SynapNet: A Complementary Learning System Inspired Algorithm With Real-Time Application in Multimodal Perception

Nilay Kushawaha, Lorenzo Fruzzetti, Enrico Donato, *Member, IEEE*, and Egidio Falotico, *Member, IEEE*

***Abstract*— Catastrophic forgetting is a phenomenon in which a neural network, upon learning a new task, struggles to maintain its performance on previously learned tasks. It is a common challenge in the realm of continual learning (CL) through neural networks. The mammalian brain addresses catastrophic forgetting by consolidating memories in different parts of the brain, involving the hippocampus and the neocortex. Taking inspiration from this brain strategy, we present a CL framework that combines a plastic model simulating the fast learning capabilities of the hippocampus and a stable model representing the slow consolidation nature of the neocortex. To supplement this, we introduce a variational autoencoder (VAE)-based pseudo memory for rehearsal purposes. In addition by applying lateral inhibition masks on the gradients of the convolutional layer, we aim at damping the activity of adjacent neurons and introduce a sleep phase to reorganize the learned representations. Empirical evaluation demonstrates the positive impact of such additions on the performance of our proposed framework; we evaluate the proposed model on several class-incremental and domain-incremental datasets and compare it with the standard benchmark algorithms, showing significant improvements. With the aim to showcase practical applicability, we implement the algorithm in a physical environment for object classification using a soft pneumatic gripper. The algorithm learns new classes incrementally in real time and also exhibits significant backward knowledge transfer (KT).**

***Index Terms*— Catastrophic forgetting, complementary learning system (CLS), continual learning (CL), perception, pseudo episodic memory, soft gripper, variational autoencoder (VAE).**

I. Introduction

HUMANS and other animals excel at learning in a lifelong manner from an ever-changing environment that generates nonstationary streams of data, but it remains an open challenge for deep learning algorithms. The ability of an agent to continually learn over time by accommodating new knowledge while consolidating and retaining the previously acquired experiences is referred to as continual learning (CL) or lifelong learning [1]. Such ability is a crucial step in our quest to develop truly efficient models. Autonomous agents [2] learn in an open-ended and continual manner; crucial components of such a developmental approach consist of learning the ability to explore the environment and add new tasks while preserving the old ones. However, a major impediment for these computational models is their tendency to lose previously acquired capabilities when trained on samples from a new task or distribution, a phenomenon known as catastrophic forgetting. Catastrophic forgetting often results in a sharp decline in performance due to the old knowledge being overwritten by the new one or a change in data distribution [3].

A naive way to combat catastrophic forgetting would be to save all data, shuffle it, and come back to a traditional machine learning setting. Although retraining from scratch pragmatically addresses the issue of catastrophic forgetting, it is very inefficient and hinders the learning of novel data in real time. For instance, there may not be a difference between the training and testing phases for an autonomous agent that learns by actively interacting with its environment. In classical machine learning, an algorithm has access to all training data at the same time, whereas, in CL, the data instead arrive in a sequence of experiences, and the underlying distribution of the data changes over time.

To mitigate the issue of catastrophic forgetting, the learning system needs to accomplish two critical tasks. First, it must be capable of gaining new knowledge and enhancing what it already knows through ongoing input. Second, it has to ensure that the introduction of new information does not disrupt existing knowledge. This leads to the necessity of a fine balance, where the system needs to be plastic enough to incorporate new knowledge (plasticity) without causing disruption of existing knowledge (stability). This balancing act is known as the stability-plasticity dilemma [4]. In the brain, the ability to continually acquire, consolidate, and transfer knowledge is mediated by a rich set of neurophysiological principles [5]. Too much plasticity will result in previously encoded data being constantly forgotten, whereas too much stability will obstruct the network from learning new representations. Therefore, finding the optimal point between plasticity and stability is essential for effective learning and knowledge retention.

At the computational level, supervised CL can be classified into three primary types or "scenarios" [6]. The first scenario, task-incremental learning, requires an algorithm to learn a set of distinct tasks progressively. The second scenario, domain-incremental learning, involves learning the same type of task in diverse environmental conditions or domains. In the third

Manuscript received 20 October 2023; revised 19 February 2024 and 7 June 2024; accepted 13 August 2024. Date of publication 27 August 2024; date of current version 9 July 2025. This work was supported by Italian National Recovery and Resilience Plan (NRRP), M4C2, funded by European Union–NextGenerationEU (Project IR0000011, "EBRAINS-Italy") under Grant CUP B51E22000150006. *(Corresponding author: Nilay Kushawaha.)*

The authors are with the Department of Excellence in Robotics and AI, Scuola Superiore Sant'Anna, The BioRobotics Institute, 56025 Pisa, Italy (e-mail: nilay.kushawaha@santannapisa.it; lorenzo.fruzzetti@santannapisa.it; enrico.donato@santannapisa.it; egidio.falotico@santannapisa.it).

Digital Object Identifier 10.1109/TNNLS.2024.3446171

scenario, class-incremental learning, an algorithm must learn to differentiate between a growing number of classes incrementally. However, such scenarios are not mutually exclusive, and a CL application may involve a combination of them, such as learning new tasks in different domains.

Motivated by the work of Arani et al. [7] on complementary learning system (CLS) theory [8], we advance the work by combining a plastic model (fast learner, signifying hippocampus in the brain) and a stable model (slow consolidator, signifying neocortex in the brain) with a variational autoencoder (VAE)-based pseudo buffer [9], which serves as pseudo episodic memory. During training, the network applies lateral inhibition masks on the gradients of convolution layers to suppress the activity of neighboring neurons and improve the selectivity, as well as the discriminability of the learned features. Finally, the network proceeds to the sleep phase where the stable model is trained on pseudo episodic memory for a small duration to reorganize the learned representations. By reorganizing and refining its internal representations, the network can improve its ability to generalize and perform well on a variety of tasks. Notably, our network also exhibits a dynamic nature; when encountering new classes, it dynamically extends the output layer by adding new neurons. Section VI provides a detailed explanation of this dynamic adaptation mechanism. The main contributions of this article are summarized as follows.

1) We present a fast learner and a slow consolidator network with a VAE-based pseudo episodic memory for rehearsal.
2) We incorporate a lateral-inhibition mechanism and a sleep phase to enhance the learning process of the CL algorithm.
3) We evaluated the SynapNet algorithm on various CL datasets and compared the results with standard benchmark algorithms.
4) To demonstrate the practical applications of the proposed CL algorithm, we tested it in a real-time dynamic environment characterized by an unknown number of classes, specifically on a soft pneumatic gripper [10] equipped with two force sensors and two flex sensors for object classification.

This article is structured as follows. In Section II, an overview of the related works in CL is presented, with a focus on all four strategies employed in the literature. Section III provides a detailed description of the proposed methodology that has been adopted in the algorithm used in this study. Following this, the experimental setup is presented in Section IV, which includes the model architecture, evaluation criteria, and the baseline algorithms used for comparison. The evaluation of the proposed methodology on five class-incremental datasets, namely, MNIST [11], FMNIST [12], CIFAR10 [13], CIFAR100 [13], and ImageNet100 [14], and two-domain incremental datasets, namely, Permuted MNIST (PMNIST) [15] and Rotated MNIST (RMNIST) [16], is provided in Section V. Moreover, in Section VI, this article showcases the real-world dynamic application of the SynapNet algorithm on a soft pneumatic gripper. Finally, a summary noting the advantages and disadvantages of the proposed method along with the scope for improvements and possible developments in the future is stated in Section VII.

II. Related Work

While each category of CL algorithms designed to address the problem of catastrophic forgetting is being populated with an increasing number of novel strategies, there is quite a room for yet-to-be-explored techniques, especially at the intersection of the four categories, namely, architectural strategies, regularization strategies, rehearsal strategies, and generative replay. It is also important to note that the strategies are not confined to these categories, and there can be a mixture of the strategies.

A. Architectural Strategies

The performance of the CL algorithm greatly depends on the architecture of the learning model. One approach is to modify dynamically the architecture of a model to make it learn new concepts or skills without interfering with the old ones. The simplest form of architectural regularization is freezing certain weights in the network so that it stays exactly the same [17]. In progressive neural networks (PNNs) [18], the network is designed as a sequence of interconnected modules, each of which is responsible for learning a specific task. During training, the modules are added incrementally as new tasks are introduced. The output of the previous module is passed as input to the next module, allowing the network to retain the knowledge of previous tasks while learning new ones. PathNet [19] uses a hierarchical neural architecture that can reuse modules for new tasks while preserving the old ones.

B. Regularization Strategies

Regularization approaches alleviate catastrophic forgetting by imposing constraints on the update of the neural weights. Simple regularization techniques such as sparsification [20], dropout [21], and early stopping [22] can reduce the likelihood of forgetting by limiting the extent to which weights are modified during training. Elastic weight consolidation (EWC), introduced by [15], uses the diagonal of the Fisher information matrix (FIM) to estimate the importance of the network parameters for the previous tasks; it constrains the parameters to remain close to the previous values. EWC balances the tradeoff between learning new tasks and retaining knowledge of the previous tasks by adjusting the strength of the regularization term [23]. Synaptic intelligence (SI) [24] extended the method in an online learning fashion to relax the constraint. The authors argued that the computation of Fisher information is expensive for continuous learning and proposed to calculate weight importance online during SGD. Li and Hoiem [25] proposed the learning without forgetting (LWF) approach composed of convolutional neural networks (CNN) in which the network with predictions of the previously learned tasks is enforced to be similar to the network with the current tasks by using knowledge distillation [26]. The distillation term in the loss function encourages the network to transfer the learned knowledge from previous tasks to new ones.

C. Rehearsal and Pseudo Rehearsal Strategies

The most simple yet effective approach is storing a subset of previously learned data in a memory buffer and replaying it

during training on new tasks. The chosen samples are selected to be representative of past tasks, allowing the network to adapt its weights and strengthen its connections for memories already learned. Unlike other approaches, rehearsal-based methods do not force the network to retain previous weights. Experience replay (ER) proposed by Lin and Mitchell [27] involves training the algorithm on a mixture of old experiences from the replay buffer and the current experience. By doing so, ER exposes the model to past experiences and prevents it from forgetting. The gradient episodic memory (GEM) [16] algorithm stores the gradients of the model on past tasks in a memory buffer and uses them to constrain the model's updates while training on new tasks. ICARL [28] includes an external fixed memory to store a subset of old task data based on an elaborated sample selection procedure and then employs a distillation step acting as a regularization.

An alternative to rehearsal-based approaches is generative replay (explicit generative replay) or pseudo rehearsal, which involves training generative models to capture the data distribution of past experiences. Synthetic data can be generated using techniques such as generative adversarial networks (GANs) [29], VAE [9], or simple data augmentation techniques. These approaches can be seen as model-based learning, where the model learns the data distribution of past experiences by regenerating samples or internal states. Shin et al. [30] combined deep generative models with ER to augment the training set by generating synthetic data for past experiences, allowing it to revisit and consolidate the previous knowledge over time.

