Overcoming Catastrophic Forgetting in Continuous Image Classification Tasks with Low-Rank Adapters

Everton Lima Aleixo[1] **and Juan G. Colonna**[1]

[1]**Institute of Computing (IComp), Federal University of Amazonas (UFAM), Brazil**

Corresponding author:
Everton Lima Aleixo[1]

Email address: everton.aleixo@icomp.ufam.edu.br

ABSTRACT

Current convolutional neural network (CNN) models excel at image classification tasks, often achieving performance comparable to or surpassing human capabilities. However, when these models are subjected to continuous learning scenarios–where new image classes are progressively added–their accuracy on previously learned classes tends to decrease drastically, a phenomenon known as catastrophic forgetting (CF). Existing techniques to mitigate CF often incur significant computational costs, manifesting as increased model sizes that demand more memory and hardware resources, or extended training times due to larger datasets and retraining efforts. To address this issue, we explore the use of Low-Rank Adapters, which applies the Low-Rank Adaptation (LoRA) technique to convolutional neural networks (CNN), to mitigate CF in continuous learning. Our approach effectively reduces CF while only minimally increasing the number of new parameters added to the CNN model. We evaluate this method on benchmark datasets commonly used in continuous learning setups, demonstrating that CF can be mitigated with less than a 15% increase in parameters for each new task in the convolutional layers, and approximately a 2% increase when considering the entire model compared to our baseline, while maintaining similar accuracy.

INTRODUCTION

Image classification tasks are used in many fields, e.g., anomaly detection in industry 4.0, crowd monitoring, autonomous driving (Kharitonov et al., 2022; Sharma et al., 2023; Zhang et al., 2023), etc. In the last decade, with the evolution of Convolutional Neural Networks (CNNs), most computer vision models have superpassed humans in many benchmarks regarding image recognition (Russakovsky et al., 2015). However, when it comes to continuous learning–where new classes are incrementally added and learned by a model already trained over time–its performance on previously learned classes begins to decline.

A CNN model M can be represented as a set of parameters $\Theta = \{\theta_1, \theta_2, \dots, \theta_n\}$, where each θ_i corresponds to one model's parameter with a specific value. These values are typically real numbers, and a particular configuration of Θ enables the model to achieve a desired performance level (e.g., at least 80% accuracy) on a given task. McCloskey and Cohen (1989) show that in continuous learning scenarios, a fundamental challenge arises when adapting the model to learn a new task, the updates to Θ may overwrite the critical values for previously learned tasks, leading to a phenomenon known as catastrophic forgetting (CF).

CNN models have millions of parameters (θ_i) and require access to a rich and balanced dataset, composed of diverse samples from all the classes they will need to predict throughout their lifespan. However, in real-world scenarios, this is not always possible. For example, when a model is trained to guide an autonomous driving car, it has a limited set of samples of its daily activities. Consequently, new situations, such as new road signs or significant changes in weather conditions, can lead to unexpected model outputs (Bang et al., 2021).

There are many real-world scenarios where enabling a CNN model to continuously learn and adapt is highly beneficial. A naive strategy to achieve continuous learning in CNNs is to retrain the entire model whenever new images from previously unseen classes become available. However, depending on the dataset's size and the number of image classes, this approach can become prohibitive over time due to significant computational and storage requirements. One major challenge is data storage: to effectively retrain the model, all previous data must be combined with the new data. If previous data are not included, the model may forget information about previously learned classes, leading to the phenomenon known as Catastrophic Forgetting (CF) (McCloskey and Cohen, 1989). This forgetting can significantly degrade the model's performance on previously learned tasks, undermining its overall effectiveness.

Traditional convolutional neural network (CNN) models such as VGG (Liu and Deng, 2015) and ResNet (He et al., 2016) face several obstacles to achieve continuous learning due to their limited, fixed number of parameters, which restricts their learning capacity. Since the learning capacity of a model depends on its size—specifically, the number of learnable parameters—it is inherently finite. Furthermore, as the amount of data and the number of classes increase, the model requires more parameters to learn effectively, causing it to grow in size and potentially become unmanageably large. This growth can lead to increased computational demands, longer training times, and greater storage requirements, making it impractical for continuous learning scenarios.

Aleixo et al. (2023) categorized the current approaches to mitigating Catastrophic Forgetting (CF) into five main groups: (i) rehearsal methods, which store and replay past data to retain knowledge but face challenges related to memory limitations and data privacy regulations(Rebuffi et al., 2017; Chaudhry et al., 2019); (ii) distance-based techniques, which rely on feature space distance metrics to maintain prior knowledge but often struggle with scalability (Pham et al., 2022); (iii) sub-network approaches, which allocate specific parameters to tasks, thereby limiting overall learning capacity due to fixed architecture sizes (Mallya et al., 2018); (iv) dynamic networks, which dynamically expand the model's architecture to accommodate new tasks, balancing learning capacity and scalability but at the cost of increased resource requirements (Rusu et al., 2016; Zacarias and Alexandre, 2018); and (v) hybrid methods, which combine multiple approaches to balance trade-offs, but often introduce added complexity (Kirkpatrick et al., 2017; Rebuffi et al., 2017). Despite the variety of these techniques, no single method has proven capable of fully addressing the multifaceted challenges posed by CF.

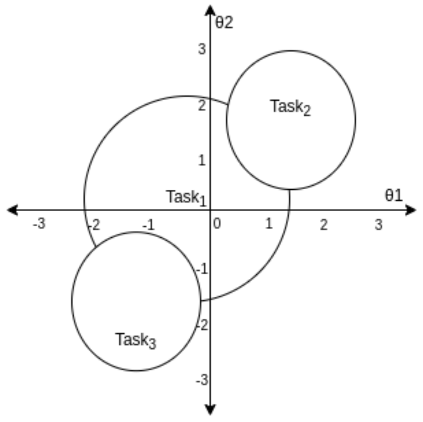
Among these categories, dynamic networks have shown particular promise and are most relevant to this study. These methods achieve higher accuracy compared to other approaches and effectively prevent knowledge loss in continual learning, as they ensure that previously learned tasks are not forgotten. However, dynamic networks face the significant drawback of model size growth, which can limit their scalability. For instance, Progressive Neural Networks (Rusu et al., 2016) duplicate the model size for each new task, while Sena-CNN (Zacarias and Alexandre, 2018) increases the model size by at least 50% per task, highlighting the resource demands of this category.

