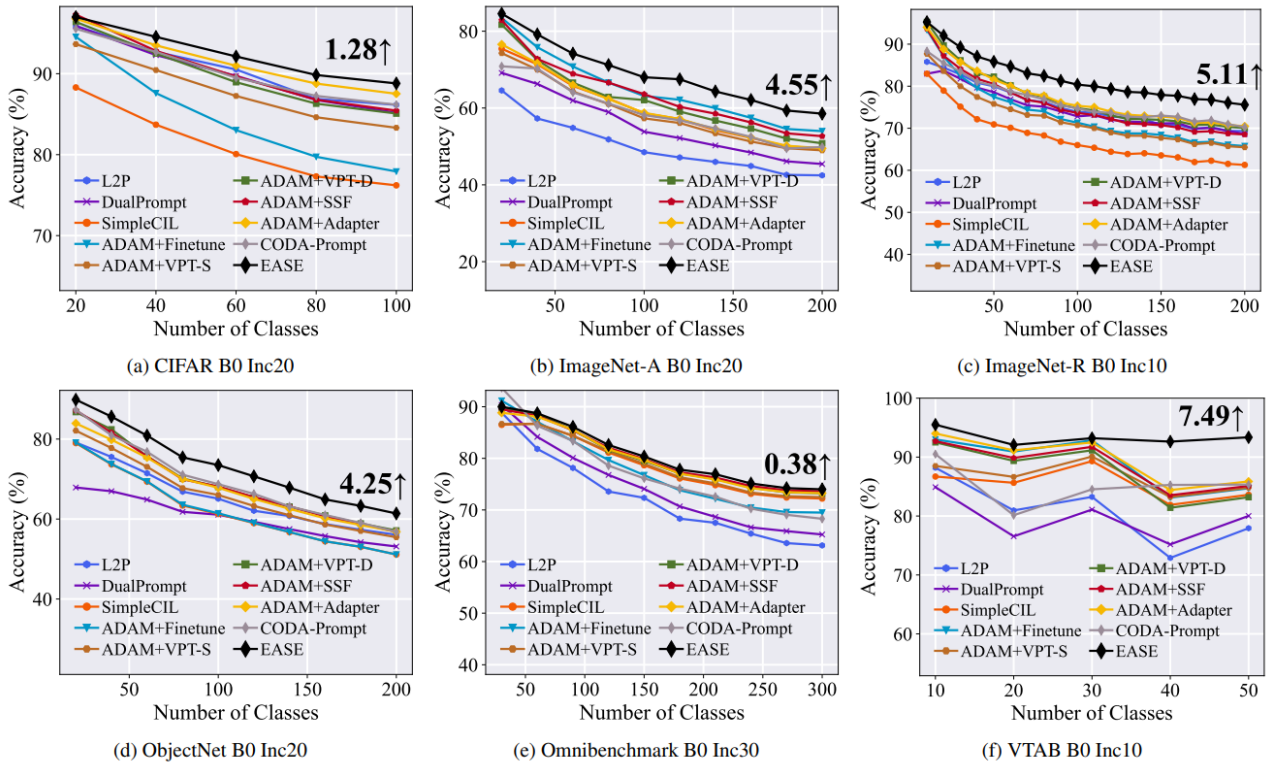

Fig. 3. Performance curve of different methods under different settings. All methods are initialized with ViT-B/16-IN1K. We annotate the relative improvement of EASE above the runner-up method with numerical numbers at the last incremental stage. Image taken from the original paper [9].

which corresponds to an ensemble of multiple (prototype-embedding) matching logits across different subspaces. Among these terms, only the adapter $\mathcal{A}_b$ is specifically trained to extract task-specific features for the b-th task. Therefore, these prototypes are considered to be more effective in classifying the corresponding task and should have a greater influence during the final inference.

To emphasize the importance of the task-specific subspace, we modify Equation 16 by assigning greater weights to the matching subspace:

$$P_{b,b}^{\top}\phi(x;A_b) + \alpha \sum_{i \neq b} P_{b,i}^{\top}\phi(x;A_i) \tag{17}$$

where α is the trade-off parameter, set to 0.1 in our experiments. By re-weighting the logits, we highlight the contribution of the core features in making the final decision.