Another variation presented by Van de Ven et al. [31] introduces a brain-inspired generative replay, where internal or hidden representations are replayed. These representations are generated by the network's own context-modulated feedback connections. However, a significant drawback of this method is the "photocopy problem," where the quality of synthetic images from previous classes deteriorates with the incremental addition of new classes. Khan et al. [32], [33] improve upon this approach by incorporating distillation in the latent space between the current and previous models to reduce feature drift. In addition, they also implement latent matching, where the decoder's output is sent back to the encoder, and the loss is calculated between the original and pseudo latent representations. However, a drawback of this approach is the necessity to store the previous models in memory to perform latent matching. Chenshen et al. [34] utilized conditional GANs (CGANs) to generate synthetic images from past tasks, either by storing the GAN after every task or by regularizing the GAN loss with EWC to prevent forgetting. Similarly, Ostapenko et al. [35] employed a CGAN that features neural plasticity through efficient learning of sparse attention masks for the network weights. However, a major challenge with GAN-based image generators is the prolonged training time and heightened computational costs.

D. Mixed Strategies

DualNet [36] comprises a fast learning system for supervised learning of task-specific patterns and a slow learning system for unsupervised representation learning of task-agnostic general representation via a self-supervised learning (SSL) technique. In addition, it utilizes a replay buffer to retain data from prior tasks. FearNet [37], on the other hand, employs a combination of architectural and pseudo rehearsal strategies; it uses a hippocampal network capable of quickly recalling new examples, a prefrontal cortex (PFC) based on generative neural network for storing long-term memories, and a third network to determine whether the system should use the PFC or hippocampal network for a particular example. FearNet also relies on a separate pretrained network, held fixed, to extract feature embeddings that are then input into the model. Unlike FearNet, our approach does not require a separate feature extractor. Moreover, SynapNet involves training the VAE network in an online setting to generate synthetic images for the pseudo episodic memory. We argue that pretraining and freezing the feature extractor network may undermine the CL objective, as the network needs to consistently explore new classes, potentially leading to nongeneralizable embeddings from the pretrained network at some stage.

CLS-ER proposed by Arani et al. [7] uses a triple-model static architecture composed of a fast learner, a slow consolidator, and a temporary model to facilitate incremental learning. It also uses a replay buffer of fixed size for rehearsal of the previous knowledge. Our approach is inspired by CLS-ER but with a novel adaptation: instead of retaining actual samples, which may pose privacy concerns, we utilize a VAE to generate synthetic images for the pseudo episodic memory. Unlike CLS-ER that treats all the feature gradients equally, our method incorporates a lateral inhibition mechanism. This mechanism involves suppressing the gradients of convolution layers with lower activation values relative to their neighbors, thereby introducing a form of competition among features. In addition, a significant distinction lies in the fact that both CLS-ER and FearNet operate as a static algorithm, necessitating prior knowledge of the number of classes—an aspect that contradicts the CL objective. In contrast, our network demonstrates dynamic expansion capabilities upon encountering an unknown class, as elaborated further on in Section VI.

Replay-free CL methods (nonexemplar methods) have gained attention due to their ability to address data privacy and storage challenges associated with exemplar-based approaches. Zhu et al. [38] proposed the PASS framework that leverages prototype augmentation in the deep feature space to maintain decision boundaries of previous tasks. In addition, PASS incorporates semisupervised learning (SSL) to strengthen the acquisition of more generalizable and transferable features. Always-be dreaming (ABD) proposed by Smith et al. [39] utilizes the concept of model-inversion image synthesis to generate images for the old tasks and employs a novel importance-weighted feature distillation strategy. FeTrIL [40] combines a frozen feature extractor and a pseudo feature generator for incremental learning. Both FeTrIL and FeCAM [41] store the centroids or the covariance matrices for every class. However, a major limitation of these approaches is the requirement to store either the feature extractor or the generator for every task, which could pose memory challenges.

Algorithm 1 SynapNet Algorithm

Input: Data stream D, learning rate η, Consistency weight λ, Normal distribution $\mathcal{U}$, Update rate r_P and r_S, Decay parameters α_P and α_S

Models: Working model $f_W(x;\theta_W)$, Stable model $f_S(x;\theta_S)$, Plastic model $f_P(x;\theta_P)$

Pseudo Memory $(PM) \leftarrow \{\}$

while Training **do**

$(X_r, Y_r) \sim D$ and $(X_g, Y_g) \sim PM$

$(X, Y) = (X_r, Y_r), (X_g, Y_g)$

$Z_P, Z_S \leftarrow f_P(X_g), f_S(X_g)$

$Y_g \leftarrow \text{OnehotEncoding}(Y_g)$ ▷ One hot encode for logit selection

$Z \leftarrow Z_P$ **if** $\sigma(Z_P)[Y_g] > \sigma(Z_S)[Y_g]$ **else** Z_S

$L = L_{CE}(\sigma(f_W(X)), Y) + \lambda L_{mse}(f_W(X_g), Z)$ ▷ Update working model

$\theta_W \leftarrow \theta_W - \eta \nabla_{\theta_W} L$

$\theta_W \leftarrow (\theta_W)_{masked}$ **if** inhibition is true **else** θ_W

$a, b \sim \mathcal{U}(0,1)$ ▷ Update stable and plastic model

$\theta_P \leftarrow \alpha_P \theta_P + (1-\alpha_P)\theta_W$ **if** $a < r_P$ **else** θ_P

$\theta_S \leftarrow \alpha_S \theta_S + (1-\alpha_S)\theta_W$ **if** $b < r_S$ **else** θ_S

$X_g \leftarrow VAE(X_r)$ ▷ Generate synthetic image for PM

$PM \leftarrow Concat(PM, (X_g, Y_g))$ ▷ Update pseudo episodic memory

end while

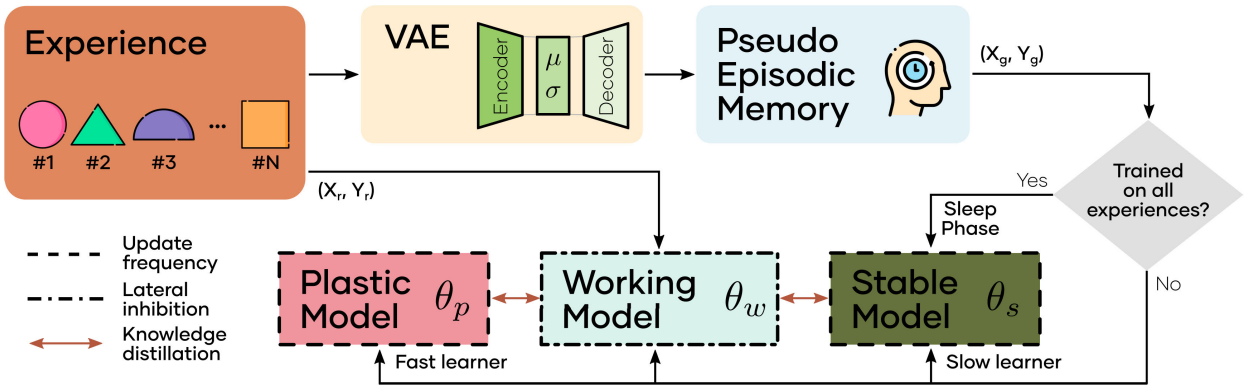


Fig. 1. SynapNet algorithm utilizes a triple model architecture, in which the working model acquires knowledge from a continuous data stream. The stable and plastic models interact with the working model to build short- and long-term memories of the learned representations. During the training phase of the working model, an inhibition mechanism is incorporated to enhance its learning process. Subsequently, after training on all experiences, the stable model undergoes a sleep phase to reorganize the acquired knowledge.

III. Methodology

In this section, we describe the proposed SynapNet algorithm that consists of a fast learner and a slow consolidator. These are supplemented by a VAE-based pseudo episodic memory, which stores and replays past experiences. In addition, the network employs lateral inhibition and enters a "sleep" phase, both of which serve to reduce the activity of adjacent neurons and reorganize the learned representations.

A. Complementary Learning System

The CLS theory [8] posits that effective learning in humans relies on two complementary systems: the neocortex, which gradually acquires structured knowledge about the environment, and the hippocampus, which enables rapid learning of specific items and experiences.

Building upon the research conducted by Arani et al. [7] on the CLS theory, we have adopted a similar triple model architecture for our study. The network consists of three models: a plastic model, a stable model, and a working model as depicted in Fig. 1. The plastic model emulates the neocortex, gradually acquiring structured knowledge about the environment. The stable model represents the hippocampus, facilitating rapid learning of specific information. The working model serves as a temporary connection between the two. The working model learns representations from the incoming data stream by accumulating knowledge in its weight. The acquired knowledge is distilled to the stable and the plastic model using the Mean Teacher [42] approach, which employs two exponential moving averages (EMAs) weighted over the working model's weight with decay parameters α_P and α_S (see Algorithm 1 for more details).

However, there are differences in the update frequencies and window sizes for the stable and plastic models. The stable model is updated less frequently with a larger window size. This design choice ensures that the stable model retains more information from earlier tasks, facilitating the consolidation and preservation of knowledge over time. On the other hand, the plastic model is updated more frequently with a smaller window size enabling it to adapt faster to information from new tasks.

B. VAE-Based Pseudo Episodic Memory

The main issue in CL is the consolidation of the previously learned knowledge after training on new information. In this study, we used a symmetric-VAE-based pseudo episodic memory [9] to deal with this problem. The stable and plastic models interact with the pseudo episodic memory to extract the consolidated activations for the memory samples, which are then utilized to constrain the update of the working model so that the acquisition of new knowledge does not significantly alter the performance on previously learned tasks. The VAE architecture consists of an encoder network q_ϕ that maps the input vector x into latent space, where the data are represented by a set of stochastic latent variable z (parameterized by mean μ^x and standard deviation σ^x). The encoder network learns to generate a distribution of latent variables instead of a single-point estimate. This stochasticity in the latent space allows the VAE to capture the underlying structure of the data in a more robust manner. The decoder network p_θ of the VAE maps those latent variables z to a reconstructed or decoded input vector $\hat{x}$.