To illustrate the limitations of non-dynamic models with shared parameters across different tasks, consider a simple model M where the set of parameters is $|\Theta| = 2$, and each parameter $\theta_i \in [-3, 3]$, resulting in a total of $49 = 7^2$ possible distinct configurations of the model. Also assume the existence of three tasks, T_1, T_2, and T_3, each associated with specific regions in the (θ_1, θ_2) parameter space. Task T_1 is solvable by the model M when the parameter configuration lies within a circular region centered at $(0, 0)$ with a radius of 2. Task T_2 is solvable when the parameter configuration lies within a smaller circle centered at $(2, 2)$ with a radius of 1. Finally, Task T_3 is solvable when the parameter configuration falls within another circle centered at $(-2, -2)$ with a radius of 1.

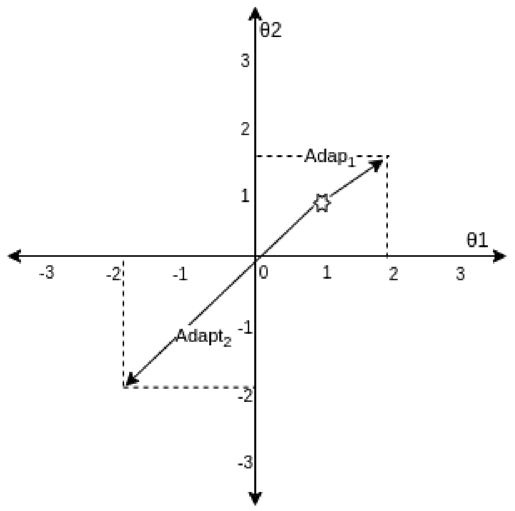
From Figure 1a, it is clear that the parameter configurations capable of solving all tasks simultaneously correspond to the intersection of their solution regions. However, no such intersection exists for the solution sets of T_1, T_2, and T_3. Consequently, the model M cannot achieve a single parameter configuration Θ that satisfies all tasks simultaneously. Even with access to the training data for all tasks and extensive training, the model must enhance its capacity to handle these three tasks.

One approach to address this limitation is to increase the number of parameters, such as introducing an additional parameter θ_3. Another approach is to use dynamic adapters that modify the parameter values on the fly based on the target task. Figure 1b illustrates this concept. The model, represented by the star point, solves T_1 in its original configuration. To solve T_2, it applies an adapter $(1, 1)$ to the parameters, and for T_3, it applies a different adapter $(-3, -3)$. The key challenge with adapters is to design them with minimal parameter overhead compared to simply adding new parameters.

Building on the strengths and challenges of dynamic networks discussed above, this paper introduces a methodology to eliminate Catastrophic Forgetting (CF), enabling models to retain knowledge of



(a) Task solution regions in the parameter space. Each circle represents the region of parameter configurations that solve one specific task (T_1, T_2, or T_3). No intersection exists between these regions, meaning no single configuration can solve all tasks.



(b) Dynamic adapter concept. The star point represents the model solving T_1. Adapters $(1,1)$ and $(-3,-3)$ are applied to solve T_2 and T_3, respectively.

Figure 1. Parameter configurations for solving tasks and the concept of dynamic adapters.

previously learned tasks while effectively learning new ones. Our approach applies Low-Rank Adaptation (LoRA)(Hu et al., 2022) and its convolutional adaptation, ConvLoRA(Aleem et al., 2024), which allow continual learning with minimal model expansion. We demonstrate that adapting convolutional layers can yield accuracy improvements of over 15%, emphasizing the importance of adapting the entire model when learning new tasks, rather than focusing solely on the final layers. This approach leverages the efficiency of LoRA and ConvLoRA, which add less than 2% additional trainable parameters, making them ideal for continual learning scenarios.

By applying LoRA and ConvLoRA for continuous learning, we effectively mitigate accuracy degradation that is typically observed in continuous learning scenarios, providing a robust solution to handle CF. As a result, forgetting metrics such as those proposed by (Kemker et al., 2018) become unnecessary, since forgetting is effectively eliminated. The focus of our experiments lies in balancing the trade-off between model size increase and accuracy improvements. We summarize the main contributions of this work as follows:

- **Reduction in model size increase**: the use of adapters achieves a significant reduction in model size increase, adding approximately 2% in trainable parameters to the original VGG19 model;
- **Elimination of old data storage requirement**: our approach removes the necessity to store previous data, a common requirement in many existing techniques addressing CF, thereby enhancing scalability and compliance with data privacy constraints;
- **Preservation of accuracy**: the modularity of LoRA and ConvLoRA ensures that accuracy on old tasks is preserved, with our approach empirically outperforming baseline methods in accuracy for continual learning scenarios.

The rest of the paper is organized as follows. Background section provides an overview of current CF techniques and their limitations, with a focus on dynamic networks. The meteorology section details the design and implementation of adapters and experiments. Results section depicts a discussion about the results, demonstrating the effectiveness of adapters. Finally, the Conclusions and Future Works section offers final considerations and suggests future research directions.

BACKGROUND

CNN architectures are predominantly made up of convolutional layers (LeCun et al., 2015). The convolutional layers are the fundamental components to extract hierarchical features from the input images.

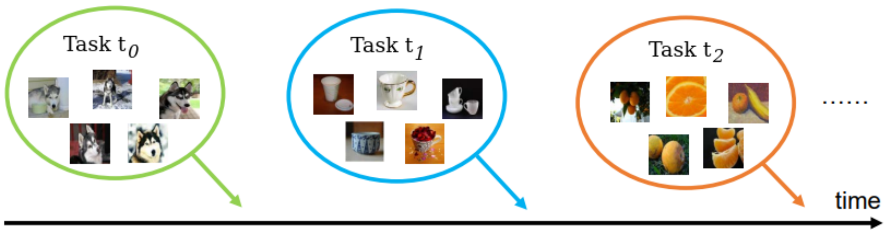


Figure 2. In a continuous learning scenario, at each time step, the model is tasked with acquiring knowledge about new classes while maintaining its proficiency in accurately classifying samples from prior instances. This dynamic process ensures continual adaptation without compromising the model's ability to correctly classify previously encountered patterns.

