IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 March 2025, accepted 26 March 2025, date of publication 31 March 2025, date of current version 9 April 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3556471

TOPICAL REVIEW

Opportunistic Dynamic Architecture for Class-Incremental Learning

FAHRURROZI RAHMAN, ANDREA ROSALES SANABRIA, AND JUAN YE
School of Computer Science, University of St Andrews, KY16 9AJ St Andrews, U.K.
Corresponding author: Fahrurrozi Rahman (fr27@st-andrews.ac.uk)

This work was supported by the Leverhulme Research Project under Grant RPG-2021-355.

ABSTRACT Continual learning has attracted increasing attention over the last few years, as it enables to continually learn new tasks over time, which has significant implication to many real-world applications. A large number of continual learning techniques are proposed and achieve promising performance; however, many of them commit to a fixed, large architecture at the beginning, which can waste the memory space and incur high training cost. To directly tackle this challenge, we propose an *Opportunistic Dynamic Architecture*, ODA, based on mixture of experts. ODA can automatically grow with more experts for new incoming tasks and opportunistically shrink by merging experts with similar weights. We evaluated ODA on three commonly used datasets: CIFAR-100, CUB, and iNaturalist, and compared against eight existing continual learning techniques. ODA not only outperforms these techniques but does so with a parameter size that is slightly smaller on average, maintaining memory efficiency without compromising accuracy. Furthermore, ODA achieves this with only around 16% of the training time across all datasets when updating for each new task, making it a highly resource-efficient solution for continual learning applications.

INDEX TERMS Class-incremental learning, continual learning, mixture of expert.

I. INTRODUCTION

In recent years we have witnessed advanced development in artificial neural networks and deep learning. The advancement has been widely used in a range of domains including medicine, transportation, agriculture, and finance. One key question emerging in the deep learning community is *how to make these deep learning models automatically update and evolve over time* so that they can be deployed and used over an extended period.

This research is often referred to as continual learning, which is the ability to learn new tasks and accumulate knowledge continuously while retaining previously learned information. It is crucial in many real-world applications, including improving personalised healthcare by continually updating their knowledge on patients' symptoms and recovery progression [1], [2], and enhancing driving safety of autonomous vehicles by continually updating its understanding of road conditions, traffic patterns, and unexpected events [3], [4]. The challenge in these example applications lies in the unpredictability of new tasks and task sequences, as well as the impracticality of learning all tasks simultaneously. Therefore, machine learning models need to learn tasks in a streaming manner, which leads to *catastrophic forgetting*. That is, knowledge related to old tasks is progressively overwritten while learning new tasks, resulting in a decline in overall performance over time.

Prior attempts to mitigate catastrophic forgetting have primarily involved preserving the weights associated with old tasks [5] or incorporating strategies such as replaying samples from old tasks [6]. While these approaches have demonstrated promising performance, the main limitation lies in their early commitment to a large network, even when the initial task includes only a limited number of classes (e.g., 2 or 5 classes). This architectural choice can lead to substantial memory footprint and prolonged training times due to the abundance of parameters.

Recent advancements in continual learning have shifted towards dynamic architectures, where the network's structure evolves to accommodate an increasing number of tasks [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Yiming Tang.

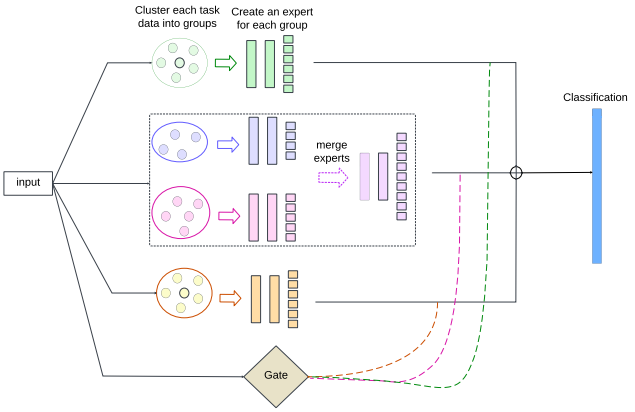

FIGURE 1. The workflow of opportunistic dynamic architecture (ODA), which clusters each task's data into groups, creates or assigns an expert for each group, and merges experts opportunistically.

This progressive approach aims to optimise the network's capacity in response to task variations, offering a potential solution to the challenges posed by catastrophic forgetting while minimising resource inefficiencies. This paper introduces an opportunistic dynamic architecture (ODA), where the network proactively identifies opportunities to both expand and compress its structure. This adaptive approach is designed to accommodate new tasks effectively while also addressing memory limitations.

ODA (as shown in Figure 1) is built on top of a Mixture of Experts (MoE), which assigns an expert to a specific task, with a gating mechanism to regulate expert selection and integration. MoE [8], [9] is an emerging architecture for many language models because of its computational efficiency. MoE in ODA is different from these MoE architectures in that we host a collection of heterogeneous experts trained independently for different tasks. In the context of continual learning, naively mapping an expert to each new task can result in a large number of experts, leading to the accumulation of redundant weights. To tackle this problem, ODA continuously clusters similar classes from incoming tasks into groups. An expert is created for each group, allowing for updates with classes that share similarities with the expert's existing classes. ODA allows to merge similar experts to accommodate on-demand memory constraints. This dynamic strategy optimises the architecture for memory efficiency and task adaptation in continual learning.

The contributions of our work are listed as follows.

1) We propose a novel MoE-based dynamic architecture approach for continual learning that automatically grows and compresses the structure of the network in response to incoming tasks and memory restrictions of devices.
2) We evaluate our approach on three widely-used datasets: Canadian Institute for Advanced Research (CIFAR-100) [10], Caltech-UCSD Birds (CUB) [11], and iNaturalist [12]. The comparison with the state-of-the-art continual learning techniques demonstrates that ODA can achieve comparable accuracy (76% on CIFAR-100, 88% on CUB, and 71% on iNaturalist) while maintaining a small training time; the average time is 37 seconds per task and can be as low as 12% of the time to the most expensive techniques.

II. RELATED WORK

The main challenge of continual learning is *catastrophic forgetting* (CF) and representative strategies for addressing CF include regularisation, rehearsal, and dynamic networks. Regularisation-based approaches penalise large updates on important parameters identified to a certain task, including knowledge distillation in learning without forgetting (LWF) [5] and elastic weight consolidation (EWC) [13]. However, regularisation alone does not prevent forgetting and recent approaches combine regularisation with rehearsal; that is, they store or generate samples for old tasks and merge these samples with new tasks' data when updating the network. A classic approach is Incremental Classifier and Representation Learning (iCaRL) [6] that employs a herding algorithm to select the most informative samples for each task and utilise knowledge distillation when updating the network. More recent approaches adopt class prototypes to characterise old classes' distribution [14]. However, the main limitation of these approaches is that they assume a fixed network, whose parameters may saturate eventually when there is a high volume of tasks to learn.