The parameters of a VAE are trained by maximizing a variational lower bound on the evidence (or ELBO), which is equivalent to minimizing the following per-sample loss function for an input x:

$$\begin{aligned} L(x;\phi,\theta) &= E_{z\sim q_\phi(z|x)}\left[-\log p_\theta(x|z)\right] + D_{\mathrm{KL}}\left(q_\phi(z|x)||p(z)\right) \\ &= L^{\mathrm{recon}}(x;\phi,\theta) + L^{\mathrm{latent}}(x;\phi) \end{aligned} \tag{1}$$

where $q_\phi(z|x) = \mathcal{N}(\mu^{(x)}, \sigma^{(x)^2}\mathbf{I})$ denotes the posterior distribution over the latent variables z defined by the encoder given input x, $p(z) = \mathcal{N}(0, \mathbf{I})$ denotes the prior distribution over the latent variables, and D_{KL} denotes the Kullback–Leibler divergence [43]. It is a measure of the divergence between the approximate posterior distribution $q_\phi(z|x)$ and the prior distribution $p(z)$. It encourages the approximate posterior to be close to the prior distribution, which helps to regularize the latent space. To simplify the reconstruction term L^{recon}, we made the output of the decoder network p_θ to be deterministic, which is a common modification in the VAE literature. We define the reconstruction loss as the expected mean squared error between the original and decoded pixel values

$$L^{\mathrm{recon}}(x;\phi,\theta) = E_{\epsilon\sim\mathcal{N}(0,\mathbf{I})}\left[\sum_{p=1}^{N_{\mathrm{pixels}}}(\hat{x_p} - x_p)^2\right] \tag{2}$$

where x_p denotes the p^{th} pixel of the original input image and $\hat{x_p}$ denotes the p^{th} pixel of the decoded images $\hat{x} = p_\theta(z^{(x)})$ with $z^{(x)} = \mu^x + \sigma^x \bigodot \epsilon$ (reparameterization trick) and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.

C. Lateral Inhibition Using Gradient Masking

Lateral inhibition is a crucial concept in neural networks that imitate the behavior observed in biological systems. It involves a fascinating process where active neurons effectively suppress the activity of their neighboring neurons by establishing inhibitory connections. Neuroscience research has revealed that inhibitory circuitry may have a significant role in associative long-term depression (LTD) by ensuring that a postsynaptic cell is inhibited when specific inputs to that cell are active [44].

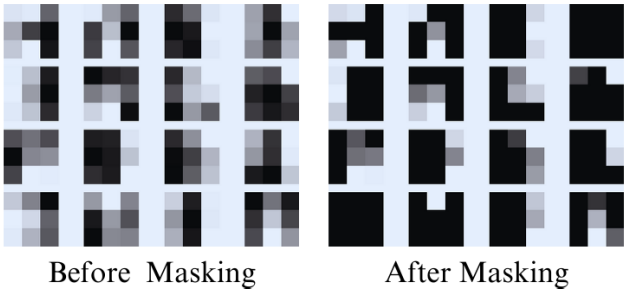


Fig. 2. Figure shows the gradients of the FMNIST dataset on some arbitrary experience before and after masking. The presence of masked gradients results in the enhancement of network robustness, ensuring that weight updates are influenced primarily by the most significant gradients.

Inspired by the work of Cao et al. [45], on a lateral inhibition-based CNN using an effective computational model to produce attention maps, in our work, we propose an inhibition mechanism using gradient masking. It involves modifying the gradient computations during backpropagation to encourage competition among neighboring neurons. By suppressing the gradients of neurons with lower activation values relative to their neighbors, gradient masking promotes the emergence of more distinct and selective features. In contrast to the approach taken by [45], where the network requires offline training on a pretrained model, our method employs inhibition in an online manner by masking the appropriate gradients during backpropagation.

Formally, the lateral inhibition value is computed as

$$x_{ij}^{LI} = a * \underbrace{e^{-\overline{x_{ij}}}}_{\text{average}} + b * \underbrace{\sum_{uv}(d_{uv}e^{-d_{uv}}\delta(x_{uv} - x_{ij}))}_{\text{differential}} \tag{3}$$

where $\overline{x_{ij}}$ denotes a point in the Max-C Map at the location (i, j), with x_{ij}^{LI} as its inhibition value. Max-C Map is composed of a matrix with dimensions (W, H) obtained by simple max operation on all channel values; x_{uv} is the neighbor of x_{ij} in its lateral inhibition zone (LIZ), which is formed by a square patch of k neighboring points of x_{ij}. The Euclidean distance between $\overline{x_{ij}}$ and x_{uv} is denoted by d_{uv}, which is normalized by the length of the LIZ. δ is defined as $\max(0, x)$.

The first term (average term) in (3) protects the neurons within a high response zone, whereas the second term (differential term) sharpens the object's boundaries and increases the contrast between objects and background in the protected zone created by the average term. The parameters a and b allow for a balance between the two terms, controlling their relative contributions.

Once the lateral inhibition values are computed, they can be used to mask the weights of the gradients during backpropagation, as shown in Fig. 2. By setting the gradients corresponding to unimportant features to 0, only the important feature gradients influence the weight updates. This masking of unimportant feature gradients helps to denoise the retained feature information in the weights.

D. Offline Memory Reorganization

The human brain possesses an extraordinary ability to independently reinforce and restructure memories during periods of inactivity, particularly during sleep [46]. Extensive research in psychology and neuroscience has delved into the mechanisms through which the brain autonomously reorganizes memories offline, leading to improvements in behavior. This work suggests that the offline brain can go beyond what is learned during wakefulness, giving rise to useful changes in behavior through spontaneous processes. For example, after learning a sequence of tasks, a period of sleep can facilitate the restoration of memories that may have been impaired or disrupted due to interference in a preceding wakeful period.

Once the network has undergone training on all the available experiences, we introduce a sleep phase where the stable model is trained on the pseudo episodic memory for a brief duration with a small learning rate. The inclusion of a sleep phase serves the purpose of allowing the network to restructure the acquired representations.

E. SynapNet Working

During each training step, the working model receives a training batch X_r from a non-independent identically distributed (i.i.d.) data stream and retrieves a random batch of synthetic images X_g from the pseudo episodic memory. This pseudo buffer contains synthetic images generated by a VAE after each experience. Upon the arrival of every experience, the VAE network is first trained, and then, a specific number of synthetic images are generated along with the corresponding class labels and stored in the pseudo buffer for replay. The working model learns representations and updates its semantic memory composed of the stable and plastic model. The semantic memories are designed such that the plastic model performs better on recent tasks, while the stable model prioritizes retaining information on older tasks.

Consequently, we prefer to use the logits from the stable model Z_S for older pseudo images and the plastic model Z_P for recent ones. While training on these experiences, the working model also applies gradient masking on the output of its convolutional layers to suppress the activity of neighboring neurons. The selected pseudo logits from the semantic memories are then employed to enforce a consistency loss on the working model so that it does not deviate significantly from the previously learned experiences. The overall loss function is composed of two components: the cross-entropy loss on the combined data stream and pseudo episodic memory (X) and the consistency loss on the pseudo images (X_g)

$$L = L_{CE}\left(\sigma(f_W(X)), Y\right) + \lambda L_{\text{mse}}\left(f_W(X_g), Z\right) \tag{4}$$

where σ denotes the softmax function, λ is the regularization parameter that maintains a tradeoff between learning new experiences and retaining the old information, and f_W denotes the working model.

After training on all the available experiences, the stable model goes through a sleep phase for memory reorganization. During the testing phase, the SynapNet algorithm leverages the stable model to make predictions based on past experiences, resembling the function of the neocortex in the brain. Conversely, the plastic model is employed for handling very recent experiences, akin to the role of the hippocampus. This division of roles reflects the brain's mechanism of using different structures for processing and recalling information from different time frames [47].

IV. Experimental Setup

In our research, we evaluated the effectiveness of the SynapNet algorithm across seven distinct datasets. Specifically, we employed MNIST [11], FMNIST [12], CIFAR10, CIFAR100 [13], and ImageNet100 [14] in a class-incremental manner, while PMNIST [15] and RMNIST [16] were utilized in a domain-incremental fashion. To ensure a fair and unbiased assessment of various CL methods, we leveraged the Avalanche framework [48] for experience generation and training of both the SynapNet algorithm and the baseline architectures. This approach allowed us to compare and analyze the performance of different CL methods accurately.

A. Model Architecture and Training

The SynapNet algorithm includes a VAE network that generates synthetic images and a CLS that learns sequentially. We used a VAE model in our study similar to the one from [31], composed of five convolutional layers with channel sizes of 16, 32, 64, 128, and 254. Each layer implements a 3×3 kernel, a padding of 1, and a stride of 1 in the first layer, while subsequent layers have a stride of 2. Batch normalization [49] is applied after each convolutional layer, followed by a rectified linear unit (ReLU) nonlinearity. Mirroring these convolutional layers, the decoder part of the VAE is made up of five deconvolutional or transposed convolutional layers [50] containing 128, 64, 32, 16, and three channels. The first four deconvolutional layers use a 4×4 kernel, a padding of 1, and a stride of 2 followed by a batch-norm and ReLU nonlinearity, while the final layer uses a 3×3 kernel, a padding of 1, and a stride of 1 with sigmoid activation at the end. All the layers are trained from scratch in an online fashion. In addition to the above architecture, for the RGB (32×32) images (CIFAR10, CIFAR100, and ImageNet100), we also introduce a skip connection between E32 and D128[1] to capture the high-level information in the images. The VAE is trained on normalized images using the Adam optimizer ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) [51] with a learning rate of 0.0001, a batch size of 32, and a latent embedding of 50 for MNIST and 100 for all other datasets.

For the MNIST dataset, all three models (stable, plastic, and working) consist of a convolutional layer with 32 channels, a 3×3 kernel, and a stride and padding of 1. This is followed by a max pooling layer with a 2×2 kernel, a stride of 1, and a padding of 1. Subsequently, three fully connected layers are employed with $32 \times 27 \times 27$, 300, and 100 neurons, respectively. ReLU nonlinearity is applied in all layers except for the final layer. For CIFAR100 and ImageNet100, we use

[1]E32 and D128 refer to the output of the encoder and decoder layers at channels 32 and 128, respectively.

TABLE I
HYPERPARAMETERS USED BY THE SYNAPNET ALGORITHM FOR VARIOUS DATASETS. SMUF REFERS TO THE STABLE MODEL UPDATE FREQUENCY, AND PMUF REFERS TO THE PLASTIC MODEL UPDATE FREQUENCY

Dataset	CLS epochs	VAE epochs	Batch size	Pseudo memory batch size	lr	SMUF	PMUF	λ
MNIST	25	15	64	64	0.001	0.10	0.65	0.75
FMNIST	15	40	32	32	0.0001	0.15	0.90	0.75
CIFAR10	45	50	128	64	0.01	0.30	0.90	0.15
CIFAR100	50	100	128	64	0.087	0.75	0.95	0.10
ImageNet100	50	100	128	64	0.1	0.5	1.0	0.15
PMNIST	35	15	1024	128	0.01	0.30	0.80	0.35
RMNIST	35	20	1024	128	0.01	0.35	0.75	0.25

the ResNet-18 [52] architecture without pretraining. For all other datasets, the base architecture that we employ is the encoder part of the VAE, which is then followed by three fully connected layers with 1016, 2000, and 1000 neurons. Similar to the MNIST case, the ReLU nonlinearity is applied in all layers except the final layer. The CLS architecture is trained using the stochastic gradient descent (SGD) optimizer [53] with default parameters. Prior to training, all the datasets are standardized with respective mean and variance. For the CIFAR10, CIFAR100, and ImageNet100 datasets, random horizontal flipping and random cropping are applied to both the data stream samples and the pseudo buffer samples. More details about the parameters of the models and generator are provided in Table I.

