TinySubNets: An Efficient and Low Capacity Continual Learning Strategy

Marcin Pietroń[1], Kamil Faber[1], Dominik Żurek[1], Roberto Corizzo[2]

[1]AGH University of Krakow, Krakow, Poland
[2]American University, Washington DC, USA
pietron@agh.edu.pl, kfaber@agh.edu.pl, dzurek@agh.edu.pl, rcorizzo@american.edu

Abstract

Continual Learning (CL) is a highly relevant setting gaining traction in recent machine learning research. Among CL works, architectural and hybrid strategies are particularly effective due to their potential to adapt the model architecture as new tasks are presented. However, many existing solutions do not efficiently exploit model sparsity, and are prone to capacity saturation due to their inefficient use of available weights, which limits the number of learnable tasks. In this paper, we propose TinySubNets (TSN), a novel architectural CL strategy that addresses the issues through the unique combination of pruning with different sparsity levels, adaptive quantization, and weight sharing. Pruning identifies a subset of weights that preserve model performance, making less relevant weights available for future tasks. Adaptive quantization allows a single weight to be separated into multiple parts which can be assigned to different tasks. Weight sharing between tasks boosts the exploitation of capacity and task similarity, allowing for the identification of a better trade-off between model accuracy and capacity. These features allow TSN to efficiently leverage the available capacity, enhance knowledge transfer, and reduce computational resources consumption. Experimental results involving common benchmark CL datasets and scenarios show that our proposed strategy achieves better results in terms of accuracy than existing state-of-the-art CL strategies. Moreover, our strategy is shown to provide a significantly improved model capacity exploitation.

Code — https://github.com/lifelonglab/tinysubnets

Introduction

Continual learning (CL) is a machine learning paradigm aiming at designing effective methods and strategies to analyze data in complex and dynamic real-world environments (Parisi et al. 2019). As models are challenged with multiple tasks over their lifetime, a desired property for CL strategies is to maintain a high performance across all tasks. This paradigm is receiving increasing attention, which led to many works being proposed (Parisi et al. 2019; Baker et al. 2023; Faber et al. 2023; Wortsman et al. 2020; Zhou et al. 2023). There are three main types of CL strategies: rehearsal (also known as experience replay), regularization, and architectural.

Among architectural strategies, forget-free approaches are particularly relevant. They typically accommodate new tasks by assigning a subset of available model weights to each task. State-of-the-art approaches in this category include Packnet (Mallya and Lazebnik 2017), WSN (Kang et al. 2022), AdaQPacknet (Pietron et al. 2023), and DyTox (Douillard et al. 2022). However, their effectiveness strongly depends on the degree to which they can exploit model capacity to accommodate as many tasks as possible. One important pitfall of most existing pruning-based architectural forget-free methods is that they are limited in their ability to exploit model sparsity, since each layer is pruned with the same constant sparsity level. As a result, they do not properly tune the number of weights to be removed while preserving classification accuracy (Xu et al. 2021). A possible approach to address this issue is incorporating different sparsity levels for each layer (Pietron et al. 2023). Another general limitation of recent forget-free methods is that they are prone to a quick saturation of the available model capacity in that they cannot assign more than one value to each weight and are, therefore, limited by the available weights, as in (Douillard et al. 2022).

In this paper, we propose TinySubNets (TSN), a novel forget-free method for continual image classification that addresses these limitations by incorporating adaptive pruning with sparsity-level identification and adaptive non-linear weight quantization (Figure 1). To effectively exploit model sparsity, TSN incorporates pruning with different sparsity levels for each layer. Moreover, to deal with the issue of quickly saturating model capacity, TSN performs an adaptive quantization stage, where each task quantizes its weights via a stored codebook obtained via clustering. Quantized weights can be shared between tasks if their KL-divergence is low, or they can be separated into components, i.e. subsets of the available 32 bits, if their KL-divergence is high, resulting in totally independent memory banks with reduced bit-widths. Weight sharing is carried out through a trainable mask that automatically selects relevant weights for the current task from previous tasks during gradient descent learning, minimizing the bias associated with learning previous tasks. Thanks to these capabilities, our strategy allows model capacity to be drastically reduced while preserving model accuracy.

In summary, the contributions of our paper are as follows:

- We propose a forget-free model-agnostic continual strategy that adapts model architectures via adaptive pruning,

quantization, and fine-tuning stages.

- We devise an approach to seamlessly perform and combine quantization with weight sharing and replay memory for storing task samples, leading to an effective trade-off between model performance and capacity exploitation.
- We showcase the positive impact of our proposed strategy on model performance through an extensive experimental analysis. Our results show that TSN can outperform state-of-the-art continual learning strategies with commonly adopted scenarios and datasets.

In the following sections, we summarize related works in CL strategies and pruning methods. Then, we describe our proposed TSN method. We follow with a description of our experimental setup, and we present the results extracted in our experiments. Finally, we provide a summary of the results obtained and outline relevant directions for future work.

Related Works

Continual learning methods are commonly categorized as replay-based, regularization-based, and architectural-based (Parisi et al. 2019). Replay-based methods usually store some of the experiences form the past episodes and replay them while training with data from new tasks (Parisi et al. 2019). The most popular methods are GEM (David Lopez-Paz 2017), A-GEM (Chaudhry et al. 2019), and GDumb (Ameya Prabhu and Dokania 2020).