Recent research has increasingly focused on dynamic networks, which have the advantage of dynamically extending the network's representational capacity. These approaches include extending neurons or branches and using masks to select parameter subspaces for each task. In the following, we introduce representative approaches in each category.

Early works focus on neuron expansion; that is, adding neurons to a certain layer, or introducing a new layer or a new subnetwork to the existing model. For example, ProgressNet [15] creates a new network for each task and uses the task ID to select which network for prediction. However, it is impractical to know the task ID *a priori* so ProgressNet only works in task-incremental learning, not in class-incremental learning. To improve on this, Progress & Compress [16] progresses the architecture with new parameters for each new task and then compresses by distilling the new parameters into the shared knowledge base. Similarly, ExpertGate [17] is composed of a collection of expert models, each corresponding to a task. Each expert is designed as an autoencoder, and expert selection is based on the task relevance score based on the reconstruction errors. Based on the relevance score, the model decides whether to create a new expert or reuse and fine-tune an existing one. Dynamically expandable representation (DER) [18] introduces a new backbone network for each new task and aggregates the features from all backbones into *super-feature*. However, this approach often accumulates a large number of redundant parameters, requiring substantial memory size and

extended training time. To tackle this problem, dynamically expandable networks (DEN) [19] is proposed to add new neurons only when necessary; that is, when the retraining loss for a new task exceeds a threshold. It also eliminates unnecessary neurons using group-sparsity regularisation.

Another strand is to use masks to allocate parameter subspaces for each task; that is, a binary mask is applied to the neurons or parameters to indicate whether they will be updated (if the mask value is 1) or frozen. This helps mitigate the forgetting effect by reducing interference during model updates for different tasks, while still allowing parameter sharing across tasks for feature learning. For example, PackNet [20] identifies important neurons for the current task and releases unimportant neurons for future tasks. This is achieved via iterative pruning, activation values, and uncertainty estimation. Piggyback [21] proposes learning binary masks without updating the backbone network. The task-specific binary mask weight matrix 'piggybacks' onto the backbone network.

Additive parameter decomposition (APD) [22] decomposes the network parameters at each layer of the target network. When a new task arrives, it first utilises the shared parameters and learns the incremental difference that requires task-specific parameters. To improve the effectiveness of parameter sharing, APD clusters the task-specific parameters into hierarchically shared parameter groups. However, this approach still assumes a fixed network, whose capacity is limited, therefore, their parameters will saturate when a high number of tasks are learnt. Hence, this approach requires sparsity constraints on parameter usage and selective reuse of the old parameters, which might affect the learning of new tasks [23].

Else-net [24] is another example of a dynamic architecture, consisting of multiple layers of elastic units, each of which comprises several learning blocks. Each block stores different knowledge from different human actions and the selection of a relevant block is managed by a switch block. While dynamic architectures in continual learning show good performance, Zhou et al. argue that the same memory size to extend the model can be used for the exemplar buffer to improve standard models [25]. They compare three exemplar-based algorithms with two model-based ones, giving the exemplar-based approaches more memory. They found the best configuration by sharing the shallow layers in the model backbone and only creating deep layers for new tasks. Built on this work, we apply pre-trained feature extractors and only create experts for new tasks.

To maintain low memory overhead when the tasks grow as well as at the test time, Douillard et al. [26] expand the model with special tokens for each task. These tokens have the same dimension as the data features and are concatenated to the input during training and prediction. As the new tokens are only created with linear size, the total token size for all tasks is still considerably small compared to expanding the backbone of the model. Xie et al. [27] apply the expansion technique to capture new domain representation in domain incremental learning (DIL). The intra-class structure is learnt using von Mises-Fisher (vMF) mixture model. After expanding the mixture model in each training step, a reduction strategy by merging existing clusters is applied to make it more compact.

Compared to the above approaches, our method expands and merges experts based on class similarity and memory constraints. It does not require prior knowledge on task IDs, nor does it store extra parameters such as masks.

III. OPPORTUNISTIC DYNAMIC ARCHITECTURE

This section describes our proposed technique, ODA, which is built on the MoE framework, with each expert assigned to a set of similar classes. The key novelty of ODA resides in its automated expansion and expert fusion capabilities. For each task, classes are first clustered into groups, after which an expert is created and trained for each group. Expert weights are monitored to merge experts with similar weights to reduce memory redundancy when needed. The following sections describe the two main steps: expert creation and fusion, with a focus on how to create and train experts, integrate predictions from experts, and merge experts to mitigate catastrophic forgetting in both the experts and gates.

A. PROBLEM STATEMENT

Class-incremental continual learning refers to a sequence of tasks with mutually exclusive class sets. Each task T_i is associated with a dataset $D_i = \{(x_i^{(j)}, y_i^{(j)}) | y_i^{(j)} \in C_i\}_{j=1}^{N_i}$, where $x_i^{(j)}$ is the jth input feature and $y_i^{(j)}$ is the output label belonging to the corresponding class set C_i. The objective is to recognise all the classes from the tasks being observed.

B. EXPERT CREATION

The initial phase involves the aggregation of similar classes to form a cohesive expert group. Drawing on inspiration from the multi-task learning literature [28], we group similar classes to enhance overall classification performance. We illustrate this concept via a toy example by randomly selecting 20 classes from CIFAR-100. Three distinct grouping settings are explored: based on their superclass, clustering, and random assignment. An expert is created for each group, and the classification accuracy of the MoE across these settings is evaluated. As presented in Figure 2, clustering-based grouping demonstrates accuracies close to those of superclass-based grouping, both of which outperform random grouping. This example highlights that building experts for similar classes can improve classification accuracy.

1) PROTOTYPING AND CLUSTERING

For each incoming task, we create a prototype using a Gaussian Mixture Model (GMM) for each class, and then run a clustering algorithm on the prototypes. This reduces the computational complexity of clustering and prevents samples from the same class being assigned to different groups.

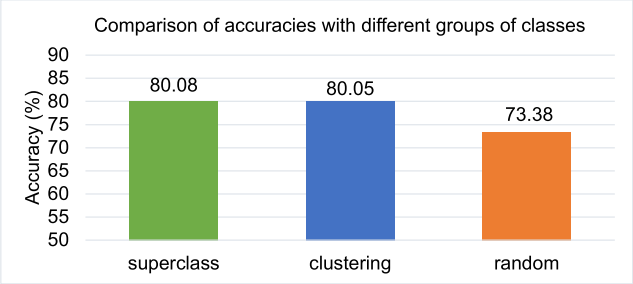

FIGURE 2. A toy example of investigating the performance of different grouping strategies. The results show that clustering similar classes to construct an expert achieves comparable accuracy to superclass grouping and higher accuracy than random grouping.