B. Evaluation Criteria and Baseline Architectures

Evaluation criteria for CL involve assessing the model's ability to adapt, retain, and generalize knowledge over time. In CL research, numerous evaluation standards exist. First and most significant is the capacity of the model to learn from new data while avoiding severe "catastrophic forgetting" of prior knowledge. Second is the model's ability to learn both effectively and efficiently from new incoming information, without being hindered by past tasks. The capacity of the model to transfer knowledge from one task to another also plays a vital role in leveraging previously acquired skills and experiences.

To assess the performance of our framework, we categorized the datasets into two groups: the first comprising datasets with ten classes and the second with datasets featuring 100 classes. In the first group, we employed the EWC [15], SI [24], and LWF [25] algorithms, all integrated with a generative replay buffer for rehearsal. In addition, we also included a "naive" method that utilizes a non-CL algorithm, as well as a joint method that undergoes offline training on the complete dataset to establish lower and upper bounds on the accuracy. For the second group, we utilized four distinct CL algorithms: DualNet, deep generative memory (DGM), brain-inspired replay (BI-R), looking through the past (LTP), and ABD. These methods either employ a generative replay architecture or utilize a CLS-inspired network, except for ABD, which is a nonexemplar-based method. All the algorithms in the second group used ResNet-18 architecture. To ensure a fair comparison with our approach, all algorithms were trained from scratch and underwent parameter tuning on a small validation set using the Optuna framework [54].

V. EXPERIMENTS

A thorough evaluation of the CL algorithms was performed using seven distinct classes, classified into class- and domain-incremental categories. To ensure the robustness and reliability of our results, we executed the experiments three times, each with a unique initialization. The average accuracy and standard deviation after three runs for all the datasets with varying pseudo buffer sizes are reported in Table II.

A. Class-Incremental Dataset

In the class-incremental setting, we conducted experiments using five datasets: MNIST, FMNIST, CIFAR10, CIFAR100, and ImageNet100. To maintain consistency, we divided the classes in each dataset into five or ten balanced experiences/tasks. In addition to the balanced experience tests, we also investigated the SynapNet algorithm on unbalanced experience data to simulate a more realistic scenario and evaluate the algorithm's performance under challenging conditions.

Fig. 3 (left) illustrates the accuracy of the SynapNet algorithm on different experiences for the MNIST dataset, both with and without the sleep phase. Surprisingly, we observed that the inclusion of the sleep phase provided marginal improvement in the overall accuracy of the model. Fig. 3 (right) provides the performance of benchmark models on various experiences serving as a reference baseline. In Fig. 4, we observe that the SynapNet algorithm demonstrates efficient learning of new experiences while maintaining the accuracy of previously learned ones. The rate of misclassification made by the model gradually decreases as it gets trained on new experiences.

We examined the effect of lateral inhibition and sleep phase on our algorithm's performance, in three different conditions using the FMNIST and CIFAR10 datasets. The first condition, as depicted in Fig. 5 (top left and top right), excluded both lateral inhibition and the sleep phase, enabling us to observe the algorithm's performance without these elements. It can also be referred to as the generative variant of the CLS-ER algorithm. In the second scenario, we activated the lateral inhibition but omitted the sleep phase. In the third and last conditions, we included both lateral inhibition and the sleep phase. This condition represents the combined impact of these

TABLE II

COMPARISON OF THE ACCURACY OF THE SynapNet ALGORITHM WITH BENCHMARK MODELS ON DIFFERENT DATASETS. THE BENCHMARK MODELS INCLUDED THE EWC, LWF, AND SI ALGORITHMS, ALL OF WHICH WERE EQUIPPED WITH A GENERATIVE REPLAY BUFFER. THE JOINT AND NAIVE MODELS WERE USED TO ESTABLISH UPPER AND LOWER BOUNDS ON THE ACCURACY

Pseudo Buffer	Method	Class-IL			Domain-IL	
		MNIST	FMNIST	CIFAR10	R-MNIST	P-MNIST
-	JOINT	98.40$\pm$0.34	92.40$\pm$0.21	84.69$\pm$0.49	98.0$\pm$0.18	95.80$\pm$0.09
-	NAIVE	20.0$\pm$0.0	20.0$\pm$0.0	12.40$\pm$0.09	35.19$\pm$5.40	33.14$\pm$5.64
500	EWC	63.20$\pm$4.60	38.80$\pm$1.67	23.39$\pm$2.23	68.39$\pm$7.34	40.0$\pm$2.89
	LWF	52.40$\pm$3.61	27.99$\pm$3.46	25.20$\pm$2.12	**71.0**$\pm$3.89	49.0$\pm$3.41
	SI	58.20$\pm$6.65	35.60$\pm$5.42	12.80$\pm$1.41	70.40$\pm$3.94	32.80$\pm$8.04
	SynapNet	**91.0**$\pm$1.78	**65.99**$\pm$6.96	**40.79**$\pm$2.65	69.12$\pm$2.47	**80.20**$\pm$0.77
1000	EWC	60.40$\pm$3.79	42.10$\pm$1.70	31.80$\pm$3.01	69.0$\pm$8.28	56.80$\pm$3.75
	LWF	45.60$\pm$3.74	29.23$\pm$3.53	32.0$\pm$1.43	72.40$\pm$4.42	54.80$\pm$3.42
	SI	57.99$\pm$4.04	43.12$\pm$6.70	16.99$\pm$4.64	69.40$\pm$4.56	40.40$\pm$7.63
	SynapNet	**91.0**$\pm$1.84	**67.0**$\pm$1.54	**45.39**$\pm$5.65	**73.0**$\pm$3.50	**83.0**$\pm$1.54
2500	EWC	57.40$\pm$4.56	45.23$\pm$2.58	42.80$\pm$6.88	69.80$\pm$7.02	53.80$\pm$3.58
	LWF	43.59$\pm$4.36	27.42$\pm$2.95	42.0$\pm$2.98	74.20$\pm$3.80	63.19$\pm$5.35
	SI	61.19$\pm$4.63	48.31$\pm$7.81	23.39$\pm$7.23	67.40$\pm$3.99	42.60$\pm$6.81
	SynapNet	**91.0**$\pm$1.20	**67.40**$\pm$5.36	**50.79**$\pm$6.59	**75.0**$\pm$1.96	**83.0**$\pm$1.68
5000	EWC	61.20$\pm$2.46	45.34$\pm$1.51	51.26$\pm$4.67	69.67$\pm$8.16	64.20$\pm$4.64
	LWF	44.10$\pm$4.13	30.31$\pm$2.44	47.79$\pm$3.19	74.22$\pm$3.69	63.60$\pm$3.25
	SI	69.98$\pm$5.34	53.10$\pm$4.21	24.11$\pm$9.82	73.05$\pm$4.31	57.0$\pm$8.09
	SynapNet	**92.0**$\pm$1.28	**72.0**$\pm$5.94	**60.59**$\pm$3.29	**75.19**$\pm$3.42	**85.20**$\pm$0.85

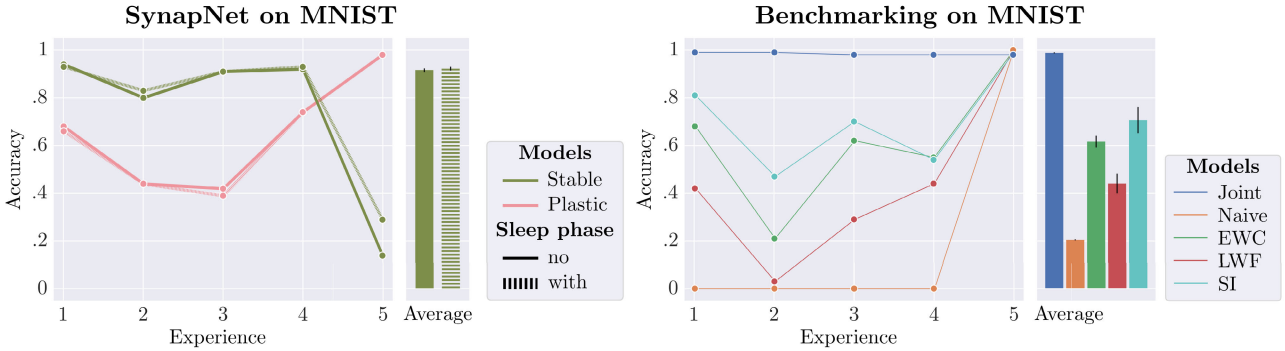


Fig. 3. Accuracy after every experience on the MNIST dataset with three runs; "Average" denotes the mean of the model's accuracy taken over all the experiences. PM denotes the plastic model, and SM denotes the stable model. Left: SynapNet accuracy with and without sleep phase. Right: benchmark model accuracy.

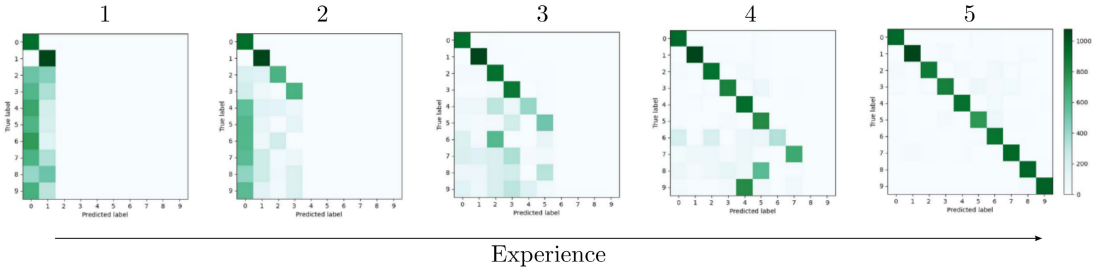


Fig. 4. Sequence of confusion matrices computed after each training experience for the SynapNet algorithm on the MNIST dataset. It demonstrates the capability of the algorithm to learn new experiences while also consolidating previously acquired knowledge.

two mechanisms on the algorithm's performance. The results clearly demonstrate the positive influence of lateral inhibition and the sleep phase on the model's accuracy. The accuracy of the algorithm considerably improved when both mechanisms were utilized. Fig. 5 (bottom left and bottom right) shows the accuracy of the benchmark models on the test experiences for the two datasets.