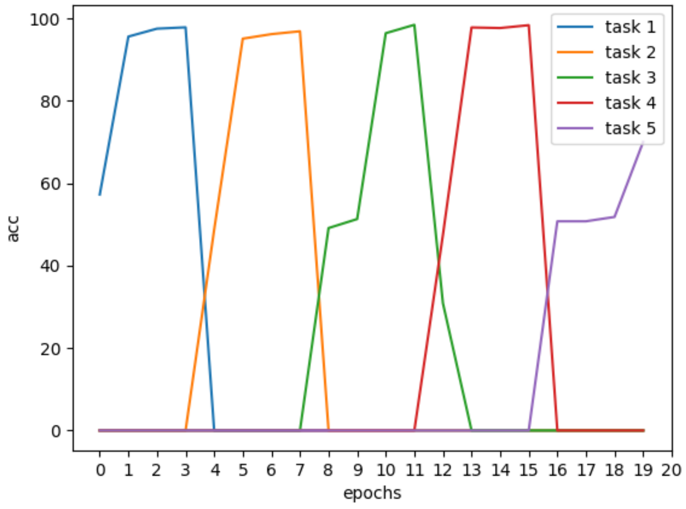


Figure 3. Effects of CF in a CNN trained on the SplitMNIST dataset, where the model has to classify two classes per task. The graph shows how the accuracy evolves over 20 epochs. The CNN performs well on the last task but completely forgets previous tasks, demonstrating CF.

These layers employ kernel matrices to systematically transform the input, enabling the network to detect patterns such as edges, textures, and more complex forms (Krizhevsky et al., 2012; Goodfellow et al., 2016).

In a continuous learning scenario, a CNN has to incrementally learn new tasks $\{t_i, t_{i+1}, \dots\}$ composed of different image classes, where each new task (*e.g.* t_i) is learned without access to samples from past tasks $\{\dots, t_{i-2}, t_{i-1}\}$, as shown in Figure 2. Because CNNs update their parameters W through gradient descent, they are particularly prone to forgetting learned patterns when their parameters are updated to recognize images from the new incoming task (French, 1999). Consequently, during training, when a model updates its parameters, the accuracy on previously learned tasks can drastically decrease.

Figure 3 depicts how this forgetting effect negatively impacts the accuracy of a small CNN composed of four convolutional blocks with ReLU activation and normalization layers, followed by max pooling, and ending with a single fully-connected layer. This small CNN was trained on the SplitMNIST dataset. Note that in this example, each task involves classifying two different digits from the MNIST dataset. For instance, Task 1 could involve learning to classify digits '1' and '6', while Task 2 could involve classifying digits '0' and '5'.

A recent survey organized methods aimed at handling CF into five main categories: (i) rehearsal; (ii) distance-based; (iii) sub-networks; (iv) dynamic networks; and (v) hybrid methods (Aleixo et al., 2023). Our approach falls within the dynamic networks category, as we address CF by progressively expanding the model by adding adapters. Currently, only methods from the dynamic networks and sub-networks categories have shown promising practical results in mitigating CF. However, sub-network methods face resource constraints because the number of model parameters is kept constant, limiting their learning capacity. This limitation underscores our advocacy for the utilization of dynamic networks as the optimal strategy for continuous learning in lifelong settings.

Dynamic Networks

Currently, the most promising approaches to solving or mitigating CF in neural networks belong to the dynamic networks category. A pioneering work in this category is Progressive Neural Networks (PNN) (Rusu et al., 2016), which adopts a progressive growth strategy by adding new sets of parameters each time a new task is introduced. Unlike other methods, PNN explicitly creates task-specific modules for each new task, consisting of new sets of trainable parameters that are integrated into the existing network architecture. Importantly, the parameters from previously learned tasks are kept frozen to prevent forgetting. This strategy allows PNN to retain knowledge from previous tasks while isolating task-specific learning to the new modules, avoiding interference between tasks. Unlike techniques such as LoRA, which add trainable parameters in a compact and efficient way, PNN results in significant model expansion, duplicating the size of the model for each new task.

The integration of these modules facilitates the transfer of knowledge from prior tasks during training on subsequent tasks, preventing the loss of previously acquired information. This incremental expansion mechanism try to empower the network to accumulate knowledge over time, ensuring the seamless integration of new tasks without compromising performance on earlier ones. However, this comes at the cost of increased model size, which implies greater computational requirements.

Sena-CNN, as introduced by Zacarias *et al. (Zacarias and Alexandre, 2018)*, employs a dynamic approach to expand the network architecture. Unlike Progressive Neural Networks (PNN), Sena-CNN preserves the first 50% of the network layers frozen after learning the initial task, maintaining the general feature extraction capabilities. For each new task, the final 50% of the layers are replicated and trained independently, creating task-specific modules. These new layers are then integrated into the model, ensuring that knowledge from earlier tasks is preserved while enabling adaptability to new tasks. This strategy effectively isolates the learning of new tasks but results in a trade-off: while it prevents interference with previously learned knowledge, it significantly increases the model size as new modules are added for each task.

Figure 4 summarizes how new modules are added in PNN and Sena-CNN when a new task is introduced. In PNN (Figure 4a), the entire network is duplicated, freezing previously learned parameters to prevent forgetting, while new layers are added for task-specific learning. In contrast, Sena-CNN (Figure 4b) freezes the initial layers to retain general features and replicates only the final layers for task-specific learning, reducing parameter redundancy compared to PNN.

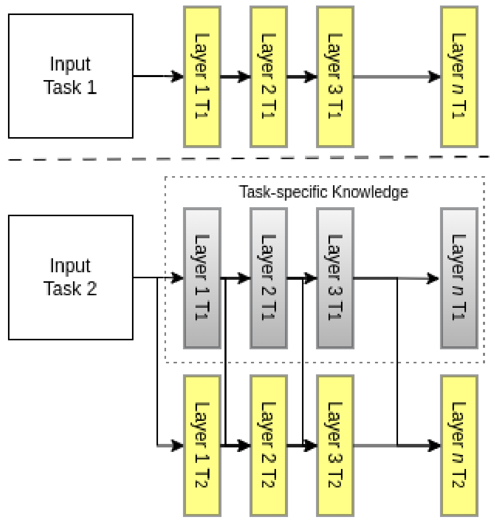
Regarding the identification of tasks during inference, all presented methodologies possess the capability to discern the task to which a given sample belongs beforehand. This feature ensures that the right set of model parameters will be chosen during deployment, contributing to the overall versatility and applicability of these continuous learning approaches.