To cluster similar classes within a task, we applied K-means clustering to the prototype of each class, varying the number of clusters, K, between two and three. This ensures that the classes are grouped into at least two and at most three clusters. The optimal number of clusters is determined based on the highest Silhouette score, a widely used metric that quantifies the wellness of clustering by measuring how similar each sample within its assigned cluster compared to neighbouring clusters [29].

2) EXPERT NETWORK CREATION

A small expert model (e.g., with one hidden layer of 128 neurons) is created for each cluster within a task, and each is trained independently (e.g., outside the MoE framework) using the training data from their corresponding classes. Once each expert has been trained independently, we freeze their weights, integrate them into a MoE framework, and train the gate and classification mapper (CM) layer, which projects the logits of each expert to a unified layer. The CM layer's dimension is $C_i * C_{all}$, where C_i is the number of classes known by an expert i, and C_{all} is the total number of classes seen so far. The gate learns the importance coefficients for the existing M experts and integrates their output into the final prediction:

$$\hat{y} = \sum_{i \in [1,M]} g_i \mathrm{CM}(z_i), \tag{1}$$

where z_i is the logit output from expert i, CM maps the logits to the unified classification layer, and g_i is the gate coefficient for that expert. The gate and CM layer are trained with a mixture of replay samples from the buffer and those generated by the GMM. With this two-step training strategy, each expert can still be used independently to classify what it has learnt by taking the output from its classification layer, while also being used in an ensemble with the other experts in the ODA architecture by taking the output from the CM layer.

C. EXPERT FUSION

To calculate the distance between parameters of experiments, we employ singular value decomposition (SVD) to decompose them first, as SVD has shown promising results in calculating the distance of high-dimensional vectors [30].

To merge experts, we need to address three questions: (1) when to merge, (2) which experts to merge, and (3) how to merge without degrading the performance. Our motivation for merging experts stems from limited memory capacity or the maximum number of experts that a device can host. When the current number of experts reaches the maximum, we merge the most similar experts. Similarity is calculated based on their hidden layer's weights. While the Jonker-Volgeneant algorithm [31] has been adopted to calculate the similarity between experts' weights [9], it is computationally expensive. Here, we first employ SVD to decompose the weights and calculate the distance between their singular values [30]. A small distance indicates that the singular values of both layers are close, suggesting they capture similar information or exhibit similar patterns. More specifically, for each layer l in expert i and expert j, SVD is performed to decompose their parameters (i.e., the weights and biases) into three main components: the left singular vectors ($\boldsymbol{U}_{i,l}, \boldsymbol{U}_{j,l}$), the singular values ($\boldsymbol{S}_{i,l}, \boldsymbol{S}_{j,l}$), and the right singular vectors ($\boldsymbol{V}_{i,l}, \boldsymbol{V}_{j,l}$):

$$\boldsymbol{W}_{i,l} = \boldsymbol{U}_{i,l}\boldsymbol{S}_{i,l}\boldsymbol{V}_{i,l} \tag{2}$$

$$\boldsymbol{W}_{j,l} = \boldsymbol{U}_{j,l}\boldsymbol{S}_{j,l}\boldsymbol{V}_{j,l} \tag{3}$$

Then the distance between two experts is calculated as the sum of Euclidean distance between corresponding singular values of their layers $\boldsymbol{S}_{i,l}$ and $\boldsymbol{S}_{j,l}$:

$$d_{i,j} = \Sigma_{l=1,2} |\boldsymbol{S}_{i,l} - \boldsymbol{S}_{j,l}| \tag{4}$$

After calculating the distance between each pair of experts and rank them in ascending order, we then select the top pairs for merging until the memory budget is met. If there is any overlap in the identified expert pairs, for example, if two pairs of experts (i, j) and (i, k) are in the top list, we merge the three experts i, j, k. Another key consideration is that expanding an expert with too many classes is not ideal, as the model may become saturated. Therefore, we set a maximum limit on the number of classes an expert can take and will not merge any expert whose number of classes already exceeds the threshold. In the future, we may also investigate checking neuron activation at each layer to determine whether an expert has become saturated.

Once we have identified the experts, we perform the merging by averaging the weights of their hidden layers. For the classification layer, we concatenate their weights as they correspond to different classes in each expert. We then align the norms of the weight vectors of classes from the two experts. Since each expert is trained independently with potentially different numbers of training samples, their weight magnitude can be different, potentially causing bias when we fine-tune the merged expert. To address this, we adopt the weight alignment technique [32] that balances the weight magnitudes between old and new classes in class-incremental learning. More specifically, let $\boldsymbol{W}_i$ and $\boldsymbol{W}_j$ refer to the weights of the classification layer corresponding to the

ith and jth experts; that is:

$$\boldsymbol{W}_i = (\boldsymbol{w}_{i,1}, \boldsymbol{w}_{i,2}, \ldots, \boldsymbol{w}_{i,C_i}), \tag{5}$$
$$\boldsymbol{W}_j = (\boldsymbol{w}_{j,1}, \boldsymbol{w}_{j,2}, \ldots, \boldsymbol{w}_{j,C_j}), \tag{6}$$

where C_i and C_j are the number of classes in the two experts. Their norms are defined as:

$$\text{Norm}_i = (||\boldsymbol{w}_{i,1}||, \ldots, ||\boldsymbol{w}_{i,C_i}||) \tag{7}$$
$$\text{Norm}_j = (||\boldsymbol{w}_{j,1}||, \ldots, ||\boldsymbol{w}_{j,C_j}||). \tag{8}$$

We concatenate their weights $W = (\boldsymbol{W}_i, \boldsymbol{W}_j)$, and normalise them as

$$\boldsymbol{W}' = (\boldsymbol{W}_i, \gamma \boldsymbol{W}_j) \tag{9}$$
$$\gamma = \frac{\text{Mean}(\text{Norm}_i)}{\text{Mean}(\text{Norm}_j)} \tag{10}$$

This ensures that the norms of both experts' weights are equal. In summary, given a collection of experts $1, 2, \ldots, m$, each with k layers, the merging function `merge` is defined as follows:

$$\text{merge}(E_1, E_2, \ldots, E_m) = \begin{cases} \text{average}(\boldsymbol{W}_{1,j}, \boldsymbol{W}_{2,j}, \ldots, \boldsymbol{W}_{m,j}) & \forall j \in [1, k-1] \\ \text{wa}(\text{concat}(\boldsymbol{W}_{1,j}, \boldsymbol{W}_{2,j}, \ldots, \boldsymbol{W}_{m,j})) & j = k \end{cases} \tag{11}$$

where $\boldsymbol{W}_{1,j}, \boldsymbol{W}_{2,j}, \ldots, \boldsymbol{W}_{m,j}$ represents the weights at jth layer of each expert, which will be averaged if j is not the last layer; i.e., the classification layer; otherwise, the weights will be concatenated first and then aligned via normalisation. `average` is a function to average the weights of each layer, `concat` is a function to concatenate the weights in the last layer from all the experts, and `wa` is the above weight alignment function.