III. Code Implementation

I have attempted to reproduce the experiments from the paper. However, due to time constraints and limited access to high-performance GPUs, I was unable to train and experiment with all the Class-Incremental Learning (CIL) methods mentioned, such as L2P [7] and ADAM [10]. I was able to reproduce the experiments for the EASE method [9]. While I tried implementing other methods using their respective source codes, many repositories were outdated, leading to persistent errors during training that required significant debugging.

Given the time limitations and computational challenges, I decided to focus solely on reproducing the results for the Expandable Subspace Ensemble (EASE) method. I was able to successfully train the EASE model on six benchmark datasets (except on the objectnet dataset- owing to the size), using the original source code from their repository. However, I had to make several modifications to ensure a successful run.

The training was conducted on the lab system, but access to the full computational capacity was limited due to shared use with other students. Despite these constraints, I adhered to the conditions specified for EASE and logged the training details for each dataset. These log files have been uploaded to my GitHub repository for reference. All the implementation is done using **ViT-B/16-IN1K** model.

All codes and results for this work can be found in Github whose link is Here.

IV. Results

I have plotted the accuracy plots for the training I did on these datasets, with the ViT model [2]. I have plotted the average accuracy and the top-1 and top-5 accuracy for the same. The definitions of top-1 and top-5 accuracy is mentioned in the below part.

Figure 4 shows the training results on CIFAR dataset.

Top-1 Accuracy: The fraction of instances where the model's highest probability prediction matches the true label. This is defined as:

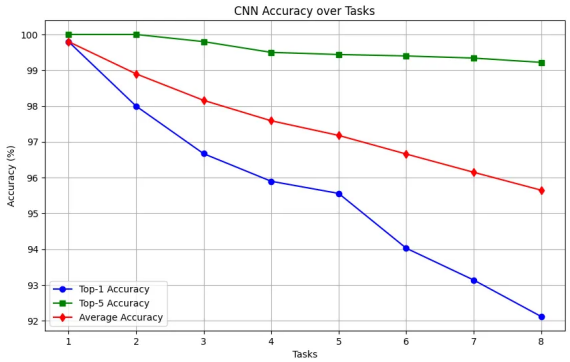

Fig. 4. Performance over CIFAR224 dataset.

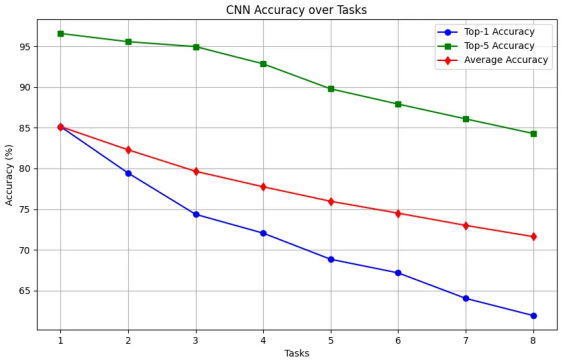

Fig. 5. Performance over ImageNet-A dataset.

$$\text{Top-1 Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i)$$

where y_i is the true label for the i-th example, $\hat{y}_i$ is the predicted label, and $\mathbb{1}$ is the indicator function that is 1 when the condition is true and 0 otherwise.

Top-5 Accuracy: The fraction of instances where the true label is among the top 5 predicted classes (ranked by predicted probability). This is defined as:

$$\text{Top-5 Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i \in \hat{y}_i^5)$$

where $\hat{y}_i^5$ represents the top 5 predicted classes for the i-th example.

Inference on EASE Performance

The results demonstrate that EASE consistently achieves the highest accuracy across various datasets (CIFAR, ImageNet, ObjectNet, VTAB) as the number of classes increases. EASE exhibits a smaller decline in accuracy.

EASE's architecture, particularly its expandable subspace ensemble, likely mitigates *catastrophic forgetting*, allowing the model to retain previously learned information while learning

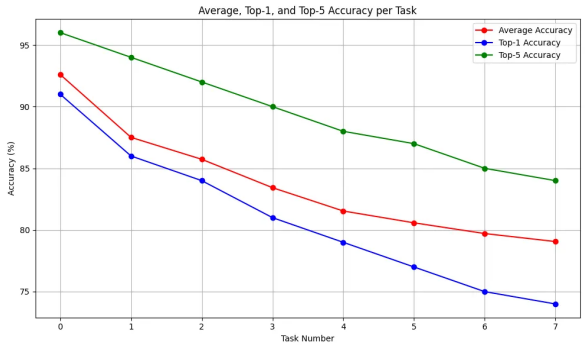

Fig. 6. Performance over ImageNet-R dataset.