As we have observed, there are few proposals in the dynamic network category, and none of them employ the technique we present in the following sections. In the subsequent sections, we introduce our approach to incrementally learning new tasks in image classification, with a specific focus on minimizing the number of new parameters required to adapt an already trained model, thus reducing resource requirements. Our method is designed to improve efficiency, modularity and parameter reuse.

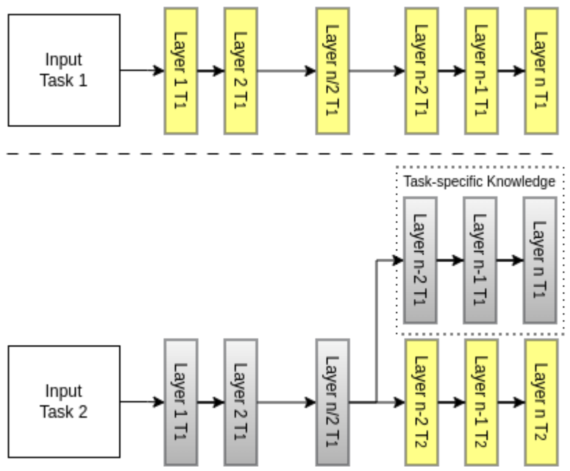
Adapters

Adapters are modules designed to facilitate the fine-tuning of large models without significantly increasing the number of parameters. These modules are inserted into the existing architecture of a neural network, enabling it to learn new tasks more efficiently. Instead of retraining the entire model, adapters focus on optimizing smaller, task-specific parameters, which are added to certain layers. This approach helps preserve the knowledge the model has acquired from previous tasks while allowing it to learn new tasks with minimal computational cost. This targeted adaptation enhances the model's flexibility, enabling it to handle multiple tasks with minimal overhead, while maintaining strong performance across all learned tasks.

Low Rank Adaptation (LoRA) is an adapter specifically designed to significantly decrease the number of trainable parameters in Large Language Models (LLMs) (Liu et al., 2020; Brown et al., 2020). LoRA



(a) PNN progressively adds new layers for each task while freezing previously learned parameters, resulting in duplicated modules for new tasks.



(b) Sena-CNN preserves the initial layers frozen to retain general features and replicates only the final layers for task-specific learning. This approach reduces parameter redundancy compared to PNN.

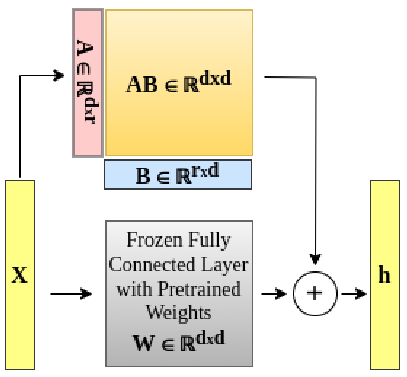
Figure 4. Illustration of PNN and Sena-CNN architectures when learning new tasks. In both figures, yellow modules represent layers with trainable parameters for the current task, while gray modules correspond to frozen layers that preserve previously learned knowledge. The dotted square highlights the task-specific knowledge retained from prior tasks. The k value after 'T' indicates the task number in which the corresponding layer was trained.

accomplishes this by introducing a scaled-down set of new weights into the model, where only these specific weights undergo the training process. These adapters are inserted into all fully connected layers, contributing to a more streamlined and resource-efficient fine-tuning process for large LLMs (Hu et al., 2022).

The LoRA method freezes the pretrained weights matrix, $W \in \mathbb{R}^{d\times d}$, of a layers and focuses on training two smaller matrices, A and B. Despite the significantly reduced size of A and B compared to W, the dot product AB produces a new matrix with the same dimensions as W, enabling the element-wise addition $W+AB$, which effectively generates new parameter values. Figure 5a depicts the LoRA training approach. Essentially, the LoRA procedure can be extended to tensors with an arbitrary number of dimensions, not limited to two dimensions. In Aleem et al. (2024), a simple but effective alternative inspired by LoRA, called ConvLoRA, was proposed, where the product of matrices A and B is reshaped to fit the dimensions of the weights in any convolutional layer, effectively reducing the number of parameters to be trained for each new task. Figure 5b depicts the ConvLoRA method.

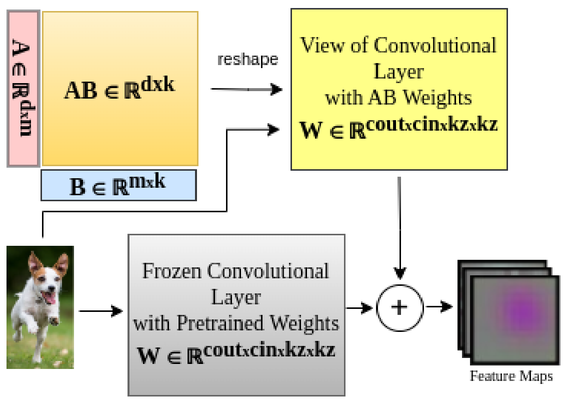
A convolutional layer is composed of a set of **kernels** (or filters) that perform convolution operations over the input to generate feature maps. Each kernel has dimensions $kz \times kz \times c_{in}$, where kz is the spatial size of the kernel (height and width), and c_{in} represents the number of input channels. For example, in an RGB image, c_{in} would be 3, corresponding to the three color channels. A convolutional layer applies c_{out} kernels, where c_{out} determines the number of output feature maps (also referred to as output channels). Thus, a convolutional layer can be mathematically represented as a 4-dimensional weight matrix $W \in \mathbb{R}^{c_{out}\times c_{in}\times kz\times kz}$, where c_{out} is the number of output channels or kernels, c_{in} is the number of input channels, kz is the spatial size of the filters, and W contains the weights for all kernels in the layer. This representation formalizes how each kernel connects the input channels to produce output feature maps, allowing the convolutional layer to extract relevant patterns, such as edges, textures, or more complex features, from the input data.