Once merging the experts, we fine-tune the new expert with the data from all the merging experts, which consists of a mixture of replay samples from the buffer and generated samples from GMM. While it is preferable to use only generated samples, we find that incorporating a small number of real samples significantly reduces forgetting. After training, the merged expert will replace the merging experts and classify their classes. Algorithm 1 outlines the overall training process of ODA.

Algorithm 1 ODA Training Procedure

```
for each task t do
    generate prototype and GMM for each class in t
    run clustering on the prototypes
    create and train an expert for each cluster
    store in the buffer the replay samples for each class in t
    if the current number of experts exceeds the maximum
    number of experts then
        select the experts whose maximum number of classes
        below a pre-defined threshold
        compute the similarity between the weights of their
        hidden layers
        select and merge the top similar experts
        fine-tune the merged experts using the buffer and
        GMM samples
    end if
    train the gate and classification mapper layer
end for
```

IV. EXPERIMENT, RESULTS, AND DISCUSSION

This section describes our experiment methodology including datasets, evaluation metrics, and baseline selection, and presents and discusses the results.

A. EXPERIMENT METHODOLOGY

The experiments are conducted on three datasets that have been widely used in continual learning [33], [34]: CIFAR-100[1] [10], CUB[2] [11], and iNaturalist[3] [12].

[1]https://www.cs.toronto.edu/ kriz/cifar.html
[2]https://www.vision.caltech.edu/datasets/cub_200_2011/
[3]https://github.com/visipedia/inat_comp/tree/master/2018

The CIFAR-100 dataset consists of 60000 32×32 colour images, spanning 100 fine classes and 20 coarse classes. Each image is associated with two labels: fine and coarse. The fine label specifies the actual class to which the image belongs whereas the coarse label denotes its superclass. For each fine class, there are 500 training samples and 100 test samples. Thus, CIFAR-100 is a balanced dataset.

The CUB dataset consists of 200 birds species with about 30 samples per class, among which 5994 samples are for training and 5794 samples for testing. Like CIFAR-100, this dataset is balanced.

The iNaturalist dataset contains over 800000 images of more than 5000 different plants and animals species. The dataset includes several classification groups based on taxonomy: category, kingdom, family, supercategory, class, phylum, genus and order. In our experiment, we use the category group and select the top 1011 categories. This results in a long-tailed dataset where the number of samples per class ranges from 1000 to 80.

Metrics: Performance is measured using two mostly commonly used metrics in continual learning [33]: Final Average Accuracy (FAA) and Final Forgetting (FF). FAA measures the average accuracy of the model across all tasks at the end of the learning process, providing a comprehensive view of the model's performance after exposure to the full sequence of tasks.

$$\text{FAA} = \frac{1}{T} \sum_{i=1}^{T} A_{T,i} \tag{12}$$

where T is the total number of tasks, and $A_{T,i}$ is the accuracy on task i after learning all T tasks.

FF measures the degree of forgetting for each task by comparing the model's accuracy on that task immediately after learning it with its accuracy at the end of the learning process. This quantifies how much information is lost over

TABLE 1. Experiments for the three datasets. The numbers are in percentage (%) with the standard deviation in brackets. ODA outperforms the state-of-the-art continual learning techniques on all the datasets.

	CIFAR-100 10 tasks/100 classes		CUB 10 tasks/200 classes		iNaturalist 20 tasks/1011 classes	
Techniques	FAA	FF	FAA	FF	FAA	FF
Joint training	80.98 (0.20)		85.84 (0.17)		78.42 (0.04)	
Finetune	16.58 (1.32)	86.98 (1.48)	22.34 (1.54)	82.82 (1.85)	10.63 (0.41)	88.87 (0.64)
LwF	12.31 (0.94)	95.68 (1.21)	18.80 (0.85)	87.04 (0.71)	7.05 (0.36)	90.40 (0.23)
EWC	18.39 (0.95)	85.83 (0.80)	29.20 (1.80)	75.18 (1.64)	20.15 (1.39)	74.91 (1.43)
iCaRL	64.34 (0.38)	**9.26 (0.23)**	79.76 (0.37)	7.78 (0.40)	60.78 (0.39)	**10.44 (0.41)**
Gdumb	63.02 (0.38)		77.70 (1.34)		49.69 (0.22)	
BiC	73.64 (0.66)	12.46 (2.15)	82.64 (0.39)	**7.74 (0.75)**	68.28 (0.25)	11.62 (0.31)
DER++	71.01 (0.58)	25.15 (0.51)	81.58 (1.10)	14.14 (1.10)	58.63 (0.26)	33.79 (0.39)
ProgressNet	14.92 (0.51)	88.41 (1.21)	22.49 (1.00)	82.70 (0.89)	11.06 (0.35)	91.28 (0.56)
VE Average	16.00 (1.88)	78.23 (1.82)	21.58 (1.81)	75.49 (1.65)	13.78 (0.54)	77.75 (0.62)
VE Majority	15.52 (0.72)	78.61 (0.61)	21.72 (0.68)	75.37 (0.93)	14.36 (0.59)	77.47 (0.44)
VE Hard	14.97 (0.63)	78.90 (0.85)	19.67 (1.46)	76.73 (2.10)	13.86 (0.74)	77.82 (0.49)
ODA	**76.03 (0.94)**	26.53 (1.96)	**87.53 (0.87)**	43.47 (1.87)	**70.82 (4.73)**	47.69 (6.66)

time.

$$\text{FF} = \frac{1}{T-1}\sum_{i=1}^{T-1}(A_{i,i} - A_{T,i}) \tag{13}$$

where $A_{i,i}$ is the accuracy on task i right after training it, and $A_{T,i}$ is the accuracy on task i after learning all T tasks.

B. BASELINE SELECTION

To evaluate ODA, a set of commonly used baseline models is selected. This selection covers all three categories in continual learning: regularisation, replay, and dynamic architecture. They are LwF [5], EWC [13], iCaRL [6], Gdumb [35], bias correction (BiC) [36], dark experience replay (DER++) [37], ProgressNet [15] and Vanilla Ensemble [38]. Two additional baselines are also used to serve the upper bound (joint training) and the lower bound (fine tuning). Most of the implementations are sourced from the existing libraries Mammoth [37], FACIL [39] and Avalanche [40]. A home-made Vanilla Ensemble is constructed as a readily available implementation is not found.