The right-hand side of Fig. 6 illustrates the accuracy of our algorithm alongside other baseline algorithms applied to the CIFAR100 dataset, which consists of 100 classes divided into ten equal tasks. Our algorithm exhibits higher accuracy compared to benchmark models, except for the ABD algorithm, which achieves comparable accuracy. Nonetheless, Fig. 7 reveals that the ABD algorithm demands substantial disk space to retain the feature generators after each task, whereas our algorithm necessitates minimal memory for storing the pseudo buffer.

On the left-hand side of Fig. 6, a comparison of our algorithm on the ImageNet100 dataset (a subset of the ImageNet dataset) with 100 classes, again partitioned into

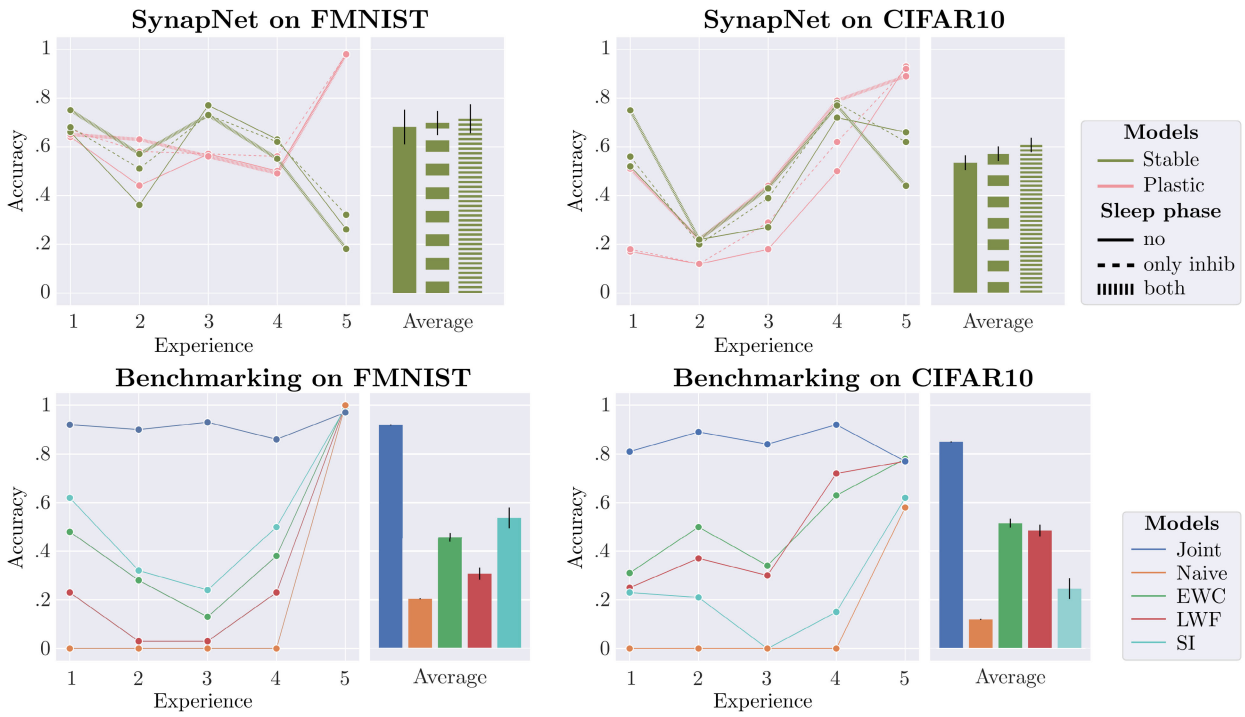


Fig. 5. Accuracy after every experience of the SynapNet algorithm on FMNIST (top left) and CIFAR10 (top right) datasets for three different conditions (with both inhibition mechanism and sleep phase disabled; inhibition mechanism enabled but sleep phase disabled; both inhibition mechanism and sleep phase enabled). Bottom left and right: benchmark model accuracy for each experience.

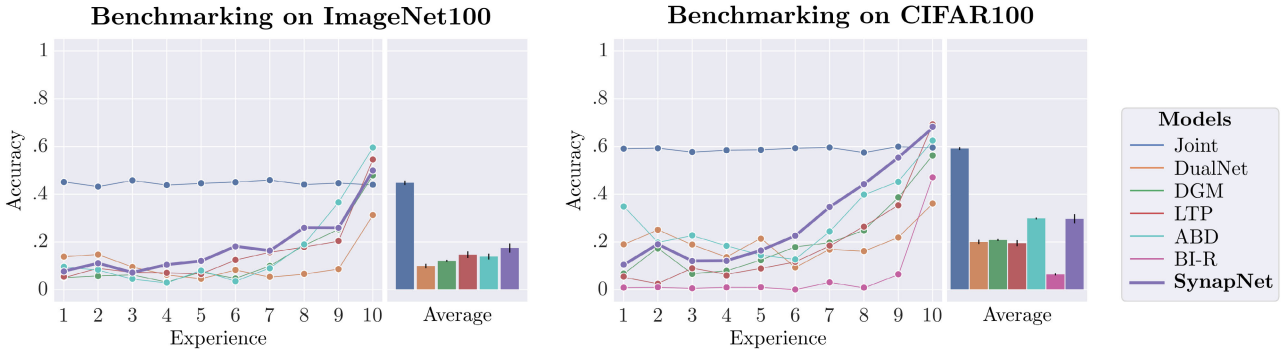


Fig. 6. Comparison between the accuracy of SynapNet algorithm and benchmark algorithms on ImageNet100 (ImageNet subset) and CIFAR100 datasets in class-incremental setting; each experience contains ten random classes sampled from the dataset.

ten tasks, is depicted. Our algorithm demonstrates superior accuracy in contrast to benchmark algorithms. However, we excluded BI-R as a benchmark algorithm for the ImageNet100 dataset due to its poor performance on the CIFAR100 data. This discrepancy arises from BI-R's prerequisite for pretraining on a base dataset, whereas, for our comparison, all algorithms were trained from scratch. In addition, BI-R does not store any buffer memory or generator/extractor network; thus, it was not featured in disk space plots. Further details and comparisons of the SynapNet algorithm against benchmark models across different numbers of tasks and buffer sizes are presented in Table III. It is worth noting that for a pseudo buffer size of 2000, the ABD algorithm outperforms our SynapNet algorithm on the CIFAR100 dataset with five and ten tasks. However, considering the disk space it requires, we contend that our algorithm's accuracy is satisfactory.

We conducted another experiment to simulate a real-life scenario by using the unbalanced experience test on the MNIST dataset. In this test, we divided the dataset into three separate experiences. The first experience consisted of three classes, followed by the second experience with four classes and, finally, the third experience with three classes. In addition to the unbalanced experiences, we also created a class imbalance scenario in the second experience by randomly sampling only 50% of the instances from the dataset for each class. At the end of each experience, we saved the stable and the plastic model along with the pseudo memory. This allowed us to capture the knowledge learned up to that point. When a new experience arrives, we load the model from the memory and continue training it sequentially with the new data. Fig. 8 illustrates the accuracy of the SynapNet algorithm throughout the experiment. The plot demonstrates the performance of the model as it learns from each experience

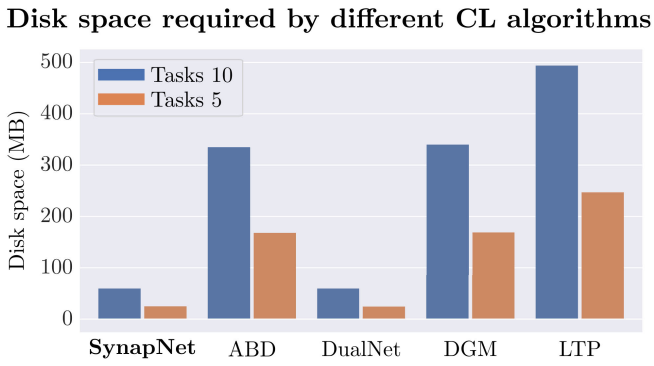


Fig. 7. Comparison of the disk space required to store the pseudo buffer in the case of SynapNet algorithm, feature generator in the case of ABD, buffer in the case of DualNet, weights and masks of the GAN generator in the case of DGM, and the previous VAE models in the case of LTP algorithm for the ImageNet100 dataset with ten tasks. Here, we only show the disk space requirements for the ImageNet100 dataset; however, CIFAR100 also follows an exact plot.

TABLE III

ACCURACY COMPARISON OF SynapNet ALGORITHM AND THE BENCHMARK ALGORITHMS ON CIFAR100 AND IMAGENET100 DATASETS WITH DIFFERENT BUFFER SIZES (NOTE THAT "*" DENOTES ALGORITHMS REQUIRING A BUFFER/PSEUDO BUFFER)

Pseudo Buffer	Tasks	Method	CIFAR100	ImageNet100
-	-	JOINT	$58.95_{\pm 0.627}$	$44.634_{\pm 0.890}$
5000	5	DualNet*	$20.01_{\pm 0.647}$	$11.906_{\pm 0.56}$
		DGM	$27.86_{\pm 0.67}$	$15.33_{\pm 0.41}$
		BI-R	$9.54_{\pm 1.17}$	-
		LTP	$25.19_{\pm 1.67}$	$18.91_{\pm 1.56}$
		ABD	$39.0_{\pm 0.280}$	$17.64_{\pm 0.78}$
		SynapNet*	$\mathbf{39.80}_{\pm 1.48}$	$\mathbf{23.20}_{\pm 1.26}$
5000	10	DualNet*	$19.84_{\pm 0.89}$	$9.741_{\pm 0.94}$
		DGM	$20.65_{\pm 0.398}$	$11.90_{\pm 0.43}$
		BI-R	$6.24_{\pm 1.45}$	-
		LTP	$19.32_{\pm 1.30}$	$14.47_{\pm 1.39}$
		ABD	$\mathbf{29.72}_{\pm 0.366}$	$13.71_{\pm 1.23}$
		SynapNet*	$29.50_{\pm 1.96}$	$\mathbf{17.30}_{\pm 1.939}$
2000	5	DualNet*	$11.77_{\pm 0.52}$	$7.608_{\pm 0.97}$
		DGM	$27.86_{\pm 0.67}$	$15.33_{\pm 0.41}$
		BI-R	$9.54_{\pm 1.17}$	-
		LTP	$25.19_{\pm 1.67}$	$18.91_{\pm 1.56}$
		ABD	$\mathbf{39.0}_{\pm 0.280}$	$17.64_{\pm 0.78}$
		SynapNet*	$34.40_{\pm 0.748}$	$\mathbf{20.599}_{\pm 0.802}$
2000	10	DualNet*	$10.98_{\pm 0.43}$	$5.59_{\pm 0.34}$
		DGM	$20.65_{\pm 0.398}$	$11.90_{\pm 0.43}$
		BI-R	$6.24_{\pm 1.45}$	-
		LTP	$19.32_{\pm 1.30}$	$14.47_{\pm 1.39}$
		ABD	$\mathbf{29.72}_{\pm 0.366}$	$13.71_{\pm 1.23}$
		SynapNet*	$27.80_{\pm 1.24}$	$\mathbf{14.50}_{\pm 1.35}$

and adapts to the increasing number of classes. It is worth noting that the accuracy achieved in the unbalanced experience test is slightly lower than that of the normal MNIST test. However, considering the imbalanced classes and experiences, this slight decrease in accuracy is reasonable and expected.