In LoRA, the dimension d matches the original dimension of the frozen fully connected layer, while the dimension r is defined by the network's architect. The parameter r controls the number of additional trainable parameters introduced to the model. In ConvLoRA, however, all dimensions of matrices A and B must be adjusted to fit the frozen convolutional layer. Like LoRA, ConvLoRA also uses the parameter r



$A \in \mathbb{R}^{dxr}$
$AB \in \mathbb{R}^{dxd}$
$B \in \mathbb{R}^{rxd}$
X
Frozen Fully Connected Layer with Pretrained Weights
$W \in \mathbb{R}^{dxd}$
h

(a) The LoRA method involves freezing the pretrained weights W of a layer and learning two vectors, A and B. The vector A is initialized using a normal random distribution $\mathcal{N} \sim (0, \sigma^2)$, while B is initialized as 0. These vectors are considerably smaller in size compared to W.



$A \in \mathbb{R}^{dxm}$
$AB \in \mathbb{R}^{dxk}$
$B \in \mathbb{R}^{mxk}$
reshape
View of Convolutional Layer with AB Weights
$W \in \mathbb{R}^{coutxcinxkzxkz}$
Frozen Convolutional Layer with Pretrained Weights
$W \in \mathbb{R}^{coutxcinxkzxkz}$
Feature Maps

(b) The ConvLoRA matrix is produced using the dot product of two trainable small vectors A and B then reshape it to have the same dimensions as W. The initializations are the same of LoRA. The values d, m and k are calculated based on layer hyper-parameters and r informed by the architect.

Figure 5. Architecture of adapters LoRA and ConvLoRA to train a new task.

to manage the number of new trainable parameters, which is manually set. In ConvLoRA, the value of d is calculated as the product of input channels and kernel size, $d = c_{in} \times \text{kernel\_size}$, while k represents the product of output channels and kernel size, $k = c_{out} \times \text{kernel\_size}$. Finally, m is computed as the product of r and kernel size, $m = r \times \text{kernel\_size}$.

For instance, consider a ConvLoRA setup where both c_{in} and c_{out} are set to 512, with a kernel size of 3. In the original convolutional layer, the number of parameters is calculated as $512 \times 512 \times 3 \times 3 = 2,359,296$. In ConvLoRA, matrices A and B introduce a new set of parameters with dimensions defined by r. If we set $r = 16$, matrix A would have dimensions $(512 \times 3) \times (16 * 3)$ and matrix B would have dimensions $(16*3) \times (512 \times 3)$, resulting in a total of $(512 \times 3) \times (16*3) + (16*3) \times (512 \times 3) = 147,456$ new parameters. Thus, by using ConvLoRA with $r = 16$, we achieve a significant reduction in the number of trainable parameters, from the original 2,359,296 to only 147,456. This result in an adapter with approximately 93.7% less parameters, greatly decreasing the memory and computational costs while training but maintaining the layer's capacity to learn new features.

As the number of added adapters increases, the model gains the capacity to learn more tasks. Therefore, if the tested task is known beforehand, the appropriate adapter for the given task can be chosen, making the network capable of classifying any previously learned task. Importantly, the ConvLoRA layer serves as a versatile module for a convolutional layer in various architectures, such as VGG or ResNet. We use VGG19 in our experiments because it has sufficient capacity to handle the CIFAR-100 and CUB-200 datasets, while also being straightforward to analyze the effect of adapters, avoiding the complexity of more advanced architectures like ResNet with residual connections.

METODOLOGY

In our experiments, we compare the usage of LoRA adapters against SenaCNN (Zacarias and Alexandre, 2018). SenaCNN incorporates a new set of layers dedicated to each new task (Figure 4b). As discussed previously, such dynamic methods increase the model size as a side effect. However, the adapter adds fewer parameters compared to SenaCNN. As an advantage, both methods maintain the accuracy of previously learned tasks without storing past data or retraining the original model using such data.

For evaluation, we consider two metrics: (1) accuracy, to ensure that our method can learn the tasks as well as the SenaCNN baseline; and (2) the amount of new parameters (resources) added, which are necessary to learn each new task.

Our evaluation was conducted using standard benchmarks for CF. We utilized the Avalanche framework to conduct our tests, which conveniently provides two pre-processed datasets, CIFAR-100 and CUB-

200 (Lomonaco et al., 2021). The CIFAR-100 consists of 100 classes, divided into ten tasks with 10 classes each. CUB-200 consists of 200 classes of birds split into 5 tasks: the first task with 100 classes and the remaining four tasks with 25 classes each. In CUB-200, we have to apply a normalization transformation to ensure that all images have the same dimensions (128x128x3).

We demonstrate the efficacy of adapters using the VGG-19 architecture due to its capacity to effectively learn both datasets without the complexities introduced by larger models. Without loss of generality, our approach can be extended to other convolutional architectures. We train each task until convergence using early stopping with a patience of five epochs. We use the Adam optimizer with a learning rate of 1×10^{-5} and a weight decay of 5×10^{-4}, which proved to be effective in convergence and regularization. The batch sizes are 64 for CIFAR-100 and 8 for CUB-200 due to computational constraints. Due to the Adaptive Average Pooling layer in VGG-19's feature extractor, reproducibility is challenging. Therefore, we run each experiment eight times and present the mean and variance of the results. Our code implementation uses PyTorch[1].

We first selected 100 classes from the CUB-200 dataset and evaluated them using a VGG19 model pre-trained on ImageNet. The classification head–the final layer–was replaced to accommodate the new 100 classes. Without fine-tuning, this model barely achieves an approximately 2% accuracy, likely attributable to random chance.

We tested four fine-tuning setups. The first involved retraining all layers (convolutional, fully connected, and batch normalization layers). We called this setup **fine-tuned upper bound** because it represents the highest accuracy the model can achieve. Second, we froze all layers except the batch normalization layers and the classification layer; we referred to this as **finetune-1**. Third, we froze all layers except the batch normalization layers and the last two fully connected layers; we referred to this as **finetune-2**. Finally, we froze all layers except the batch normalization layers and the last three fully connected layers (which are all the fully connected layers of VGG19); we referred to this as **finetune-3**.

These setups were designed to demonstrate that traditional fine-tuning techniques are insufficient to fully harness the potential of a neural network architecture in image classification tasks, underscoring the crucial role of convolutional layers in achieving higher accuracy. While fine-tuning all layers is generally avoided due to computational cost, adapters can enable layer-wise adaptation without a significant increase in computational costs.

To calibrate the chosen r value for continuous learning evaluation, we experimented with the set of values $1, 3, 6, 9, 12, 24$ to identify the lowest possible value that maintains accuracy close to that of the fine-tuned upper bound while minimizing the number of additional parameters added. For consistency, we froze all layers except the batch normalization layers and the last three fully connected layers. These setups are referred to as **convlora-*r***, where r represents the selected value.