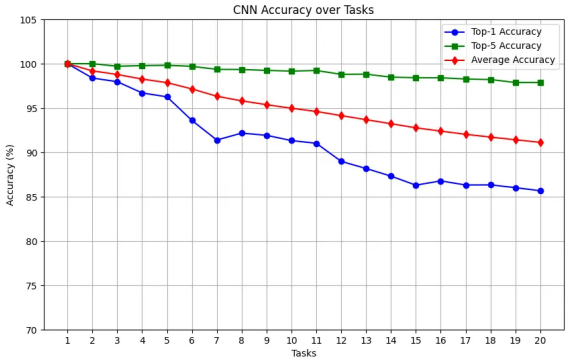

Fig. 7. Peforamnce over CUB dataset.

new classes. This adaptability is particularly beneficial in complex datasets like ImageNet-R and VTAB, where EASE achieves significant performance.

In conclusion, EASE demonstrates robust generalization across diverse datasets and scales effectively in Class-Incremental Learning tasks, making it a strong candidate for scenarios with a growing number of classes.

Figures 3 is from the original paper [9], the authors have compared the EASE model along with other existing CIL

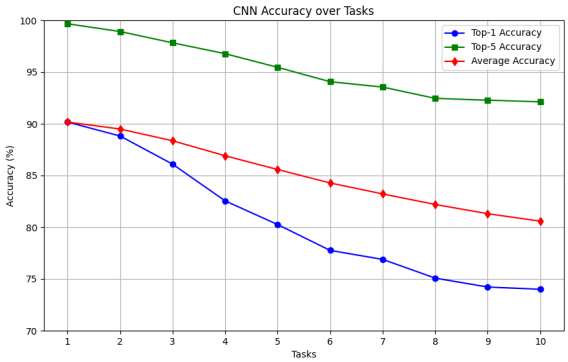

Fig. 8. Peformance over Omnibenchmark data.

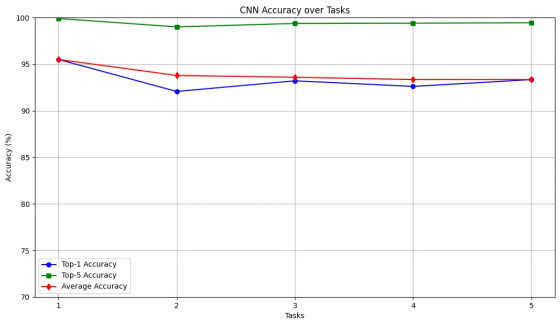

Fig. 9. Performance over Vtab dataset.

methods such as L2P [7], DualPrompt [6], ADAM+Finetune [10] etc. The results clearly show that EASE dominates on all of the benchmark datasets they have tested with.

The results in the graph show the performance of various Class-Incremental Learning (CIL) methods, including EASE, across different datasets like CIFAR, ImageNet, ObjectNet, and VTAB.

General Observations:

- **EASE's Consistent Performance:** Across all datasets, EASE consistently achieves the highest or near-highest accuracy, particularly as the number of classes increases. This demonstrates EASE's effectiveness in handling the growing complexity of class-incremental learning tasks.
- **Significant Accuracy Gains:** The accuracy improvements of EASE over competing methods are significant, as indicated by the arrows in the graphs. For instance, in VTAB, EASE shows a 7.49% improvement over other methods, highlighting its robustness in handling this challenging dataset and setup (incrementing classes in blocks of 10).
- **Smaller Deterioration in Accuracy:** While the other methods show a steady decline in accuracy as the number of classes increases, EASE maintains higher performance levels. This suggests that EASE manages the complexities of increasing classes better than its competitors. The expandable subspace ensemble likely preserves discriminative power as the number of classes grows.

Why Are the Results Like This?