C. IMPLEMENTATION AND HYPERPARAMETER SELECTION

We use Vision Transformer[4] [41] to extract features for the datasets.

For ODA, each expert consists of a hidden layer with 128 neurons and the gate has a hidden layer with 256 units, which is intentionally kept small. In practice, the maximum number of experts can be decided by the hardware memory constraints and the maximum number of classes in an expert can be decided by their model size. In our experiments, we set the maximum number of experts to be 6, 10 and 20 and the maximum number of classes in an expert to be 17, 20, and 50 for CIFAR-100, CUB and iNaturalist. This setup aims to encourage merging, and we have also evaluated ODA's performance with different configurations of these two parameters. For consistency, the same architecture is adopted for all the baseline models such that the total parameters of ODA will be less than or equal; that is, one hidden layer with 1000, 1500 and 2000 units for CIFAR-100, CUB and iNaturalist.

For rehearsal, the buffer size of 10, 10 and 20 per class is chosen for CIFAR-100, CUB and iNaturalist. ODA is updated using both replay samples and GMM-generated samples. We experimented with well-established regularisation techniques including knowledge distillation (KD) and its variants. However, they did not significantly improve the performance and required more memory. Therefore, we do not use any regularisation techniques in our current implementation.

We run grid search for hyperparameters of ODA; that is, the searching range for learning rates is [0.1, 0.01, 0.001, 0.0001] and for batch size is [32, 64, 128]. FAAs are reported in Table 2 on different combinations and we select the combination yielding the highest FAA, which are learning rate 0.001 and batch size 64. For the baseline models, we use the following hyperparameters: a learning rate of 0.01 and a batch size of 64 for Joint Training, Finetune, and ProgressNet; and a learning rate of 0.03 and a batch size of 64 for the rest. The specific hyperparameters for each technique are kept as provided by the framework.

TABLE 2. FAA of hyperparameters via grid search on learning rates and batch sizes.

batch size	learning rate			
	0.1	0.01	0.001	0.0001
32	74.20	75.66	75.06	75.26
64	74.04	75.63	**76.06**	75.58
128	73.93	73.26	75.25	74.69

D. OVERALL PERFORMANCE

The most commonly used class-incremental continual learning settings are considered, where an equal number of 10, 20, and 50 randomly selected classes are assigned to each task for CIFAR-100, CUB, and iNaturalist. Table 1 presents the overall performance across the three datasets in terms

[4]https://pytorch.org/vision/main/models/vision_transformer.html

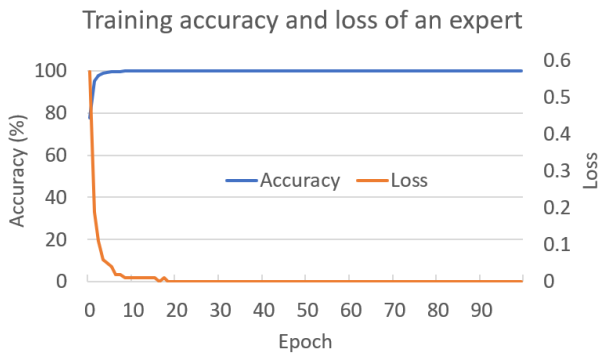

FIGURE 3. The performance progression of training one expert in 100 epochs on CIFAR-100. The stable accuracy and loss have been achieved around epoch 15.

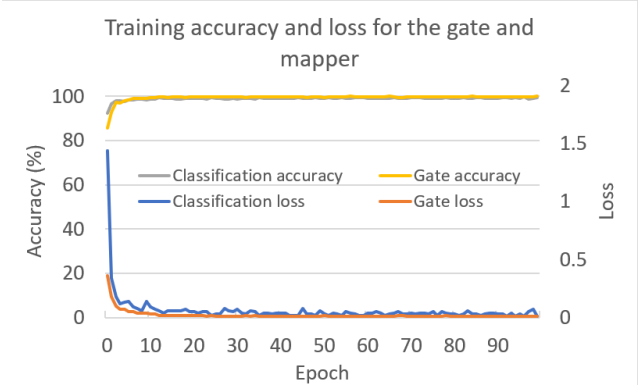

FIGURE 4. The performance progression of training the gate and the classification mapper in 100 epochs on CIFAR-100. The accuracy for both the gate and the classification mapper has been stable since around epoch 15 while their losses can still have small fluctuation.

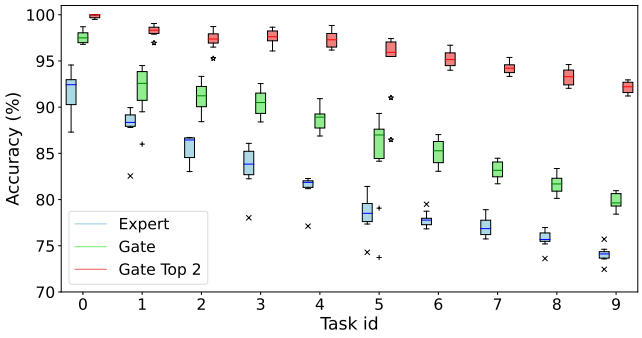

FIGURE 5. The performance progression of the experts, the gate and the gate's top-2 on CIFAR-100. The expert accuracy is taken by averaging all available experts at the end of training each task. The gate can maintain high accuracy over time with the newly introduced experts; however, the expert's accuracy with merging suffers large decrease.

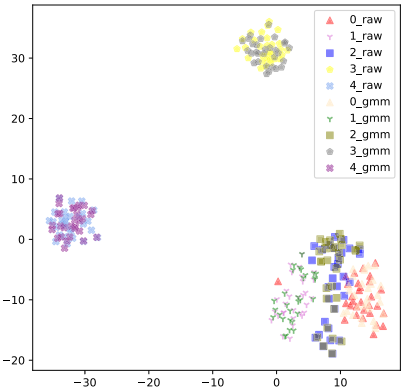

FIGURE 6. t-SNE of real images vs. GMM samples for iNaturalist. The distribution of GMM samples are very close to the original ones. The same shape represents the same class and the color separates real and generated samples.

of FAA and FF, with the mean score and standard deviation reported in brackets. ODA has outperformed the state-of-the-art techniques in all the datasets.

As discussed in Section IV-C, combining selected raw samples from the buffer with GMM-generated samples improves performance. A choice of 100, 20, and 100 was made for CIFAR-100, CUB, and iNaturalist, respectively. The selection of 20 for CUB is due to its limited 30 samples per class during training, ensuring that the GMM samples do not overpower the raw ones.