On the other hand, regularization methods usually put constraints on the loss function to prevent purging knowledge of already learned patterns. Such approach is part of methods such as Synaptic Intelligence (SI) (Zenke, Poole, and Ganguli 2017), LwF (Li and Hoiem 2017), and EWC (Kirkpatrick et al. 2016). The authors in (Liang and Li 2024) have recently proposed loss decoupling (LODE) for task-agnostic continual learning. LODE separates the two objectives for new tasks by decoupling the loss, allowing the model to assign different weights for different objectives, and achieving a better trade-off between stability and plasticity. The work in (McDonnell et al. 2024) proposes a continual learning approach for pre-trained models. Since forgetting occurs during parameter updating, the authors exploit training-free random projectors and class-prototype accumulation. They inject a frozen random projection layer with nonlinear activation between the pre-trained model's feature representations and output head to capture interactions between features with expanded dimensionality, providing enhanced linear separability for class-prototype-based continual learning. Although regularization-based methods represent a viable way to realize continual learning, one major drawback is that introducing new tasks exacerbates the forgetting of previously learned tasks over time.

Architectural-based methods concentrate on the topology of the neural model trying to alter it or leverage the available capacity to prevent the model from forgetting. One of the most known methods is CWRStar (Lomonaco, Maltoni, and Pellegrini 2019) that freezes all weights except the last layer that is trained to accommodate new tasks. One of the key components of many architectural-based methods is pruning that allows to remove weights less important to already learned tasks and reuse them to accommodate new knowledge. For example, PackNet (Mallya and Lazebnik 2017) leverages pruning to divide the neural network into independent sub-networks assigned for specific tasks. This type of approach is known as forget-free, as having independent sub-networks ensure full protection from forgetting. The DyTox algorithm (Douillard et al. 2022) is a quite novel approach based on the Transformer architecture, which shows efficient usage of transfer learning in continual learning process. However, its main disadvantage is that the same architecture with the same model capacity is used for all CL scenarios. Ada-QPacknet (Pietron et al. 2023) adopts compression techniques like pruning and quantization as a viable approach to deal with model capacity. However, despite its efficiency on a number of common CL scenarios, this method does not support weight sharing between tasks. Moreover, its pruning step is based on lottery ticket search without leveraging gradient mask optimization and post-training fine-tuning. Another potential drawback is the lack of a mechanism to measure and exploit differences in task distributions.

A number of recently proposed CL works can be regarded as hybrid, which often provide a mixture of regularization and architectural approaches, with memory-based features.

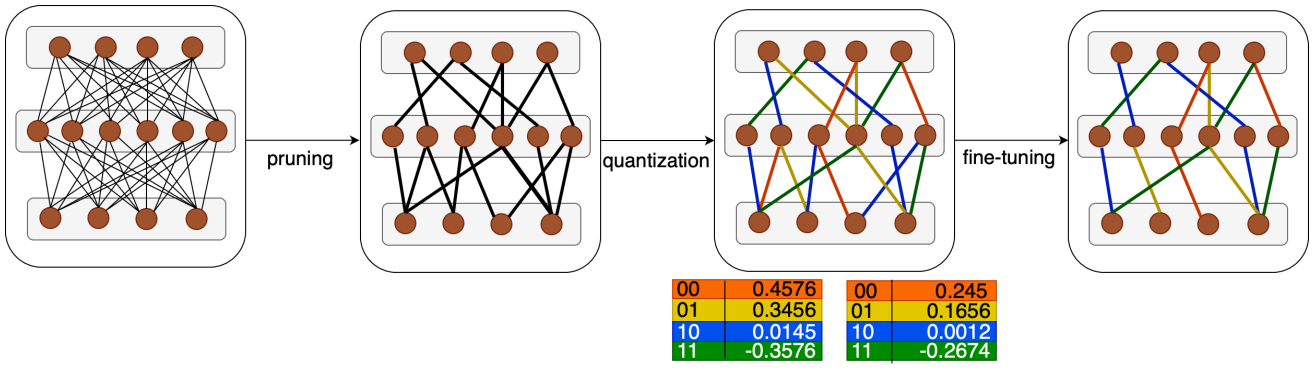


Figure 1: Model architecture evolution with TinySubNetworks (TSN). To incorporate a new task, the model-agnostic continual learning workflow involves pruning, non-linear quantization (different colors represent different clusters from the codebook), and fine-tuning (which involves additional weight pruning without re-training).

The work in (Jha et al. 2024) proposes an NP-based approach with task-specific modules arranged in a hierarchical latent variable model, in which regularizers on the learned latent distributions are adopted to alleviate forgetting. Uncertainty estimation is supported to handle the task head/module inference challenge. A different approach is taken in (Madaan et al. 2023) where, inspired by the knowledge distillation framework, the authors devise a setting where a weaker model takes the role of a teacher, while a new stronger architecture acts as a student. To deal with limited data availability, they propose Quick Deep Inversion (QDI) (Madaan et al. 2023) to enhance distillation by recovering prior task visual features that support knowledge transfer. However, this work does not discuss model capacity, making it difficult to gauge the trade-off between accuracy and memory requirements. A relevant method with memory-based features is Pro-KT (Li et al. 2024), a prompt-enhanced knowledge transfer model to open-world continual learning, i.e. learning on the fly with the goal of recognizing unknowns. The method is based on a prompt bank to encode and transfer task-generic and task-specific knowledge, and a task-aware open-set boundary to identify unknowns in the new tasks. Another example is Scaled Gradient Projection (SGP) (Saha and Roy 2023) , which combines orthogonal gradient projections with scaled gradient steps along the important gradient spaces for the past tasks. The authors scale gradients in these spaces based on their importance, and formulate an efficient way to compute it via singular value decomposition of input task representations.