B. Domain-Incremental Dataset

In the domain-incremental setting, we employed two variations of the MNIST dataset: RMNIST and PMNIST. In the RMNIST dataset, the digits were randomly rotated by a specific angle for all experiences. On the other hand, in the PMNIST dataset, the pixels were randomly permuted based

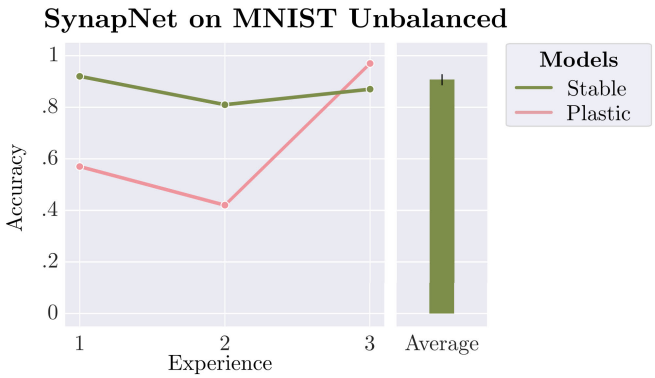


Fig. 8. To simulate a real-world scenario, we tested the SynapNet algorithm on unbalanced MNIST experiences in a class-incremental setting.

on a specific permutation for each experience. Similar to the class-incremental scenario, we divided the dataset into five experiences, with each experience representing a random rotation or pixel permutation.

Fig. 9 (top left) shows the accuracy results of the SynapNet algorithm on the RMNIST dataset consisting of five different experiences. Each experience comprises all ten classes but with random rotations applied to the images. Similarly, Fig. 9 (top right) illustrates the accuracy of the algorithm on the PMNIST dataset. The PMNIST dataset also consists of five experiences, each containing ten classes, but with random pixel permutations applied to the images. In comparison to the benchmark models [as shown in Fig. 9 (bottom left) and (bottom right)], our algorithm demonstrates higher accuracy.

VI. ROBOTIC APPLICATION

In the context of a sorting industry, where the robot's task is to differentiate objects based on their shapes and sizes, utilizing a CL algorithm is highly beneficial. Instead of starting the training process anew each time a new object is encountered, this approach allows the algorithm to be trained incrementally by building upon its existing knowledge base, making it more efficient and reducing the downtime associated with retraining from the ground up.

To simulate this sorting scenario, we applied the SynapNet algorithm to classify a wide range of objects incrementally in a real-time dynamic environment using a soft pneumatic gripper [55] equipped with two flex sensors and two force sensors [56], [57]. The remarkable aspect of the algorithm is its ability to handle an ever-growing number of object classes. Each time a new object is encountered, the algorithm intelligently expands the last layer of the network to accommodate the additional class seamlessly.

Fig. 10(a) illustrates the pneumatic gripper, which connects to a control box responsible for sending pressure signals for the actuation of the fingers. The four sensors are attached temporarily to the soft fingers using rubber bands. We used the commercially available resistive flex sensor [58] for measuring the bending of the finger and a force-sensing resistor (FSR) [59] to quantify the force applied to the fingertip. The outputs from the flex and force sensors are directed to an Arduino Due board, which, in turn, is connected to a computer for further processing and analysis.

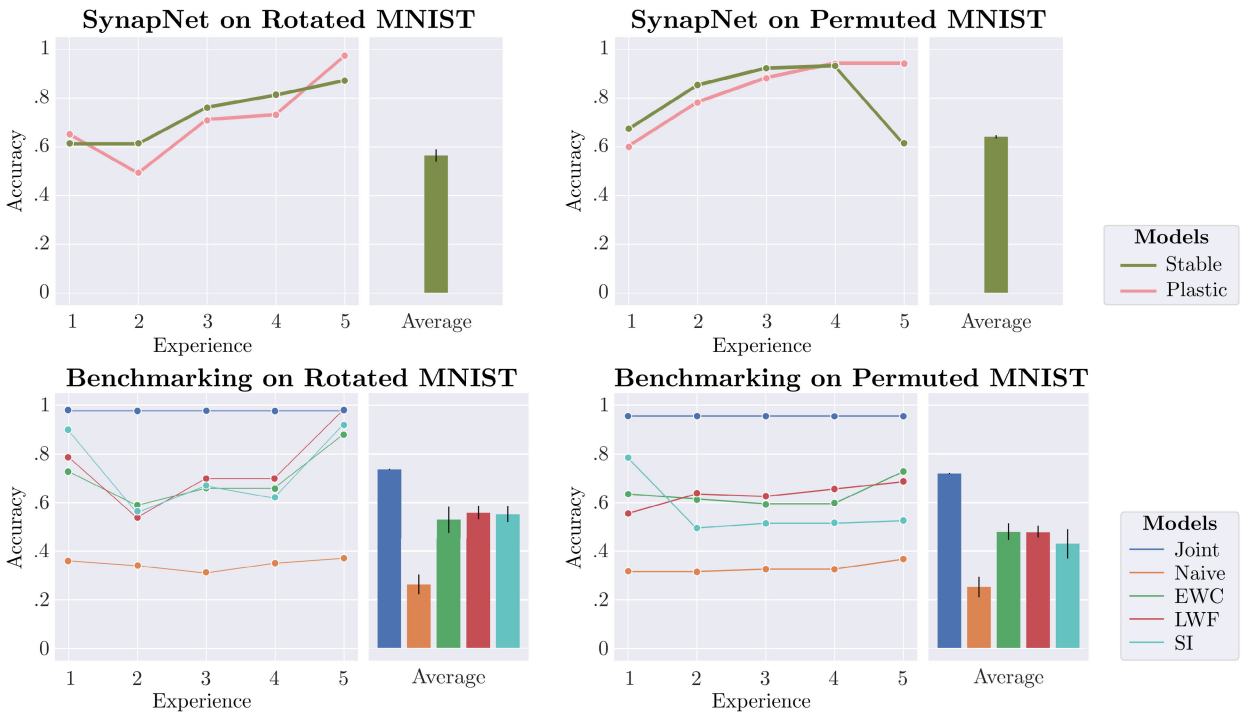


Fig. 9. Top left and top right: SynapNet accuracy on RMNIST and PMNIST datasets in domain-incremental setting; each experience contains all the ten classes but with random rotation or permutation applied to it in every experience. Bottom left and bottom right: benchmark model accuracy on RMNIST and PMNIST datasets.

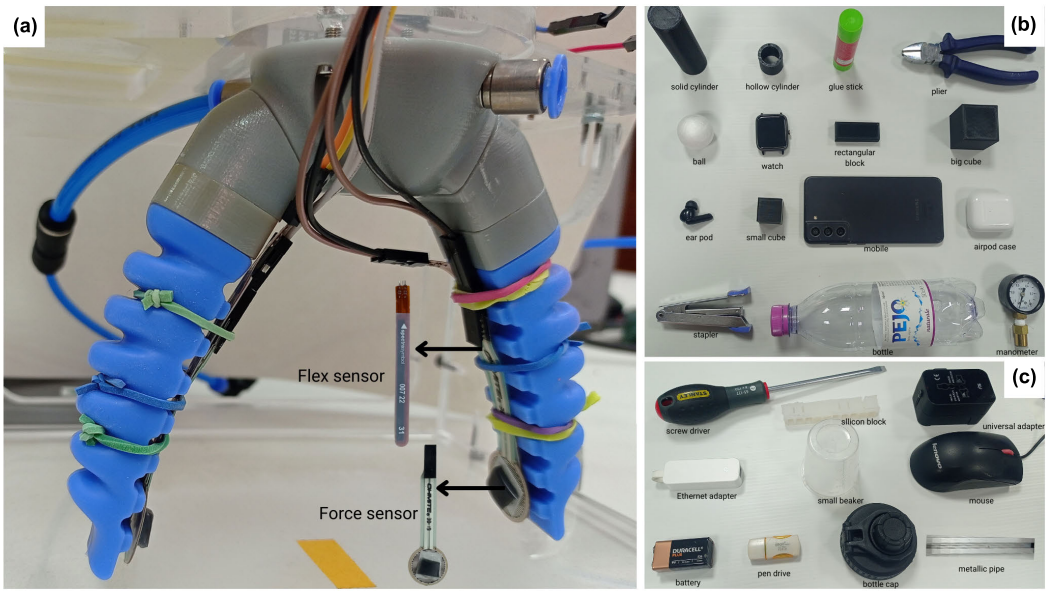


Fig. 10. Experimental setup used for SynapNet application: (a) soft pneumatic gripper with two flex and two force sensors, (b) set of 15 diverse objects used for offline data collection and experience generation to train the SynapNet algorithm sequentially, and (c) additional set of ten new objects for real-time dynamic training and testing purposes.

A. Data Collection and Experience Generation

To first train the SynapNet algorithm offline incrementally, we gather sensor data for 15 distinct objects, as illustrated in Fig. 10(b). For each object, the gripper acquires 50 data points, considering various orientations of the object. During the data collection process, for each data point, the gripper holds and releases the object repeatedly, with this cycle lasting ≈39 s.

Each data point is represented as a 600-D vector, encapsulating the sensor signals recorded during that time interval. The dataset acquisition for every object takes about 37 min to complete on a computer equipped with an Intel Core[2] i7 CPU @2.80 GHz (GTX 1070 GPU) running Windows 10.

After collecting the dataset for all 15 objects, we divide the complete dataset into five experiences. Each experience comprises data from three randomly selected objects. For each experience, 80% of the data is allocated for training, while the remaining 20% is used for validation purposes.

[2]Trademarked.