- **EASE's Architecture:** The expandable subspace ensemble design in EASE helps mitigate catastrophic forgetting, which is a critical issue in class-incremental learning. By maintaining different subspaces for different classes or tasks, EASE is likely able to preserve previously learned information while learning new classes. Its adaptability with minimal performance degradation across datasets like CIFAR, ImageNet, and ObjectNet suggests that EASE generalizes well across various domains.
- **Benchmarks and Dataset Complexity:** Datasets like VTAB and ImageNet-R pose higher challenges due to their variability in data distribution, yet EASE still achieves significant improvements. This indicates that EASE's ability to expand and adapt is particularly beneficial in handling more complex datasets. On the other hand, simpler models like L2P [7] exhibit larger drops in accuracy, likely due to their limitations in retaining previously learned information as the number of classes grows.
- **Computational Complexity:** Although EASE's ensemble approach might introduce additional computational overhead compared to simpler methods, the performance gains justify this complexity. The architecture's ability to handle class-incremental learning tasks effectively shows that the computational trade-offs are worthwhile.

V. Future Work

While the use of lightweight adapters significantly reduces the parameter footprint, several avenues can be explored to further enhance the efficiency and performance of the proposed approach:

- **Dynamic Adapter Selection:** Instead of using separate adapters for each task, dynamic selection and reuse of adapters based on task similarity could be explored. This would reduce the number of saved adapters and optimize memory usage.
- **Task-Specific Sparsity:** Applying task-specific sparsity during adapter fine-tuning can activate only relevant parts of the network, leading to a more parameter-efficient model while maintaining task-specific information.
- **Knowledge Distillation for Adapter Compression:** Leveraging knowledge distillation to compress the adapters by training a smaller model to imitate the outputs of the larger model could reduce the overall size of the model without significant performance degradation.
- **Parameter-Efficient Fine-Tuning (PEFT):** Techniques such as Low-Rank Adaptation (LoRA) or Prompt Tuning can be explored to fine-tune only a small subset of parameters, minimizing memory overhead while allowing effective class-incremental learning.
- **Efficient Multi-Task Learning:** Investigating the possibility of using a single adapter for multiple tasks with task-specific routing mechanisms could reduce the number of adapters, leading to a more efficient and scalable approach.

References

[1] Shoufa Chen et al. *AdaptFormer: Adapting Vision Transformers for Scalable Visual Recognition*. 2022. arXiv: 2205.13535 [cs.CV]. URL: https://arxiv.org/abs/2205.13535.

[2] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[3] James Peters. *Adapters: A Compact and Extensible Transfer Learning Method for NLP*. Accessed: 2024-10-23. 2020. URL: https://medium.com/dair-ai/adapters-a-compact-and-extensible-transfer-learning-method-for-nlp-6d18c2399f62.

[4] James Seale Smith et al. *CODA-Prompt: COntinual Decomposed Attention-based Prompting for Rehearsal-Free Continual Learning*. 2023. arXiv: 2211.13218 [cs.CV]. URL: https://arxiv.org/abs/2211.13218.

[5] Fu-Yun Wang et al. *FOSTER: Feature Boosting and Compression for Class-Incremental Learning*. 2022. arXiv: 2204.04662 [cs.CV]. URL: https://arxiv.org/abs/2204.04662.

[6] Zifeng Wang et al. *DualPrompt: Complementary Prompting for Rehearsal-free Continual Learning*. 2022. arXiv: 2204.04799 [cs.LG]. URL: https://arxiv.org/abs/2204.04799.

[7] Zifeng Wang et al. *Learning to Prompt for Continual Learning*. 2022. arXiv: 2112.08654 [cs.LG]. URL: https://arxiv.org/abs/2112.08654.

[8] Shipeng Yan, Jiangwei Xie, and Xuming He. *DER: Dynamically Expandable Representation for Class Incremental Learning*. 2021. arXiv: 2103.16788 [cs.CV]. URL: https://arxiv.org/abs/2103.16788.

[9] Da-Wei Zhou et al. "Expandable Subspace Ensemble for Pre-Trained Model-Based Class-Incremental Learning". In: *CVPR*. 2024, pp. 23554–23564.

[10] Da-Wei Zhou et al. *Revisiting Class-Incremental Learning with Pre-Trained Models: Generalizability and Adaptivity are All You Need*. 2024. arXiv: 2303.07338 [cs.LG]. URL: https://arxiv.org/abs/2303.07338.