Architectural and hybrid methods can be regarded among the most intriguing and sophisticated continual learning strategies. However, the major limitation of these methods appears when the capacity required to accommodate new tasks grows over time until exhaustion. Therefore, efficient capacity exploitation is of paramount importance and represents an open challenge in continual learning.

Method

Our TinySubNets continual learning strategy is a forget-free approach. The algorithm incorporates pruning and quantization with weight sharing, which allows us to drastically reduce model capacity. TinySubNets can be run in two configurations: with and without weight sharing. Weights sharing between tasks is implemented by introducing a mask that automatically selects weights from previous tasks for the current task during gradient descent learning. This capability has the advantage to minimize the bias associated with learning previous tasks. In the case of tasks with a significantly different distribution, the algorithm introduces the possibility of dividing quantized weights into memory banks.

The pruned model can be represented as $F^p_\Theta = (F_\Theta, M)$, where F_Θ is defined as:

$$F_\Theta(X) = f_{\theta_L}(f_{\theta_{L-1}}...(f_{\theta_0}(X))). \quad (1)$$

F_Θ is the initial full model with a set of convolutional and fully-connected layers f_{θ_i}, which are defined in the specified order. The Θ tensor is defined as:

$$\Theta = \{\theta_0, \theta_1, ..., \theta_L\}, \quad (2)$$

where Θ represents weights of all layers of the model, while θ_i contains weights from connections between layer i and $i+1$. $w_{i,k,l}$ denotes a single weight between k-th neuron from layer i and l-th neuron from layer $i+1$ (Figure 2). [1]

A key aspect of the method is the adoption of masks, which denote assignments of weights to specific tasks. All masks are stored in a tensor M:

$$M = \{M^0, M^1, ..., M^L\}. \quad (3)$$

[1]To simplify our notation, we assume that a single weight value is assigned to all tasks. In our method, multiple values for a single weight can be assigned to solve multiple tasks.

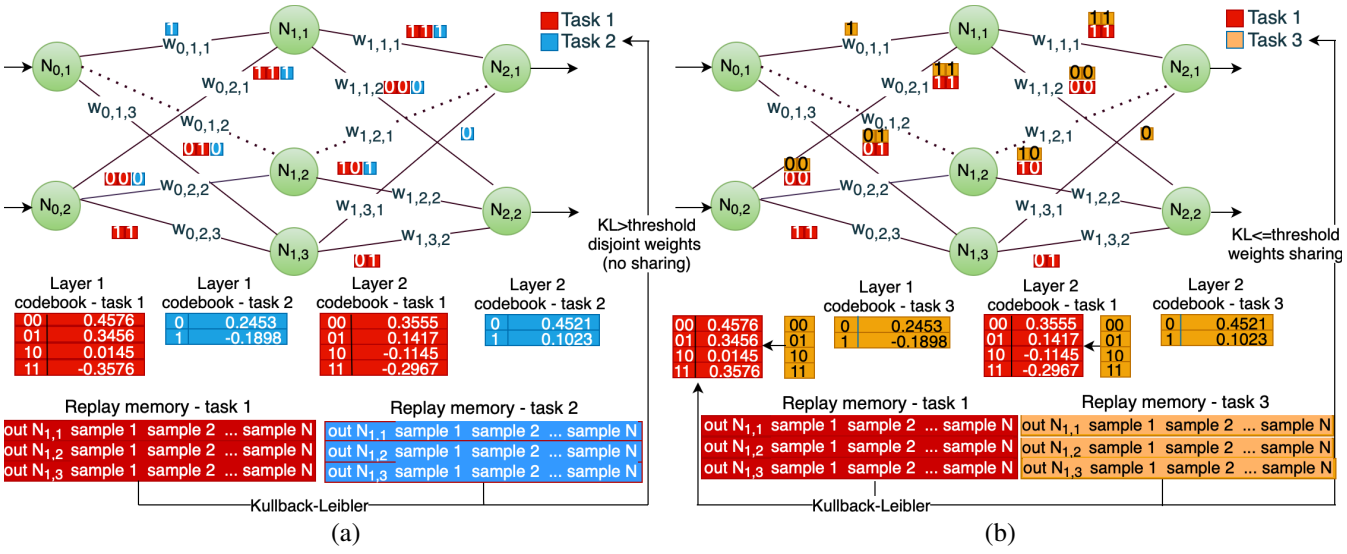


Figure 2: TinySubNetworks (TSN) - scenario with physical connection sharing with disjoint values using a reduced bit-width format where weight values are not shared between tasks (a), and with value-based weight sharing where overlapping weight values are shared between tasks (b). Different colors represent different tasks.

A single mask M^i corresponds to weights between layer i and $i+1$ and has the same size as θ_i. Each entry $M^i_{k,l}$ in mask M^i describes a set of tasks in which the weight between k-th neuron from layer i and l-th neuron from layer $i+1$ is leveraged. This single entry is defined as:

$$M^i_{k,l} = \{t_1, t_2, \ldots, t_n\}, \tag{4}$$

where t_p is a binary value indicating if weight is pruned or not by given task. The quantization generates bank, prefix, and key in the codebook to which a given weight is assigned for solving task p. Figure 2 showcases a graphical representation of this process. While the bank defines a subset of the 32-bit bandwidth of the weights, the prefix defines the specific codebook to which a weight is assigned in one bank.