Across all the datasets, ODA demonstrates better accuracy, followed by BiC, iCaRL, and DER++. In terms of forgetting, iCaRL exhibits the least forgetting. Among the dynamic architecture approaches, ODA shows the least forgetting by a factor of half or a third compared to ProgressNet and VE techniques.

An example of the training progression for the expert and the gate is illustrated in Figure 3 and Figure 4. As each individual expert is trained on samples with close representations, optimal performance is achieved in the early epochs. Similarly, when the gate and the classification mapper layer are trained, only a small number of epochs are required to achieve very good accuracy and loss.

Figure 5 illustrates the performance progression of experts and gates on the CIFAR-100 dataset. The average validation accuracy of all experts and the gate at the end of each task is calculated from five experiments. The dashed line indicates the mean accuracy. The gate performs well, with its average top-2 accuracy reaching 90.42%, suggesting that the gate can accurately assign higher coefficients to the correct experts. This implies that in the future we may only need to activate the top-k experts for prediction to further reduce computational cost. The expert accuracy drops more than the gate's. Each expert is initially trained with real samples in the training data, but as they merge and are updated with only a small portion of the original samples alongside pseudo-samples from GMM, their performance degrades.

E. GMM PSEUDO-SAMPLES

Data augmentation plays a crucial role in enhancing the generalisation and performance of deep learning models [42]. In computer vision tasks, many techniques have been developed to deal with this challenge. These techniques range from directly manipulating the images, such as flipping, rotating, clipping, mixing, to using generative models to reproduce images with similar features.

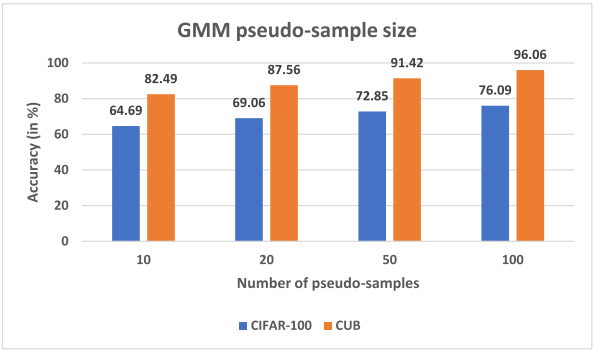

FIGURE 7. The effect of various pseudo-sample sizes generated by GMM on CIFAR-100 and CUB. The general trend is that the more generated samples, the better performance. But there is a need to consider the trade-off of training time and the limitation of memory.

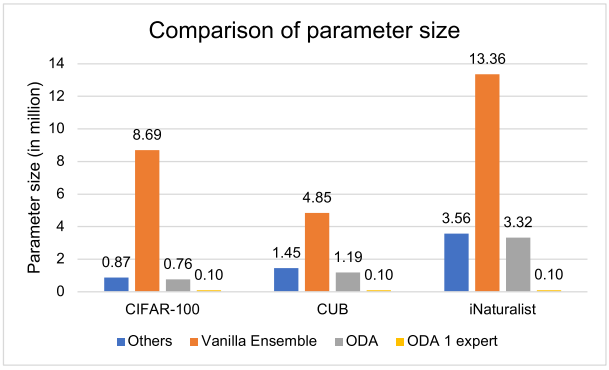

FIGURE 8. Parameter size comparison between ODA and the baselines. All baselines but Vanilla Ensemble are grouped as 'Others'. The parameter size is in millions. The average size of one expert is also provided for comparison. ODA uses much fewer parameters than the other continual learning techniques.

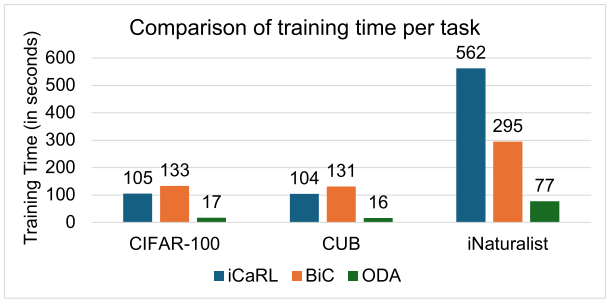

FIGURE 9. Comparison of training time between ODA and best-performing techniques: iCaRL and BiC. ODA only consumes around 13% of training time of these techniques when updating with a new task.

Similarly, GMM can augment datasets by generating pseudo-samples that capture the characteristics of the data within the same class. In our study, these pseudo-samples improve ODA performance. For example, on CUB, the accuracy of ODA even slightly exceeds that of joint training. This is likely due to the small size of the original training samples and the closely matched pseudo-samples generated by GMM, which enhances the effectiveness of the sample distribution captured by GMM during retraining steps after merging. Figure 6 shows the t-SNE distribution between real samples and GMM-generated samples for five classes in CUB, showing that GMM samples closely resemble real samples.

The effect of various pseudo-sample sizes on CIFAR-100 and CUB is illustrated in Figure 7. In general, having more pseudo-samples generated by GMM helps improve the accuracy. However, the decision to choose the right size also needs to consider the trade-off between training time and the limitation of the available memory.

F. PARAMETER SIZE

Following the decision in Section IV-C to keep the architecture to be small, we compared the parameter size between ODA and the baselines as shown in Figure 8. The plot presents a comparison of parameter sizes across different models for CIFAR-100, CUB, and iNaturalist datasets. On average, ODA maintains a comparable parameter size to the 'Others' baseline, showing that our model achieves competitive resource usage without excessive growth in parameters. Importantly, despite having a similar parameter footprint to Others, ODA is able to deliver better performance.

Additionally, the plot includes the average parameter size of a single expert in ODA, which remains minimal at around 100K parameters across datasets. This compact expert design not only aids scalability but also allows ODA to efficiently utilise memory and encourage expert merging where appropriate. Overall, ODA provides a balanced and effective solution by controlling parameter size while achieving strong performance outcomes.

G. COMPUTATIONAL COST

We ran all experiments on NVIDIA RTX A6000 with 48GB of GDDR6 GPU memory. Figure 9 compares the training time per task with the best-performing techniques across three datasets. ODA takes 12.78%, 12.22% and 26.10% of BiC's training time, and 16.19%, 15.38%, and 13.70% of iCaRL's on CIFAR-100, CUB, and iNaturalist, respectively. Since iNaturalist has the most number of classes (i.e., 1011), and more classes to learn per task (i.e., 50), the training time is significantly longer than the other two datasets. Figure 10, Figure 11 and Figure 12 show the progression of training time on each task on CIFAR-100, CUB, and iNaturalist. With more classes to learn, iCaRL's training increases much faster than ODA's and BiC's. ODA's training time stays low and does not vary much, as it separates the training of experts and gate. Table 3 lists the training time for each key component in ODA. The gate training and SVD for calculating the similarity between the weights of experts are consuming more time. In ODA, the number of parameters grows gradually with more experts being created, ultimately reaching 88% of the baseline models.