Once the optimal r value for ConvLoRA has been identified, we proceeded with three setups to evaluate the application of LoRA in fully connected layers, given their high resource demand. In the first setup, **fullyLoRA-1**, we trained only the ConvLoRA filters and the last classification layer, which was randomly initialized. In the second setup, **fullyLoRA-2**, LoRA with $r = 30$ was applied to only the last two fully connected layers. Finally, in the third setup, **fullyLoRA-3**, LoRA with $r = 30$ was applied to all three fully connected layers. These setups aim to investigate if the r values chosen for fully connected layers differ from the optimal r value used in ConvLoRA, emphasizing the unique resource requirements of each layer type.

Finally, it is worth mentioning that Batch Normalization layers pose challenges in continuous learning because they store the statistics of the dataset related to the current task (Pham et al., 2022). Therefore, we decided to make copies of these layers, creating a new layer for each newly learned task. This approach does not cause a considerable overhead, as these layers have very few parameters compared to other layers.

RESULTS

In this section, we present the results of experiments carried out to determine the optimal r values for LoRA and ConvLoRA, along with a comparison to benchmark performance.

[1]The code used in this study is publicly available at `https://github.com/evertonaleixo/effects-of-lora-on-catastrophic-forgetting`.

Table 1. Comparison of transfer learning strategies to learn a new task.

Name	Trainable Parameters	Accuracy
fine-tuned upper bound	≈140,000,000	0.77
finetune-3	120,365,256	0.61
finetune-2	17,600,712	0.59
finetune-1	819,400	0.58

Table 2. ConvLoRA configurations: trainable parameters and accuracy.

Name	r	Trainable Parameters	Accuracy
convlora-1	1	94,491	0.65
convlora-3	3	283,473	0.74
convlora-6	6	566,946	0.73
convlora-9	9	850,419	0.76
convlora-12	12	1,133,892	0.76
convlora-24	24	2,267,784	0.76

Parameter Selection for LoRA and ConvLoRA

To determine the best parameter values, we used the VGG19 model with weights pretrained on ImageNet as the baseline. We then fine-tuned all parameters of the neural network to adapt to a new task: image classification on a subset of the first 100 classes from the CUB-200 dataset. In this configuration, the model achieved an accuracy of 77.3% with approximately 140 million trainable parameters.

Table 1 demonstrates the relationship between the total number of parameters in the classifier and the achieved accuracy. The baseline model, fine-tuned upper bound, with approximately 140 million parameters, achieved an accuracy of 77.3% by fine-tuning all layers. In contrast, models that fine-tuned only the fully connected layers (finetune-1, finetune-2, and finetune-3) achieved lower accuracy scores. This suggests that traditional transfer learning techniques, which update only the fully connected layers, are insufficient to reach the same accuracy that can be achieved by fine-tuning all model parameters. Therefore, to maximize performance, it is essential to allow updates to all layers rather than limiting adjustments to the final layers alone.

As we understand the importance of training convolutional layers to achieve better accuracy in new tasks, our goal is to adapt these layers with minimal overhead in trainable parameters. To this end, we applied ConvLoRA to all convolutional layers and evaluated a range of values for the r parameter. Table 2 presents the results for each configuration, showing the relationship between r, the number of trainable parameters, and the achieved accuracy. These results indicate that while increasing r generally improves accuracy, there is a point of diminishing returns beyond which additional parameters do not substantially enhance performance.

With the convlora-9 configuration, we achieve an accuracy close to that of the fine-tuned upper bound setup, where all parameters are trained. This configuration reduces the requirement for trainable parameters from approximately 20 million to around 850,000, a reduction of 95.75%. Despite this, fully connected layers still contain a substantial number of trainable parameters. Therefore, we fix the r parameter at 9 and apply LoRA to the fully connected layers. The experiment suggests that configurations with more trainable parameters do not improve accuracy, indicating that ConvLoRA achieves an upper limit in performance.

Table 3 shows the relationship between the number of trainable parameters in the fully connected layers and the accuracy achieved in different configurations. In each case, the convlora-9 setting is applied to the convolutional layers, resulting in 850,419 trainable parameters. FullyLoRA-1 includes 819,400 trainable parameters in the classification layer alone, FullyLoRA-2 adds LoRA to all three fully connected layers along with all parameters of the third layer, totaling 2,069,560 trainable parameters, and FullyLoRA-3 applies LoRA to the first two fully connected layers plus all parameters of the third layer, totaling 1,940,680 parameters. These results suggest that selectively applying LoRA to specific fully connected layers offers a practical balance between accuracy and parameter efficiency.

In the following section, we compare the performance of the ConvLoRa adapter on two benchmark

Table 3. Comparison and evaluation of accuracy when adding fully connected layers to the LoRa adapter.

Name	**Convolutional Trainable Parameters**	**Fully Connected Trainable Parameters**	**Accuracy**
FullyLoRA-1	850,419	819,400	0.75
FullyLoRA-2	850,419	2,069,560	0.73
FullyLoRA-3	850,419	1,940,680	0.76

Table 4. Comparison of average accuracy and learning steps by task on the CIFAR-100 dataset (variance in parentheses).

Task	**Method**	**Final Accuracy**	**Epochs**
1	**ConvLoRA**	0.83 (±0.007)	67.00 (±7.329)
	SenaCNN	**0.85 (±0.012)**	**45.13 (±11.934)**
2	**ConvLoRA**	0.88 (±0.008)	41.25 (±5.922)
	SenaCNN	**0.89 (±0.007)**	**31.00 (±4.721)**
3	**ConvLoRA**	0.84 (±0.004)	54.25 (±2.765)
	SenaCNN	**0.84 (±0.008)**	**32.50 (±6.188)**
4	**ConvLoRA**	**0.89 (±0.005)**	48.88 (±6.512)
	SenaCNN	0.89 (±0.005)	**31.88 (±6.978)**
5	**ConvLoRA**	0.85 (±0.005)	53.25 (±4.590)
	SenaCNN	**0.86 (±0.009)**	**37.50 (±5.657)**
6	**ConvLoRA**	0.86 (±0.005)	43.25 (±6.065)
	SenaCNN	**0.86 (±0.006)**	**29.75 (±5.064)**
7	**ConvLoRA**	0.82 (±0.011)	37.88 (±6.289)
	SenaCNN	**0.83 (±0.005)**	**31.13 (±3.482)**
8	**ConvLoRA**	0.84 (±0.005)	50.75 (±6.341)
	SenaCNN	**0.86 (±0.004)**	**36.63 (±2.774)**
9	**ConvLoRA**	0.86 (±0.009)	40.50 (±3.251)
	SenaCNN	**0.86 (±0.007)**	**32.50 (±5.451)**
10	**ConvLoRA**	0.84 (±0.012)	48.75 (±9.677)
	SenaCNN	**0.85 (±0.012)**	**37.75 (±7.536)**

tasks to evaluate the effectiveness of our approach relative to established baseline methods.