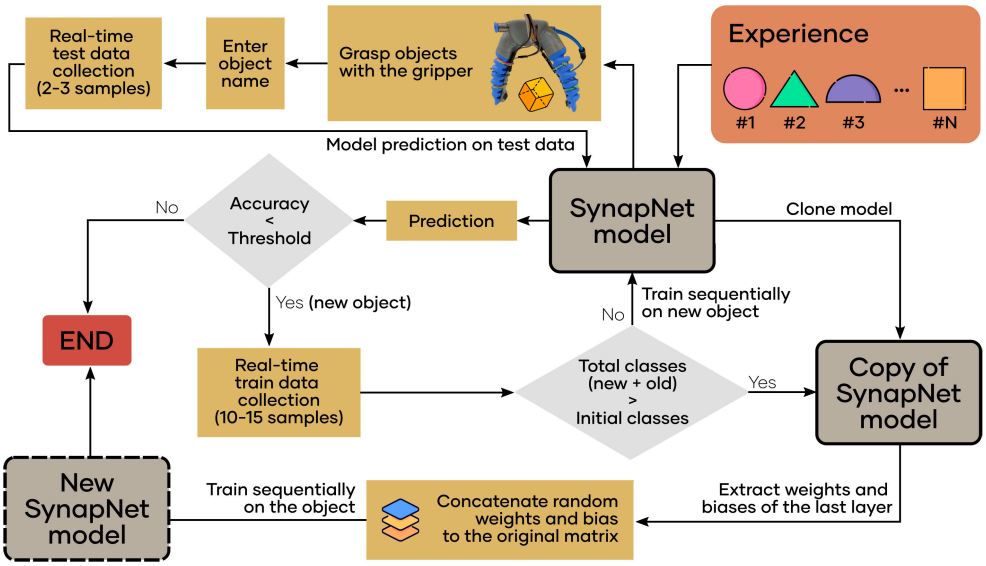


Fig. 11. Application of the SynapNet algorithm in a real-time dynamic environment. The architecture depicts an autonomous sequential training of the algorithm by dynamically expanding the output layer of the network upon encounter with an unknown class. This capability allows the algorithm to continuously learn from new data on the fly, making it well-suited for handling an evolving environment.

B. Real-Time Test and Network Expansion

The SynapNet algorithm undergoes real-time testing on both known and unknown objects. The process, as depicted in Fig. 11, involves the gripper collecting data for a test object, along with its associated class label. These data are then used by the algorithm to make predictions. Three possible scenarios can occur based on the model predictions.

1) If the accuracy of the prediction exceeds the predefined threshold[3] (hyperparameter), the algorithm considers the predictions satisfactory and halts further processing.
2) If the prediction accuracy falls below the threshold, the algorithm enters the training phase and starts collecting 10–15 data points for that particular object. Two situations can arise.
   a) If the object's class is already present in the list of classes that the model was trained on, the algorithm performs sequential training to enhance accuracy on this familiar object class.
   b) If the object's class is not present in the list, but the total number of classes (including both old and new classes) does not exceed the initially defined class limit, the algorithm sequentially trains the model to incorporate this new class.
3) If the prediction accuracy is below the threshold, the object's class is not in the existing class list, and the total number of classes exceeds the initially defined class limit, the algorithm dynamically expands. It creates a copy of the model and extends the output layer by adding an extra neuron with random weight and bias to it. Subsequently, the algorithm incrementally trains this extended model on the dataset specific to the new class. Notably, the sequential training takes a very small duration for both known and unknown classes, thereby maintaining real-time performance. However, it is worth noting that there are instances where the model goes into the training phase not because of incorrect predictions but due to changes in the sensor alignment caused by temporary fixation using rubber bands. This suggests that the model's performance might be affected by variations in sensor input conditions, and addressing this sensitivity to sensor alignment issues could be an area of improvement.

[3]It is calculated by taking the ratio of the number of expected correct predictions to the total number of real-time collected test data points.

C. Performance Evaluation

After subjecting the algorithm to a performance test using ten unseen objects [as depicted in Fig. 10(c)], we observed interesting results. Fig. 12 (left) illustrates the test accuracies of the five experiences following the model's training on these new, unseen objects. The black dashed line represents the accuracy of the experiences before the model underwent training with the new objects. Remarkably, the model not only learned to recognize the new objects but also exhibited improvements in the accuracy of the experiences that it had already learned. This suggests that the training on new unknown objects had a positive impact on the model's overall performance, extending its capabilities beyond the new data to enhance its understanding of previously seen data as well. To gain a better understanding, let us define the term knowledge transfer (KT) as follows:

KT = (Initial experience accuracy − Experience accuracy after addition of new class)

$$\text{KT} = \begin{cases} \text{Backward knowledge transfer}, & \text{if KT} < 0 \\ \text{Partial forgetting}, & \text{if KT} > 0 \end{cases}.$$

In Fig. 12 (right), after training the SynapNet algorithm on the "mouse" class, we observe that the KT for experience 1 is negative, indicating a backward KT, which ultimately improves the accuracy of that experience. On the other hand, for experience 5, KT is positive. A straightforward explanation for this positive KT is that the new objects are entirely different from the objects already present in that experience.

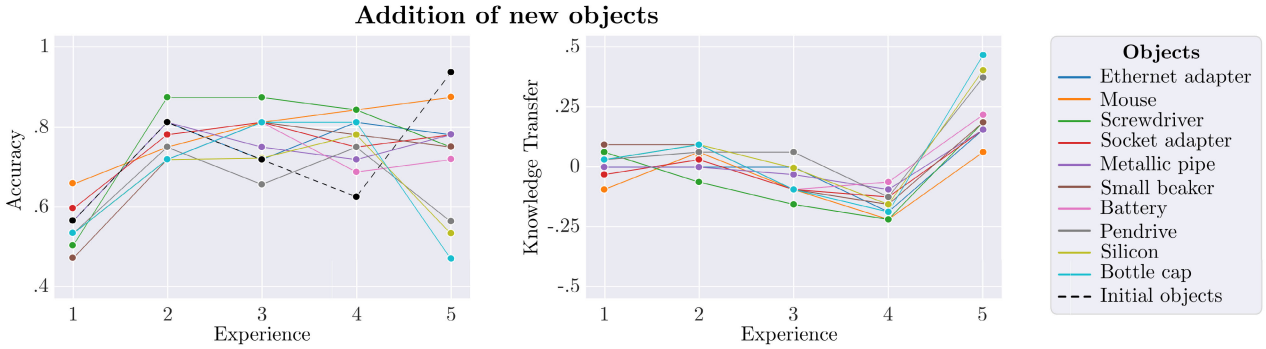


Fig. 12. SynapNet performance evaluation with increasing number of classes. Left: experience accuracy in the addition of new objects. The algorithm, in addition to learning new classes, also improves the accuracy of the already learned classes. Right: KT refers to the difference between the initial experience accuracy and the accuracy after the addition of new objects, and a negative KT refers to backward KT, whereas a positive KT denotes partial catastrophic forgetting of previous classes in that experience.

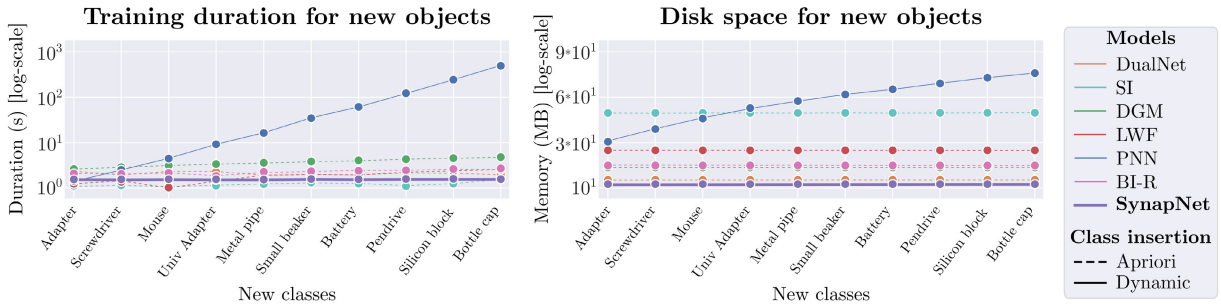


Fig. 13. Time (left) and space (right) complexity of the SynapNet algorithm in comparison to the benchmark algorithms on the encounter of new objects by the gripper. The Y-axis for both plots is in the log scale to enhance clarity.

Fig. 13 depicts the space and time complexity of the SynapNet algorithm and the benchmark algorithms when learning new classes incrementally in real time. Among these, five are static algorithms (DualNet, SI, DGM, LWF, and BI-R), while PNN is a dynamic algorithm. Our results indicate that SynapNet incurs minimal memory overhead compared to the other benchmark algorithms (with an increase of only 0.023 ± 0.0047 MB per class). This is primarily because both the stable and plastic model architectures are relatively simple, and the pseudo buffer consists of sensor signals that occupy negligible memory. In addition, when encountering new objects, our algorithm only adds a new node, while the static algorithms require prior knowledge of the number of classes, leading to constant memory usage throughout the experiment except for the PNN algorithm, which adds a new column, resulting in increased memory usage. The training duration, however, is minimal for all the algorithms (except PNN) with SI being the fastest. This speed advantage of SI is due to the absence of any distillation technique and the inclusion of only an extra regularization term in the loss function to preserve previous knowledge.

VII. Conclusion and Discussion

In this article, we proposed a brain-inspired CL algorithm combining the concept of CLS with a VAE-based pseudo episodic memory along with the incorporation of lateral inhibition and a sleep phase. This combination enabled our algorithm to learn incrementally from data with different modalities and object classes. We evaluated the performance of our algorithm with distinct SOTA CL algorithms on both balanced and unbalanced experiences across seven diverse datasets. SynapNet surpassed the compared benchmark methods demonstrating its ability to recall and consolidate recently learned information while also retaining old information. Moreover, the real-time dynamic application of our CL algorithm in object classification presents a novel application in autonomous systems, enabling them to adapt and interact with a changing environment effectively.

As part of our future work, we aim to enhance the algorithm further by integrating the VAE architecture directly into the CLS model, eliminating the need for a separate pseudo buffer. This improvement is expected to reduce the network's stochasticity and increase its capacity to handle a larger number of experiences during training. In addition, we plan to explore the algorithm's effectiveness in a real-time domain-incremental scenario, where changes in sensor position will not significantly impact the model's predictions.

Appendix
Real-Time Application Model Architecture

The input to the VAE network is a 600-D vector, which represents the signal from two force sensors and two flex sensors. The dataset contains 50 data points for each object, which are then divided into five experiences for training the algorithm. In addition, a separate dataset containing ten data points per object is gathered specifically for the testing phase. Prior to training, all the features are normalized between -1 and 1.

The VAE network comprises an encoder and a decoder. In the encoder, there are four 1-D convolutional layers with

channel sizes of 512, 256, 128, and 64. Each layer uses a 3×3 kernel, applies padding of 1, and has a stride of 1 in the first layer, while subsequent layers have a stride of 2. After each convolutional layer, 1-D batch normalization is applied, followed by a leaky ReLU activation function with a negative slope of 0.01.