Algorithm 1: TinySubNetworks (TSN) main algorithm

Require: $D = \{D_1, D_2, \ldots, D_T\}$ – training data for tasks $1, 2, \ldots, T$
Require: c – initial capacity per task
Require: Θ – model weights
Require: $M_0 = 0^{\Theta}$ – initial binary mask
Require: k – frequency (num. of batches) to trigger adaptive quantization
Require: ω – Kullback-Leibler threshold
Require: bs – batch size
Require: s_r – task replay memory size

1: Randomly initialized Θ and s
2: **for** $t = 0$ **to** T **do**
3: $R_t \leftarrow s_r$ from D_t to replay memory
4: $p \leftarrow$ -1
5: **for** $p = t - 1$ **to** 0 **do**
6: $kl \leftarrow$ compute $D_{KL(R_t, R_p)}$
7: **if** $kl < \omega$ **then**
8: set weight sharing of task t with task p
9: break
10: **for** $b = 0$ **to** $\frac{|D_t|}{bs}$ **do**
11: **if** $p >$ -1 **then**
12: Obtain mask M_t of the top-c% weights at each layer sharing with task p
13: **else**
14: Obtain new random mask M_t of the the separate weight memory bank
15: **if** $b \,\%\, k == 0$ **then**
16: ϕ_t, Θ, $K_t \leftarrow$ adaptive quantization $\Theta \odot M_t$ { See Appendix for Algorithm }
17: **else**
18: $\Theta \leftarrow \Theta \odot M_t$
19: compute $\mathcal{L}_{(x,y)}$
20: $\Theta \leftarrow \Theta$-$\eta \cdot (\frac{\delta \mathcal{L}_{x,y}}{\delta \Theta} \odot$ (1-M_{t-1}))
21: $s \leftarrow s$-$\eta \cdot (\frac{\delta \mathcal{L}_{x,y}}{\delta s})$
22: $M_t \leftarrow M_{t-1} \vee M_t$
23: Θ, $M_t \leftarrow$ fine tuned pruning {Algorithm 2}
24: $M \leftarrow M \cup M_t$
25: $K \leftarrow K \cup K_t$
26: **return** Θ - weights, M_t - mask, K - tasks codebook

A visual representation of learned masks is shown in Figure 3. Two types of weight sharing are supported: connection sharing with disjoint values (a), where separate masks with different values are learned for each task, and value-based weight sharing (b), where the same weight value is used for different tasks.

The sparsity level Υ_i describes how many weights out of all weights available in layer i are not yet assigned to any task. It is defined as:

$$\Upsilon_i = \frac{|M^i| - \sum_{k,l}^{|M^i|}(|M^i_{k,l}| > 0)}{|M^i|} \tag{5}$$

It is also possible to define the overall weighted sparsity of the model:

$$\Upsilon = \sum_{i}^{L} |\theta_i| \cdot \Upsilon_i \tag{6}$$

Additionally, masks for all tasks are encoded by Huffman algorithm to compress theirs size. The same methodology is used in (Kang et al. 2022), see Appendix[2].

The TSN main workflow is described in Algorithm 1. The process starts with random initialization of values and score weights. In the following iterations, the model is trained with subsequent tasks. In the initial processing phase, the task is filled with replay memory i.e. a fixed number of input data samples for each class of a given task. Then, the divergence between the current task and all previous ones is calculated. We adopt the KL-divergence to measure the divergence between tasks, defined as:

$$D_{KL_{t_i,t_j}} = D_{KL(D_{t_i}, D_{t_j})}. \tag{7}$$

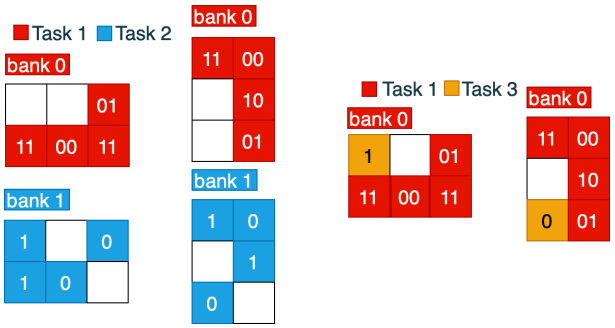
Specifically, a low KL-divergence implies that quantized weights may be shared between tasks. If the divergence is lower than the chosen threshold, the task with the minimum divergence is selected as the candidate for weight sharing. On the contrary, a high KL-divergence implies that tasks are significantly different. If the divergence is greater than the given threshold for all previous tasks, the current task allocates its own weight subspace by reserving a memory bank with a reduced number of bits (Lines 12-16).

Then, the training process begins. In the feed-forward process, weights are multiplied by the current mask. If a batch is a multiple of the k parameter, then the adaptive quantization algorithm is also executed (for additional details see Appendix). Then, the loss is computed, and the weights are updated:

$$\mathcal{L}_{(x,y)} = \frac{1}{|B|} \sum_{b=1}^{|B|} \left(\mathcal{L}_{ce}(f_\theta(x_b), y_b) + \alpha \sum_{i=0}^{L} \mathcal{L}_{mse}(\overline{f_{\theta_i}}, f_{\theta_i}) \right), \tag{8}$$

where $\mathcal{L}_{ce}$ is the conventional cross-entropy loss iteratively computed over batches B, and $\mathcal{L}_{mse}$ measures the differences in layer representations computed between the full model f_{θ_i} and the compressed model $\overline{f_{\theta_i}}$ which features pruning and quantization stages. After the learning process for a given task is completed, the global consolidated mask is updated. Next, pruning optimization is performed on the validation set. In the last step, the global mask and codebook are updated.