Comparison with Benchmark

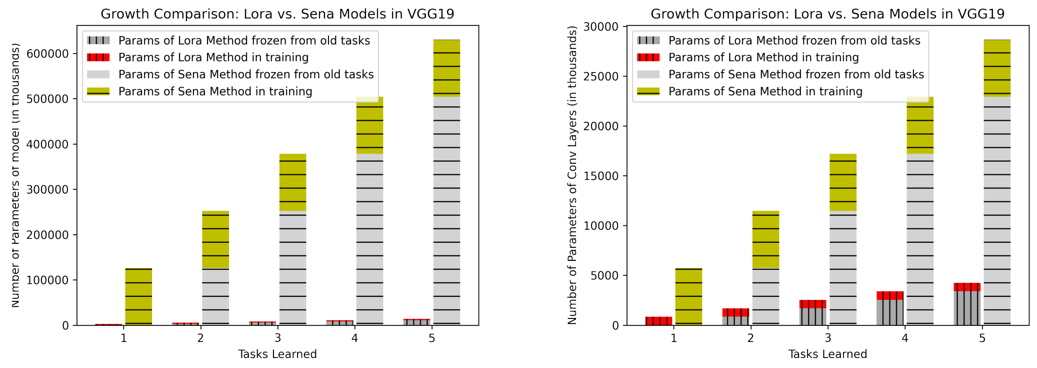
To evaluate the performance of the low-rank adapters and the baseline SenaCNN method, we used two benchmark datasets in an incremental learning setup: CIFAR-100 and CUB-200. Tables 4 and 5 present the experimental results, comparing the accuracy and convergence rates for each task using these datasets. These tables provide a comprehensive analysis, highlighting the achieved accuracies and the required training epochs for each task.

For the CIFAR-100 dataset (Table 4), both methods demonstrate similar final accuracies across all tasks, with SenaCNN slightly surpassing ConvLoRa by a narrow margin of 1-2%. A comparable trend is observed for the CUB-200 dataset (Table 5), though the margin of performance difference widens for the initial task. This first task encompasses 100 classes to learn, in contrast to the subsequent four tasks, which contain only 25 classes each. For comparison, tasks in the CIFAR-100 data set are uniformly distributed with 10 classes per task. These results suggest that the complexity of tasks with higher class counts necessitates models with increased learning capacity.

Notably, while SenaCNN achieves faster convergence in terms of epochs across most tasks, ConvLoRa exhibits significantly shorter overall training times. This discrepancy arises because each epoch of SenaCNN takes longer to complete. With fewer trainable parameters, ConvLoRa accelerates training time per epoch. A detailed analysis of this trade-off is provided in the following paragraphs.

Table 5. Comparison of average accuracy and learning steps by task on the CUB-200 dataset (variance in parentheses).

Task	Method	Final Accuracy	Epochs
1	**ConvLoRA**	0.72 (±0.018)	63.50 (±15.538)
	SenaCNN	**0.76 (±0.004)**	**50.13 (±1.885)**
2	**ConvLoRA**	0.72 (±0.014)	43.50 (±7.309)
	SenaCNN	**0.75 (±0.010)**	**42.13 (±5.540)**
3	**ConvLoRA**	0.70 (±0.010)	41.63 (±4.173)
	SenaCNN	**0.72 (±0.011)**	**38.50 (±5.099)**
4	**ConvLoRA**	**0.79 (±0.026)**	**41.50 (±6.633)**
	SenaCNN	0.79 (±0.017)	43.25 (±5.120)
5	**ConvLoRA**	0.84 (±0.010)	42.38 (±6.632)
	SenaCNN	**0.85 (±0.005)**	**37.50 (±4.899)**



(a) All Layers (b) Convolutional Layers

Figure 6. Cumulative growth comparison of model size. Bars with vertical hatching represent ConvLoRA, while those with horizontal hatching correspond to SenaCNN. The gray sections denote the amount of frozen parameters, while the red and green segments illustrate the new parameters added for each new task. ConvLoRA exhibits superior parameter efficiency, as shown in (a), which depicts cumulative growth across all layers, with ConvLoRA requiring approximately 2.2% of the parameters used by SenaCNN. In (b), the focus shifts to convolutional layers, where ConvLoRA achieves remarkable parameter savings, utilizing only approximately 14.8% of the parameters required by SenaCNN.

All methods belonging to the Dynamic Networks group in the category addressing catastrophic forgetting (CF) literature require an increase in model size for every new task learned, which is a significant drawback (Aleixo et al., 2023). Evaluating how much a model needs to grow as it learns each new task is crucial, as such growth translates to increased computational demands in terms of space, time, and memory. Ideally, techniques that reduce the number of new parameters while supporting incremental learning are preferred, as they enhance both efficiency and scalability. Figure 6 compares the parameter growth between the SenaCNN and ConvLoRA methods.

The trends depicted in Figure 6 highlight the superior tradeoff achieved by ConvLoRA compared to SenaCNN in managing parameter expansion. Using the VGG-19 backbone architecture, ConvLoRA requires only 2.2% of the total parameters required by SenaCNN per task, with a total parameter count of 2,791,099 compared to SenaCNN's 126,101,056. This represents a dramatic reduction in parameter growth, with ConvLoRA achieving an 85.17% reduction in convolutional layer parameters (850,419 versus 5,735,800) and an extraordinary 98.39% reduction in the classifier parameters (1,940,680 versus 120,365,256).