On the other hand, the decoder consists of four 1-D transposed convolutional layers with channel sizes of 128, 256, 512, and 600. All deconvolutional layers use a 3×3 kernel and have a padding of 1, a stride of 2 in the first three layers, and a stride of 1 in the final layer. Each layer is accompanied by batch normalization and leaky ReLU activation, except for the final layer, which uses the tanh activation function. During training, the VAE is optimized using the Adam optimizer with default parameters, a learning rate of 0.0001, a batch size of 16, and a latent dimension of 100 for encoding the input signals.

The architecture of the stable, plastic, and working models includes two 1-D convolutional layers, followed by a linear layer with an output dimension equal to the number of classes. In the dynamic scenario where the number of encountered classes keeps on growing, an additional neuron is appended to the output layer for each new class. The two convolutional layers are composed of 512 and 256 channels, a kernel size of 3, a padding of 1, and a stride of 2 in the first layer and 1 in the second layer. After each convolutional layer, there is a batch normalization layer followed by a leaky ReLU activation function.

REFERENCES

[1] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54–71, May 2019, doi: 10.1016/j.neunet.2019.01.012.

[2] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," 2019, *arXiv:1907.00182*.

[3] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Inf. Fusion*, vol. 58, pp. 52–68, Jun. 2020, doi: 10.1016/j.inffus.2019.12.004.

[4] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects," *Frontiers Psychol.*, vol. 4, p. 504, Jul. 2013, doi: 10.3389/fpsyg.2013.00504.

[5] A. D. Friederici and W. Singer, "Grounding language processing on basic neurophysiological principles," *Trends Cognit. Sci.*, vol. 19, no. 6, pp. 329–338, Jun. 2015.

[6] G. M. van de Ven, T. Tuytelaars, and A. S. Tolias, "Three types of incremental learning," *Nature Mach. Intell.*, vol. 4, no. 12, pp. 1185–1197, Dec. 2022, doi: 10.1038/s42256-022-00568-3.

[7] E. Arani, F. Sarfraz, and B. Zonooz, "Learning fast, learning slow: A general continual learning method based on complementary learning system," 2022, *arXiv:2201.12604*.

[8] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? Complementary learning systems theory updated," *Trends Cognit. Sci.*, vol. 20, no. 7, pp. 512–534, Jul. 2016, doi: 10.1016/j.tics.2016.05.004.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[10] Y. Hao et al., "Universal soft pneumatic robotic gripper with variable effective length," in *Proc. 35th Chin. Control Conf. (CCC)*, Jul. 2016, pp. 6109–6114.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[12] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

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

[14] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[15] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017, doi: 10.1073/pnas.1611835114.

[16] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.

[17] Y. Zhao, D. Saxena, and J. Cao, "AdaptCL: Adaptive continual learning for tackling heterogeneity in sequential datasets," 2022, *arXiv:2207.11005*.

[18] A. A. Rusu et al., "Progressive neural networks," 2016, *arXiv:1606.04671*.

[19] C. Fernando et al., "PathNet: Evolution channels gradient descent in super neural networks," 2017, *arXiv:1701.08734*.

[20] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 3598–3607.

[21] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2013, *arXiv:1312.6211*.

[22] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, Aug. 2007, doi: 10.1007/s00365-006-0663-2.

[23] F. Piqué, H. T. Kalidindi, L. Fruzzetti, C. Laschi, A. Menciassi, and E. Falotico, "Controlling soft robotic arms using continual learning," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 5469–5476, Apr. 2022.

[24] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3987–3995.

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

[26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[27] L.-J. Lin and T. M. Mitchell, *Memory Approaches to Reinforcement Learning in Non-Markovian Domains*. Citeseer, 1992. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f2eb733470921af04df5c611a6a3c76c281ce498

[28] 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. 2001–2010.

[29] I. Goodfellow et al., "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[30] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.

[31] G. M. van de Ven, H. T. Siegelmann, and A. S. Tolias, "Brain-inspired replay for continual learning with artificial neural networks," *Nature Commun.*, vol. 11, no. 1, p. 4069, Aug. 2020, doi: 10.1038/s41467-020-17866-2.

[32] H. Khan, N. C. Bouaynaya, and G. Rasool, "Brain-inspired continual learning: Robust feature distillation and re-consolidation for class incremental learning," *IEEE Access*, vol. 12, pp. 34054–34073, 2024.

[33] V. Khan, S. Cygert, B. Twardowski, and T. Trzcinski, "Looking through the past: Better knowledge retention for generative replay in continual learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2023, pp. 3496–3500.

[34] W. Chenshen et al., "Memory replay gans: Learning to generate images from new categories without forgetting," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, Montréal, QC, Canada, 2018, pp. 5966–5976.

[35] O. Ostapenko, M. Puscas, T. Klein, P. Jähnichen, and M. Nabi, "Learning to remember: A synaptic plasticity driven framework for continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11321–11329.

[36] Q. Pham, C. Liu, and S. Hoi, "DualNet: Continual learning, fast and slow," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 16131–16144.

[37] R. Kemker and C. Kanan, "FearNet: Brain-inspired model for incremental learning," 2017, *arXiv:1711.10563*.

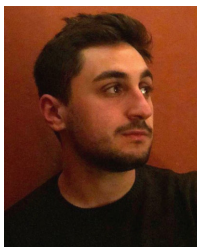
[38] F. Zhu, X.-Y. Zhang, C. Wang, F. Yin, and C.-L. Liu, "Prototype augmentation and self-supervision for incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 5871–5880.
[39] J. Smith, Y.-C. Hsu, J. Balloch, Y. Shen, H. Jin, and Z. Kira, "Always be dreaming: A new approach for data-free class-incremental learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Jun. 2021, pp. 9374–9384.
[40] G. Petit, A. Popescu, H. Schindler, D. Picard, and B. Delezoide, "FeTrIL: Feature translation for exemplar-free class-incremental learning," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2023, pp. 3900–3909.
[41] D. Goswami, Y. Liu, B. Twardowski, and J. van de Weijer, "FeCAM: Exploiting the heterogeneity of class distributions in exemplar-free continual learning," 2023, *arXiv:2309.14062*.
[42] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.
[43] J. Shlens, "Notes on Kullback–Leibler divergence and likelihood," 2014, *arXiv:1404.2000*.
[44] K. D. Miller, "Synaptic economics: Competition and cooperation in synaptic plasticity," *Neuron*, vol. 17, no. 3, pp. 371–374, Sep. 1996, doi: 10.1016/s0896-6273(00)80169-5.
[45] C. Cao, Y. Huang, Z. Wang, L. Wang, N. Xu, and T. Tan, "Lateral inhibition-inspired convolutional neural network for visual attention and saliency detection," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–8, doi: 10.1609/aaai.v32i1.12238.
[46] Z. Zhou, G. Yeung, and A. C. Schapiro, "Self-recovery of memory via generative replay," 2023, *arXiv:2301.06030*.
[47] A. R. Preston and H. Eichenbaum, "Interplay of hippocampus and prefrontal cortex in memory," *Current Biol.*, vol. 23, no. 17, pp. 764–773, Sep. 2013, doi: 10.1016/j.cub.2013.05.041.
[48] 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. [Online]. Available: https://avalanche.continualai.org
[49] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[50] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2018–2025, doi: 10.1109/ICCV.2011.6126474.
[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[53] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.
[54] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 2623–2631.
[55] G. Zhong, Y. Hou, and W. Dou, "A soft pneumatic dexterous gripper with convertible grasping modes," *Int. J. Mech. Sci.*, vols. 153–154, pp. 445–456, Apr. 2019, doi: 10.1016/j.ijmecsci.2019.02.028.
[56] R. P. Babadian, K. Faez, M. Amiri, and E. Falotico, "Fusion of tactile and visual information in deep learning models for object recognition," *Inf. Fusion*, vol. 92, pp. 313–325, Apr. 2023.
[57] G. Soter, A. Conn, H. Hauser, and J. Rossiter, "Bodily aware soft robots: Integration of proprioceptive and exteroceptive sensors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 2448–2453.
[58] G. Saggio, F. Riillo, L. Sbernini, and L. R. Quitadamo, "Resistive flex sensors: A survey," *Smart Mater. Struct.*, vol. 25, no. 1, Jan. 2016, Art. no. 013001.
[59] A. Hollinger and M. M. Wanderley, "Evaluation of commercial force-sensing resistors," in *Proc. Int. Conf. New Interfaces Musical Expression*, Paris, France, 2006, pp. 4–8.

Nilay Kushawaha received the bachelor's degree in physics from the University of Delhi, Delhi, India, in 2020, and the master's degree in physics from Indian Institute of Technology Indore, Indore, India, in 2022. He is currently pursuing the Ph.D. degree in biorobotics and AI with The Biorobotics Institute, Scuola Superiore Sant'Anna, Pisa, Italy.

His current research interests include continual learning, brain-inspired algorithms, reinforcement learning, and control theory.

Lorenzo Fruzzetti received the bachelor's degree in biological science and the master's degree in neuroscience from the University of Pisa, Pisa, Italy, in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree in biorobotics with Scuola Superiore Sant'Anna, The Biorobotics Institute, Pisa.

His current research interests include reinforcement learning, continual learning, spiking neural networks, brain-inspired algorithms, and data analysis.



Enrico Donato (Member, IEEE) received the B.Sc. degree in computer engineering from the University of Calabria, Rende, Italy, in 2018, the joint M.Sc. degree in bionics engineering from the University of Pisa, Pisa, Italy, and the Scuola Superiore Sant'Anna, Pisa, in 2020. He is currently pursuing the Ph.D. degree in biorobotics with the Scuola Superiore Sant'Anna, The Biorobotics Institute, Pisa.

His current research interests focus on learning and control strategies for bioinspired and soft robots with sensory feedback.

Egidio Falotico (Member, IEEE) received the M.S. degree in computer science from the University of Pisa, Pisa, Italy, in 2008, the Ph.D. degree in biorobotics from Scuola Superiore Sant'Anna (SSSA), Pisa, in 2013, and the Ph.D. degree in cognitive science from University Pierre et Marie Curie, Paris, France, in March 2013.

He is currently an Assistant Professor with SSSA, The BioRobotics Institute. He is currently the Deputy Leader and the Publications Manager in the Sub-Project 10 of the Human Brain Project. He is the author or co-author of more than 40 international peer-reviewed papers. He regularly serves as a reviewer for more than ten international ISI journals. He has been involved in some EU-funded projects (I-SUPPORT, SWARMs, SMARTE, RoboSoM, and RobotCub), focusing on the development of brain-inspired algorithms for robot control. His research interests focus on neurorobotics, i.e., the implementation of brain models from neuroscience in robots.

Open Access funding provided by 'Scuola Superiore "S.Anna" di Studi Universitari e di Perfezionamento' within the CRUI CARE Agreement