[2]https://arxiv.org/abs/2412.10869



(a) (b)

Figure 3: TinySubNetworks - mask with connection sharing with disjoint values (a) and value-based weight sharing (b). Each subfigure represents the weight matrix of the model: rows (input size of the layer) by columns (output size of the layer). White boxes represent pruned weights. Different colors represent weight allocations for multiple tasks assigned to each mask.

In Algorithm 2 the post-pruning without retraining process is presented. This process has a very limited computational cost (see execution times in Appendix), and its goal is to check the possibility to slightly increase the sparsity level if it does not impact accuracy, without triggering model retraining. The algorithm starts by setting the following values: weights multiplied by the input mask (mask achieved after training stage), the optimal mask (which is initially equal to the input mask), and the γ_{opt} variable, which constitutes the fitness function of the optimization process. Its value is obtained as the the sum of two components: the accuracy value multiplied by the α coefficient, and the sparsity value multiplied by the β coefficient (in all experiments α is set to 0.95 and β to 0.05).

Subsequently, the main loop of the algorithm is executed. In the next search iterations, the model layer is selected as a candidate for increasing sparsity. Then the mask is incremented (the number of zeros is increased). In the next stage, the weights are modified according to the changed mask. Accuracy is calculated for the changed weights and a new sparsity value is set. Based on them, the gamma value is calculated. If γ is greater than γ_{opt}, the modified mask becomes the current mask of the given layer and the γ value becomes γ_{opt}. Otherwise, the layer mask remains unchanged.

Moreover, in addition to the main TSN method described above, we also introduce TSN-wr (TSN without replay). In this case, weight sharing is based on pruning instead of replay buffer. While only not yet used weights are altered during learning a new task, the other weights are still used in the forward pass. When pruning weights after learning a new task, some of the weights already assigned to another task may still be assigned to a new task as well. The TSN-wr follows the same algorithm as TSN (Algorithm 1) except for lines 3-10, which are not included in TSN-wr execution. While, intuitively, this may lead to decreased performance in terms of accuracy, it should also allow to reduce the used memory due to lack of replay-buffer.

Algorithm 2: Fine tuned pruning for task t

Require: I – fine tuning iterations
Require: Θ – weights of the model
Require: M_t – current mask of the task
Require: Δ – quant of sparsity change
Require: A, Υ – initial accuracy and sparsity of the task
Require: α, β – scaling factors

1: $\Theta \leftarrow$ initialize the task weights by $\Theta \odot M_t$
2: $M_{opt} \leftarrow M_t$
3: $\gamma_{opt} \leftarrow \alpha \cdot A + \beta \cdot \Upsilon$
4: **for** $i = 0$ **to** I **do**
5: $l \leftarrow$ choose layer from 0 to L
6: $M' \leftarrow$ increment mask of layer l - M_l by Δ
7: $\Theta' = \Theta \odot M'$
8: $A' \leftarrow$ accuracy of $\Theta \odot M'$ on task t
9: $\Upsilon' \leftarrow$ compute sparsity for M' (eq: 6)
10: $\gamma \leftarrow \alpha \cdot A' + \beta \cdot \Upsilon'$
11: **if** $\gamma > \gamma_{opt}$ **then**
12: $M_{opt} \leftarrow M'$
13: $\gamma_{opt} \leftarrow \gamma$
14: **else**
15: $M_l \leftarrow$ decrement mask M_l of layer l by Δ
16: $\Theta' \leftarrow \Theta \odot M_{opt}$
17: **return** M_{opt}, Θ'

Results and Discussion

Datasets: Our main experiments were performed with three popular and commonly adopted CL scenarios: Permuted MNIST (p-MNIST) (Cun 1998), split CIFAR100 (s-CIFAR100) (Krizhevsky 2009), and 5 datasets (Ebrahimi et al. 2020), a task-incremental scenario consisting of MNIST, SVHN, FashionMNIST, CIFAR10, not-MNIST, TinyImagenet (Le and Yang 2015), and Imagenet100 (Russakovsky et al. 2015). The p-MNIST scenario consists of 10 tasks with randomly permuted pixels of the original MNIST (10 tasks with 10 classes each). s-CIFAR100 is divided into 10 tasks with 10 classes each. The 5 datasets scenario is a sequence of 5 tasks with different datasets, each with 10 classes. The TinyImagenet scenario consists of 40 tasks with 5 randomly sampled classes each. Finally, the Imagenet100 scenario consists of 10 tasks with 10 randomly sampled classes each.

Experimental Setup: The adopted models are a two-layer neural network with fully connected layers (p-MNIST), reduced AlexNet (s-CIFAR100) (Saha, Garg, and Roy 2020), Resnet-18 (5 datasets and Imagenet100), TinyNet (TinyImagenet) in accordance with model backbones used in the WSN paper (Kang et al. 2022). For weight initialization, we adopt the Xavier initializer. Experiments are executed on a workstation equipped with an NVIDIA A100 GPU. Our experiments involve 5 complete runs for each strategy. The training and testing execution times are reported in the Appendix[3]. The code of the method is available at the following public repository: https://github.com/lifelonglab/tinysubnets.