Notably, the growth trends are linear for both methods, as they add a constant number of new parameters for each task. However, ConvLoRA's constant is significantly smaller. Our experiments show that such a drastic reduction in computational requirements does not significantly impact classification performance. The trends illustrated in Figure 6 encompass both convolutional and batch normalization layers, offering a comprehensive evaluation often overlooked in related studies. ConvLoRA's superior parameter efficiency is particularly advantageous in resource-constrained environments, demonstrating its significant edge over SenaCNN while ensuring scalability and practical applicability for continuous learning scenarios.

Overall, the results suggest that both methods are effective in mitigating catastrophic forgetting, with SenaCNN showing marginally better performance in terms of accuracy. However, this often comes at the cost of increased training time.

CONCLUSIONS AND FUTURE WORKS

This study has advanced the understanding of continuous learning in CNNs by demonstrating that CF can be effectively handled by approaches such as the ConvLoRA and SenaCNN methods. Specifically, the ConvLoRA proposed method, showed considerable parameter-efficiency promise. It maintained high accuracy levels across tasks while significantly reducing the number of trainable parameters. For instance, in the CIFAR-100 dataset, the ConvLoRA method not only sustained competitive accuracy comparable to the Sena method but also did so with markedly fewer parameters.

Our results demonstrate that low-rank adapters achieve comparable accuracy to the baseline while utilizing only approximately 2.2% of the parameters, highlighting their remarkable efficiency. Furthermore, we emphasize the critical role of adapters in convolutional layers, showing that fine-tuning only the fully-connected layers does not achieve optimal accuracy compared to fine-tuning all layers. Our model reduces the parameter count in the feature extractor of VGG-19 by approximately 85% while maintaining an accuracy level similar to that of a fully fine-tuned model, underscoring its effectiveness in balancing efficiency and performance. It is worth mentioning that in this study the VGG-19 model was adopted as a proof of concept due to its simplicity and well-established architecture. Nonetheless, the ConvLoRA adapter can be applied to other CNN architectures to address computer vision tasks.

Despite the results, managing continuous learning on a scale, particularly as the number of tasks approaches infinity, remains a significant challenge in the field. Although our research suggests that ConvLoRA is a strong candidate to address this issue, further exploration is needed to assess its scalability and efficiency in broader applications.

It is important to acknowledge the limitations of the proposed adapters. First, they are currently restricted to convolutional and fully-connected layers, making batch normalization layers an additional computational overhead in this approach. Furthermore, in convolutional layers, the parameter reduction benefits are most significant when the number of input and output channels is large. Finally, this study does not evaluate the impact of the adapters in architectures with residual connections, such as ResNet, which may affect their applicability and performance in such scenarios.

For future work, we propose exploring hybrid models that combine ConvLoRA's robustness with methods from other CF categories. The combination of approaches could enhance generalization in continuous learning systems, handling more tasks without significantly increasing resource demands. Hybrid models offer a balanced solution, leveraging multiple methodologies to create flexible and scalable learning architectures. This research direction promises significant advancements in AI, enabling seamless evolution and adaptation of learning systems.

ACKNOWLEDGMENTS

The present work was carried out with the support of the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES-PROEX) - Funding Code 001. This work was partially funded by the Amazonas State Research Support Foundation (FAPEAM) through the PDPG/CAPES and POS-GRAD 2024 projects. We also thank FAPEAM for its support through the project entitled "Different Computational Approaches for Autonomous Ecoacoustic Monitoring of the Amazon Region," PUBLIC CALL NO. 013/2022 - PRODUCTIVITY CT&I.

REFERENCES

Aleem, S., Dietlmeier, J., Arazo, E., and Little, S. (2024). Convlora and adabn based domain adaptation via self-training. *2024 IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 1–5.

Aleixo, E. L., Colonna, J. G., Cristo, M., and Fernandes, E. (2023). Catastrophic forgetting in deep learning: A comprehensive taxonomy. *CoRR*, abs/2312.10549.

Bang, J., Kim, H., Yoo, Y., Ha, J.-W., and Choi, J. (2021). Rainbow memory: Continual learning with a memory of diverse samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8218–8227.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *CoRR*, abs/2005.14165.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. (2019). Efficient lifelong learning with a-GEM. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135.

Goodfellow, I. J., Bengio, Y., and Courville, A. C. (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. In McIlraith, S. A. and Weinberger, K. Q., editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3390–3398. AAAI Press.

Kharitonov, A., Nahhas, A., Pohl, M., and Turowski, K. (2022). Comparative analysis of machine learning models for anomaly detection in manufacturing. *Procedia Computer Science*, 200:1288–1297. 3rd International Conference on Industry 4.0 and Smart Manufacturing.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nat.*, 521(7553):436–444.

Liu, S. and Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2020). Ro{bert}a: A robustly optimized {bert} pretraining approach.

Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T. L., Lange, M. D., Masana, M., Pomponi, J., van de Ven, G., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara, S., Parisi, G. I., Cuzzolin, F., Tolias, A., Scardapane, S., Antiga, L., Amhad, S., Popescu, A., Kanan, C., van de Weijer, J., Tuytelaars, T., Bacciu, D., and Maltoni, D. (2021). Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop.

Mallya, A., Davis, D., and Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IV*, volume 11208 of *Lecture Notes in Computer Science*, pages 72–88. Springer.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In Bower, G. H., editor, *Psychology of Learning and Motivation*, volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Pham, Q., Liu, C., and Steven, H. (2022). Continual normalization: Rethinking batch normalization for online continual learning. In *10th International Conference on Learning Representations, ICLR 2022, Virtual, April 25 - April 29, 2022, Conference Track Proceedings*.

Rebuffi, S., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5533–5542. IEEE Computer Society.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *CoRR*, abs/1606.04671.

Sharma, V., Mir, R. N., and Singh, C. (2023). Scale-aware cnn for crowd density estimation and crowd behavior analysis. *Computers and Electrical Engineering*, 106:108569.

Zacarias, A. S. and Alexandre, L. A. (2018). Sena-cnn: Overcoming catastrophic forgetting in convolutional neural networks by selective network augmentation. In Pancioni, L., Schwenker, F., and Trentin, E., editors, *Artificial Neural Networks in Pattern Recognition - 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19-21, 2018, Proceedings*, volume 11081 of *Lecture Notes in Computer Science*, pages 102–112. Springer.

Zhang, Z., Hanwen, G., and Wu, X. (2023). Detection of pedestrians and vehicles in autonomous driving with selective kernel networks. *Cognitive Computation and Systems*, 5(1):64–70.