H. IMPACT OF THE NUMBER OF EXPERTS

The impact of the number of experts on performance is also investigated, specifically how accuracy declines as the number of experts is reduced. This analysis aims to

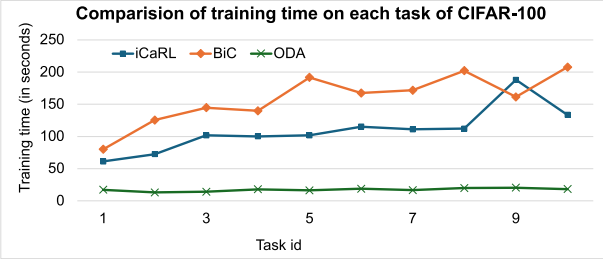

FIGURE 10. Comparison of training time on each task of CIFAR-100 between ODA and best-performing techniques: iCaRL and BiC. The comparison techniques increase training time with more tasks, while ODA stays small because ODA freezes most of the experts and only needs to update one expert and the gate, which reduces the training cost.

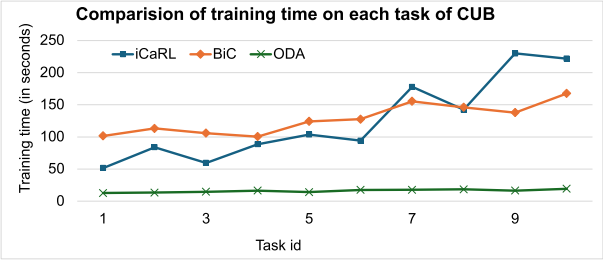

FIGURE 11. Comparison of training time on each task of CUB between ODA and best-performing techniques: iCaRL and BiC.

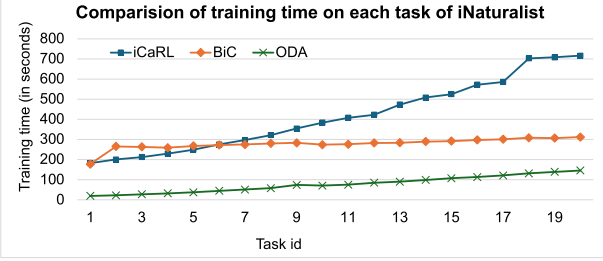

FIGURE 12. Comparison of training time on each task of iNaturalist between ODA and best-performing techniques: iCaRL and BiC.

TABLE 3. Training time per key component in ODA (in seconds). The most computationally expensive component is the gate training, and the expert training and updating has similar profile.

ODA components	CIFAR-100	CUB	iNaturalist
GMM training per class	0.84	0.74	0.93
Clustering for each task	0.03	0.04	0.17
Similarity checking	4.76	10.09	33.35
Expert training	2.95	2.46	3.53
Expert merging and training	3.85	3.48	6.20
Gate training	9.59	9.16	67.42

understand the effect of the merging process on long-term performance. Table 4 illustrates the accuracy and parameter size (in millions) on CUB. Generally, fewer experts result in lower accuracy, fewer parameters, and faster training. This is expected as the increased merging process leads to more frequent retraining and worse forgetting in experts. This highlights the need for careful management of the merging process to avoid compromising the overall performance of the architecture.

TABLE 4. Accuracy, the number of parameters (in millions), and training time per task (in seconds) on a different number of experts.

No. of experts	Accuracy	No. Params	Training time per task
10	87.53	1.26	17.40
8	84.21	1.01	13.34
6	84.28	0.84	11.42
4	82.04	0.73	10.62
2	80.52	0.56	9.83

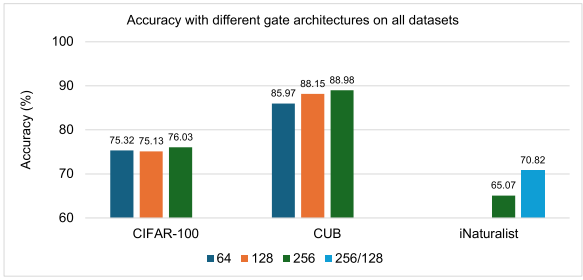

FIGURE 13. Comparison of accuracies with different gate architectures on all three datasets. When the dataset has more classes and tasks, the gate needs a larger architecture to manage the growing number of experts.

I. IMPACT OF GATE ARCHITECTURE

Figure 13 compares accuracies with different gate architectures on all three datasets. On CIFAR-100 and CUB, the best performing gate is one hidden layer with 256 neurons. On CIFAR-100, increasing the size of the gate does not significantly improve the accuracy, as the number of the experts is quite small. On iNaturalist, we have much more experts, we start with the large architecture and then increase the depth of the gate; that is, adding an extra hidden layer with 128 neurons. The accuracy improves greatly. A general rule can be drawn from the experiment is that the more classes to learn and the more experts to be created, the larger the gate needs to be.

V. CONCLUSION AND FUTURE WORK

This paper presents ODA for class-incremental learning. It is built on MoE, enabling to extend the model with more experts and shrink via merging experts. The advantage is that ODA does not need to commit to a large model at the beginning, optimising memory use and reducing training time. The experiment results demonstrate that ODA can achieve comparable accuracy, i.e., 2.39%, 6.34%, and 2.54% over the best-performing techniques on CIFAR-100, CUB, and iNaturalist, with much less training time. In terms of parameter size, ODA also uses a slightly smaller parameter compared to the baselines, which makes it a suitable candidate to maintain good performance where memory is limited. This can be promising to deploy it on resource-constrained devices for real-world continual learning applications.

A. LIMITATION DISCUSSION

One major limitation of ODA is the significant degradation in expert after merging. Future work will explore methods to mitigate the forgetting effect on experts. Another limitation is the computational cost associated with dynamically

managing experts, including the calculation of the similarity of weights between experts. Furthermore, the scalability of ODA to very large datasets and highly dynamic environments has not been extensively tested, potentially limiting its applicability to more complex scenarios. Future work will explore addressing these limitations by enhancing the merging mechanism, introducing advanced federated learning techniques, and optimising the system for large-scale, real-world deployments.

Additionally, ODA currently assumes that task boundaries are clearly defined and that training data for new tasks is fully annotated, which may not align with the realities of real-world applications where task boundaries are ambiguous, and labels are scarce.

REFERENCES

[1] F. Amrollahi, S. P. Shashikumar, A. L. Holder, and S. Nemati, "Leveraging clinical data across healthcare institutions for continual learning of predictive risk models," *Sci. Rep.*, vol. 12, no. 1, p. 8380, May 2022.