Metrics: We adopt the most standard definition of lifelong Accuracy following the description in (Díaz-Rodríguez

[3]https://arxiv.org/abs/2412.10869

et al. 2018). Moreover, we also compute the capacity (total memory occupied by each task) inspired by the capacity computation in (Kang et al. 2022). The capacity can be regarded as the sum of three components, and defined as follows:

$$CAP_t = \sum_{i}^{L}(1-\Upsilon_i)\cdot|\theta_i|\cdot b + |L|\cdot 2^b\cdot(32+b)+ \sum_{i}^{L}(1-\Upsilon_i)\cdot|M^i|$$

The first one describes the number of the weights after the pruning multiplied by the sum of bit-width, prefix and bank identifier of the codebook b. The second one describes the codebook size. The third one includes all masks' capacity.

Hyperparameters: The hyperparameters used in our experiments were set up as follows:

- **Adaptive learning rate** – for p-MNIST it starts from $3e-1$ and ends in $1e-4$, for CIFAR100 and 5 datasets it starts from $1e-2$ and ends with $1e-4$, for the rest of the scenarios is between $1e-3$ and $1e-5$
- **Number of epochs per task** – 200 epochs per task for each scenario
- **Batch size** – CIFAR100 - 2048, Imagenet100 - 32, 5 datasets, p-MNIST and TinyImagenet - 256
- **Fine tuning parameters** - 50 iterations for each scenario, α - 0.95, β - 0.95,
- **Kullback-Leibler threshold** - set empirically to have max two memory banks
- **Initial capacity per task** - 0.55 for CIFAR100, 0.5 for the rest of the scenarios
- **Frequency** (num. of batches) to trigger adaptive quantization - three times per epoch for each scenario
- **Task replay memory size** - 50 samples per task (only 15 samples per task for TinyImagenet),
- **Quantity of sparsity change** - 0.01 for each continual learning benchmark

The SGD optimizer was used for p-MNIST dataset. For the other scenarios the ADAM optimizer was adopted.

Comparative Studies

Table 2 presents the results of our experiments in terms of accuracy for two TSN variants: TSN without replay memory (TSN-wr) and TSN mixed (TSN with replay and fine tuning). We also present the results of the most popular standard CL methods and upper-bounds (Naive, CWRStar, SI, Replay, Cumulative), as well as three forget-free architectural methods: Packnet, WSN, and Ada-QPacknet.

Moreover, we also present capacity comparison for pruning-based methods in Table 1 where results for TSN without replay and fine tuning (TSN-wr-wpp) version are added. Additional results such as backward, forward transfer, sparsity levels, bit-widths and masks compression ratios achieved by Huffman algorithm are reported in the external Appendix.

	p-MNIST	s-CIFAR100	5 datasets	Tiny Imagenet
Packnet	96.38%	81.0%	82.86%	188.67%
WSN	77.73%	99.13%	86.10%	48.65%
Ada-QPacknet	81.25%	78.6%	33.7%	112.5%
TSN-wr	**22.65%**	**17.62%**	**24.68%**	**32.15%**
TSN-wr-wpp	23.41%	18.75%	30.62%	36.33%
TSN	37.5%*	41.92%	40.08%	93.6%

Table 1: Capacity comparison for pruning-based methods (* - in case of p-MNIST two memory banks without replay memory).

Accuracy-Capacity Tradeoff: As we can observe, TSN-wr allows us to significantly reduce capacity in comparison to other pruning methods (22.65% vs 77.73% for p-MNIST, 17.62% vs 78.6% for s-CIFAR100, 24.68% vs 33.7% for 5 datasets, and 32.15% vs 48.65% for TinyImagenet).

At the same time, TSN-wr presents an accuracy that is just slightly worse than the best-performing method (96.63% vs 97.14% for p-MNIST, 75.21% vs 77.27% for s-CIFAR100, 91.8% vs 94.1% for 5 datasets, and 79.81% vs 80.10% for TinyImagenet). Results obtained for the TSN method were obtained setting the Kullbach-Leibler divergence threshold empirically to have two memory banks at most, in order to balance memory consumption and accuracy. It is worth mentioning that Algorithm 2 resulted, on average, in an improvement from 1% (p-MNIST) to 6% (5 datasets) in capacity utilization (TSN-wr-wpp), with a negligible drop in accuracy (less than 1%, see Appendix). Overall, it is possible to observe that TSN-wr provides the best trade-off between accuracy and capacity. On the other hand, TSN achieves SOTA results on p-MNIST (97.14%), and TinyImageNet (80.10%), and s-CIFAR100 (77.27%), at the cost of a higher capacity when compared to TSN-wr (93.6% vs 32.15 on TinyImagenet and 37.5% vs 22.65% on p-MNIST and 41.92% vs 17.62% on s-CIFAR100).

Impact of Weight Sharing: To assess the impact of weight sharing, it is relevant to compare our approach which involves weight sharing, with architectural strategies without weight sharing, such as Ada-QPacknet. Our results show that weight sharing is beneficial in scenarios with low task heterogeneity, where task similarity can be profitably exploited (e.g. p-MNIST; s-CIFAR100). Quite interestingly, the situation is different in scenarios where tasks are more heterogeneous, such as 5 datasets. In this scenario, strategies without weight sharing achieve the best accuracy (94.1%), a 2.3% improvement over the replay variant of TSN. With TinyImagenet, we observe that TSN-wr outperforms Ada-QPacknet (79.81% vs. 71.9%). Despite the large number of classes in this dataset, we argue that TinyImagenet has a limited class representation, i.e. number of images per class, which makes the scenario less representative in terms of complexity than 5 datasets.