[2] J. Li et al., "Towards precision medicine based on a continuous deep learning optimization and ensemble approach," *npj Digit. Med.*, vol. 6, no. 1, p. 18, Feb. 2023.

[3] E. Verwimp, K. Yang, S. Parisot, H. Lanqing, S. McDonagh, E. Pérez-Pellitero, M. De Lange, and T. Tuytelaars, "CLAD: A realistic continual learning benchmark for autonomous driving," 2022, *arXiv:2210.03482*.

[4] H. Zhang and F. Mueller, "CLAIRE: Enabling continual learning for real-time autonomous driving with a dual-head architecture," in *Proc. IEEE 25th Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2022, pp. 1–10.

[5] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

[6] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "ICaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5533–5542.

[7] D.-W. Zhou, Q. Wang, Z. Qi, H.-J. Ye, D. Zhan, and Z. Liu, "Class-incremental learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 9851–9873, Jul. 2024.

[8] M. Artetxe et al., "Efficient large scale language modeling with mixtures of experts," in *Proc. Conf. Empirical Methods Natural Language Process.*, 2022, pp. 11699–11732.

[9] K. Man Lo, Z. Huang, Z. Qiu, Z. Wang, and J. Fu, "A closer look into mixture-of-experts in large language models," 2024, *arXiv:2406.18219*.

[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.

[11] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "Caltech-ucsd birds-200–2011," California Inst. Technol., Pasadena, CA, USA, Tech. CNS-TR-2011-001, 2011. [Online]. Available: https://authors.library.caltech.edu/records/cvm3y-5hh21

[12] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The iNaturalist species classification and detection dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8769–8778.

[13] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," 2016, *arXiv:1612.00796*.

[14] Y. Yang, H. Yuan, X. Li, Z. Lin, P. Torr, and D. Tao, "Neural collapse inspired feature-classifier alignment for few-shot class-incremental learning," in *Proc. ICLR*, 2023, pp. 1–18.

[15] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*.

[16] J. Schwarz, W. M. Czarnecki, J. Luketina, A. Grabska-Barwińska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *Proc. ICML*, J. G. Dy and A. Krause, Eds., Jul. 2018, pp. 4528–4537. [Online]. Available: http://proceedings.mlr.press/v80/schwarz18a.html

[17] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7120–7129.

[18] S. Yan, J. Xie, and X. He, "DER: Dynamically expandable representation for class incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Los Alamitos, CA, USA, Jun. 2021, pp. 3013–3022, doi: 10.1109/CVPR46437.2021.00303.

[19] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *Proc. ICLR*, 2018, pp. 1–11.

[20] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7765–7773.

[21] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proc. Eur. Conf. Comput. Vis.*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham, Switzerland: Springer, Jan. 2018, pp. 72–88.

[22] J. Yoon, S. Kim, E. Yang, and S. J. Hwang, "Scalable and order-robust continual learning with additive parameter decomposition," in *Proc. ICLR*, Jan. 2019, pp. 1–15.

[23] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5362–5383, Aug. 2024.

[24] T. Li, Q. Ke, H. Rahmani, R. E. Ho, H. Ding, and J. Liu, "Else-net: Elastic semantic network for continual action recognition from skeleton data," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 13414–13423.

[25] D.-W. Zhou, Q.-W. Wang, H.-J. Ye, and D.-C. Zhan, "A model or 603 exemplars: Towards memory-efficient class-incremental learning," 2022, *arXiv:2205.13218*.

[26] A. Douillard, A. Ramé, G. Couairon, and M. Cord, "DyTox: Transformers for continual learning with dynamic token expansion," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 9275–9285.

[27] J. Xie, S. Yan, and X. He, "General incremental learning with domain-aware categorical representations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 14331–14340.

[28] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017, *arXiv:1706.05098*.

[29] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, Nov. 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257

[30] M. Duan, D. Liu, X. Ji, R. Liu, L. Liang, X. Chen, and Y. Tan, "FedGroup: Efficient federated learning via decomposed similarity-based clustering," in *Proc. IEEE Intl Conf Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, Los Alamitos, CA, USA, Oct. 2021, pp. 228–237, doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00042.

[31] R. Jonker and T. Volgenant, "Technical note—An improved transformation of the symmetric multiple traveling salesman problem," *Oper. Res.*, vol. 36, no. 1, pp. 163–167, Feb. 1988.

[32] B. Zhao, X. Xiao, G. Gan, B. Zhang, and S.-T. Xia, "Maintaining discrimination and fairness in class incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 13205–13214.

[33] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022, doi: 10.1109/TPAMI.2021.3057446.

[34] H. Qu, H. Rahmani, L. Xu, B. Williams, and J. Liu, "Recent advances of continual learning in computer vision: An overview," 2021, *arXiv:2109.11369*.

[35] A. Prabhu, P. H. S. Torr, and P. K. Dokania, "GDumb: A simple approach that questions our progress in continual learning," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, Jan. 2020, pp. 524–540, doi: 10.1007/978-3-030-58536-5_31.

[36] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 374–382.

[37] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, "Dark experience for general continual learning: A strong, simple baseline," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA, Jan. 2020, pp. 1–14.

[38] T. Doan, S. Iman Mirzadeh, and M. Farajtabar, "Continual learning beyond a single model," 2022, *arXiv:2202.09826*.

[39] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer, "Class-incremental learning: Survey and performance evaluation on image classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 5513–5533, May 2023.
[40] V. Lomonaco et al., "Avalanche: An end-to-end library for continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 3595–3605.
[41] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.
[42] C. Ozdemir, Y. Dogan, and Y. Kaya, "RGB-angle-wheel: A new data augmentation method for deep learning models," *Knowl.-Based Syst.*, vol. 291, May 2024, Art. no. 111615. [Online]. Available: https: //www.sciencedirect.com/science/article/pii/S0950705124002508

FAHRURROZI RAHMAN received the Ph.D. degree in computer science from the University of St Andrews, in 2021. He is currently a Research Fellow with the School of Computer Science, University of St Andrews. His research interests include natural language processing in the clinical domain, and computational creativity, particularly in areas, such as poetry and musical harmony generation.

ANDREA ROSALES SANABRIA is currently a Research Fellow with the School of Computer Science, University of St Andrews. She specialized in generative adversarial networks and contrastive learning. Her research interests include centers around unsupervised domain adaptation in sensor-based human activity recognition.

JUAN YE received the Ph.D. degree in computer science from University College Dublin. She is currently a Reader with the School of Computer Science, University of St Andrews. Her research interests include center around adaptive pervasive systems, specializing in sensor-based human activity recognition, continual learning, sensor fusion, context awareness, and domain adaptation.