Architectural Strategies on ImageNet100: Results in Table 3 present several SOTA architectural and replay methods on the ImageNet100 scenario. DER has the highest parameter count at 112.27 million, significantly larger than the other methods. Despite its high complexity, DER achieved

an accuracy of 75.36%, which is slightly lower than DyTox. Ada-QPacknet has 11.5 million parameters, and achieved an accuracy of 72.26%, which is the lowest among the considered methods. Overall, we can observe that TSN-wr achieves the best results in terms of both parameters number and accuracy. With only 2.47 million parameters, TSN achieved a notable accuracy of 77.16%, which is particularly impressive, and suggests that TSN is highly efficient and effective. This result suggests that utilizing more capacity (as in DER or ADA-QPacknet) does not necessarily translate to a higher accuracy in complex scenarios.

Ablation Studies: We perform two ablation studies to investigate the memory capacity requirements for weights, codebooks, and masks across scenarios in weight-sharing mode. Our study suggests that while weights and masks are the primary consumers of memory, the use of codebooks introduces minimal additional memory requirements. The second ablation study evaluates the accuracy of models using different bit widths.

Our study shows that while reducing the bit width can help in compression, it often comes at the cost of accuracy. The impact varies by dataset, with larger and more complex datasets like TinyImagenet experiencing a more pronounced drop in accuracy at lower bit widths. For CIFAR100, a 4-bit width seems optimal, balancing efficiency and performance. This suggests that the choice of bit width should be carefully considered based on the specific dataset and accuracy requirements. Quantitative detailed experiments for the two ablation studies are reported in the Appendix. In all experiments variance in accuracy does not exceed 0.04. All experiments were run five times.

	p-MNIST	s-CIFAR100	5 datasets	TinyImagenet
Naive	60.12	17.32	33.08	20.27
CWRStar	31.31	20.84	36,16	24.00
SI	57.32	19.54	29.42	20.51
Replay	62.22	19.60	55.24	23.14
Cumulative	96.45	36.52	84.44	27.53
Packnet	96.31	72,57	92.59	55.46
WSN	96.41	76.38	93.41	71.96
AdaQPacknet	97.1	74.1	**94.1**	71.9
TSN-wr	96.63	75.21	91.80	79.81
TSN	**97.14**	**77.27**	93.76	**80.10**

Table 2: Comparative results in terms of average accuracy for all CL strategies with different CL scenarios (4-bit quantization): Permuted MNIST (p-MNIST), split-CIFAR100 (s-CIFAR100), and 5 datasets (5 trials, std < 0.002).

Methods	Parameters	Accuracy
DyTox	10.73M	75.54
DER	112.27M	75.36
Ada-QPacknet	11.5M	72.26
TSN-wr	**2.47M**	77.16
TSN	4.94M	**77.86**

Table 3: Comparison with SOTA architectural and replay methods on Imagenet100 (10/10) (5 trials, std < 0.002).

Energy Efficiency: An important aspect of our method is its limited energy usage. This aspect is a key factor for model deployment in low-resourced devices such as mobile phones and IoT, and it has been recognized as crucial for Green AI (Schwartz et al. 2020), which promotes environmentally sustainable models with low carbon footprint. To demonstrate this capability, we compute the theoretical complexity of our proposed method in terms of floating point operations per second (FLOPS) with different quantization levels.

To this end, FLOPS calculations for GPU is carried out via arithmetic multiplication based on the size and number of convolutional and fully connected layers in model architectures. Results in Figure 4 show that 16-bit and 8-bit quantized models provide a significant reduction in terms of FLOPS. This is a remarkable result considering the negligible drop in accuracy presented by compressed models. [4].

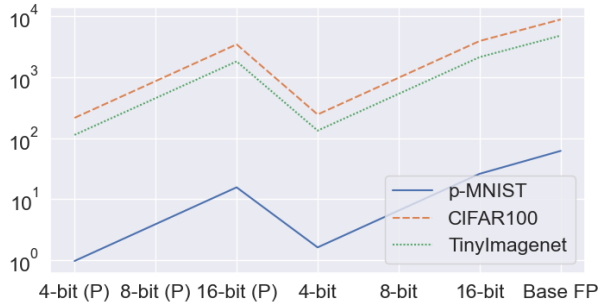


Figure 4: FLOPS with different quantization levels in different scenarios. P denotes the method with pruning. Baseline FP denotes the full precision of weights (no compression).

Conclusions

This paper introduces the TinySubNets (TSN) method, a forget-free continual learning strategy that provides an effective trade-off between model performance and capacity exploitation. TSN ensures this by adapting model architecture through pruning, adaptive quantization, and fine-tuning. Moreover, TSN is a model-agnostic approach and can be adapted to support any neural network architecture. Our extensive experimental evaluation includes the most popular continual learning benchmarks and models, as well as multiple other forget-free and architectural methods. We observe that TSN is able to significantly reduce the used model capacity while keeping a performance similar (or even better) to its competitors. This reduction may be significant in extending the longevity of the models, allowing them to learn more tasks. In future work, we will attempt to showcase the benefits of quantization performance while using specialized hardware accelerators. Moreover, we will also investigate the impact of decreasing the network size on the performance of architectural continual learning methods.

[4]The number of FLOPS and accuracy for each bit-width are presented in our external Appendix: https://arxiv.org/abs/2412.10869

Acknowledgments

We acknowledge the support of American University's National Science Foundation (NSF) funded ADVANCE AU Career Development Mini-Grants, the Polish Ministry of Science and Higher Education, and AGH University (program "Excellence initiative - research university").

References

Ameya Prabhu, P. H. S. T.; and Dokania, P. K. 2020. GDumb: A Simple Approach that Questions Our Progress in Continual Learning. *Lecture Notes in Computer Science (LNIP)*, 12347.

Baker, M. M.; New, A.; Aguilar-Simon, M.; Al-Halah, Z.; Arnold, S. M.; Ben-Iwhiwhu, E.; Brna, A. P.; Brooks, E.; Brown, R. C.; Daniels, Z.; Daram, A.; Delattre, F.; Dellana, R.; Eaton, E.; Fu, H.; Grauman, K.; Hostetler, J.; Iqbal, S.; Kent, C.; Ketz, N.; and et al., S. K. 2023. A domain-agnostic approach for characterization of lifelong learning systems. *Neural Networks*, 160: 274–296.

Chaudhry, A.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2019. Efficient Lifelong Learning with A-GEM. *Salk Institute for Biological Studies*, arXiv:1812.00420.

Cun, Y. L. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

David Lopez-Paz, M. R. 2017. Gradient Episodic Memory for Continual Learning. *arXiv*, https://arxiv.org/abs/1706.08840.

Díaz-Rodríguez, N.; Lomonaco, V.; Filliat, D.; and Maltoni, D. 2018. Don't forget, there is more than forgetting: new metrics for Continual Learning. *arXiv preprint arXiv:1810.13166*.

Douillard, A.; Ramé, A.; Couairon, G.; and Cord, M. 2022. Dytox: Transformers for continual learning with dynamic token expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9285–9295.

Ebrahimi, S.; Meier, F.; Calandra, R.; Darrell, T.; and Rohrbach, M. 2020. Adversarial continual learning. In *European Conference on Computer Vision*, 386–402. Springer.

Faber, K.; Corizzo, R.; Sniezynski, B.; and Japkowicz, N. 2023. VLAD: Task-agnostic VAE-based lifelong anomaly detection. *Neural Networks*, 165: 248–273.

Jha, S.; Gong, D.; Zhao, H.; and Yao, L. 2024. NPCL: Neural Processes for Uncertainty-Aware Continual Learning. *Advances in Neural Information Processing Systems*, 36.

Kang, H.; Mina, R. J. L.; Rizky, S.; Madjid, H.; Yoon, J.; Hasegawa-Johnson, M.; Ju-Hwang, S.; and Yoo, C. D. 2022. Forget-free Continual Learning with Winning Subnetworks. *ICML*, x.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; Hassabis, D.; Clopath, C.; Kumaran, D.; and Hadsell, R. 2016. Overcoming catastrophic forgetting in neural networks. *arXiv*, https://arxiv.org/abs/1612.00796.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images.

Le, Y.; and Yang, X. 2015. Tiny imagenet visual recognition challenge. *Stanford Computer Vision Lab*.

Li, Y.; Yang, X.; Wang, H.; Wang, X.; and Li, T. 2024. Learning to Prompt Knowledge Transfer for Open-World Continual Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 13700–13708.

Li, Z.; and Hoiem, D. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12): 2935–2947.

Liang, Y.-S.; and Li, W.-J. 2024. Loss decoupling for task-agnostic continual learning. *Advances in Neural Information Processing Systems*, 36.

Lomonaco, V.; Maltoni, D.; and Pellegrini, L. 2019. Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches. *1st Workshop on Continual Learning in Computer Vision at CVPR2020*.

Madaan, D.; Yin, H.; Byeon, W.; Kautz, J.; and Molchanov, P. 2023. Heterogeneous continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 15985–15995.

Mallya, A.; and Lazebnik, S. 2017. PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning. *arXiv*, https://arxiv.org/abs/1711.05769.

McDonnell, M. D.; Gong, D.; Parvaneh, A.; Abbasnejad, E.; and van den Hengel, A. 2024. Ranpac: Random projections and pre-trained models for continual learning. *Advances in Neural Information Processing Systems*, 36.

Parisi, G. I.; Kemker, R.; Part, J. L.; Kanan, C.; and Wermter, S. 2019. Continual lifelong learning with neural networks: A review. *Neural networks*, 113: 54–71.

Pietron, M.; Zurek, D.; Faber, K.; and Corizzo, R. 2023. Ada-QPacknet – Multi-Task Forget-Free Continual Learning with Quantization Driven Adaptive Pruning. In *26th European Conference on Artificial Intelligence, ECAI 2023*, 1882–1889.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115: 211–252.

Saha, G.; Garg, I.; and Roy, K. 2020. Gradient Projection Memory for Continual Learning. In *International Conference on Learning Representations*.

Saha, G.; and Roy, K. 2023. Continual learning with scaled gradient projection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 9677–9685.

Schwartz, R.; Dodge, J.; Smith, N. A.; and Etzioni, O. 2020. Green ai. *Communications of the ACM*, 63(12): 54–63.

Wortsman, M.; Ramanujan, V.; Liu, R.; Kembhavi, A.; Rastegari, M.; Yosinski, J.; and Farhadi, A. 2020. Supermasks in Superposition. *arXiv*, https://arxiv.org/abs/2006.14769.

Xu, K.; D. Zhang, J. A.; Liu, L.; Liu, L.; and Wang, D. 2021. GenExp: Multi-objective pruning for deep neural network based on genetic algorithm. *Neurocomputing, April*.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual Learning Through Synaptic Intelligence. *arXiv*, https://arxiv.org/abs/1703.04200.

Zhou, D.-W.; Wang, Q.-W.; Qi, Z.-H.; Ye, H.-J.; Zhan, D.-C.; and Liu, Z. 2023. Deep class-incremental learning: A survey. *arXiv preprint arXiv:2302.03648*.