IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 4, 2021, accepted June 16, 2021, date of publication June 18, 2021, date of current version June 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3090672

Overcoming Catastrophic Forgetting Using Sparse Coding and Meta Learning

JULIO HURTADO [1], HANS LOBEL [1,2], (Member, IEEE), AND ALVARO SOTO [1], (Member, IEEE)
[1]Department of Computer Science, Pontificia Universidad Católica de Chile, Santiago 7820436, Chile
[2]Department of Transport Engineering and Logistics, Pontificia Universidad Católica de Chile, Santiago 7820436, Chile

Corresponding author: Julio Hurtado (jahurtado@uc.cl)

This work was supported in part by the National Fund for Scientific and Technological Development (FONDECYT) under Grant 1181739 and Grant 11181152, in part by the Millennium Institute for Foundational Research on Data (IMFD), and in part by the National Research and Development Agency (ANID)—Doctorado Nacional 2017.

ABSTRACT Continuous learning occurs naturally in human beings. However, Deep Learning methods suffer from a problem known as Catastrophic Forgetting (CF) that consists of a model drastically decreasing its performance on previously learned tasks when it is sequentially trained on new tasks. This situation, known as task interference, occurs when a network modifies relevant weight values as it learns a new task. In this work, we propose two main strategies to face the problem of task interference in convolutional neural networks. First, we use a sparse coding technique to adaptively allocate model capacity to different tasks avoiding interference between them. Specifically, we use a strategy based on group sparse regularization to specialize groups of parameters to learn each task. Afterward, by adding binary masks, we can freeze these groups of parameters, using the rest of the network to learn new tasks. Second, we use a meta learning technique to foster knowledge transfer among tasks, encouraging weight reusability instead of overwriting. Specifically, we use an optimization strategy based on episodic training to foster learning weights that are expected to be useful to solve future tasks. Together, these two strategies help us to avoid interference by preserving compatibility with previous and future weight values. Using this approach, we achieve state-of-the-art results on popular benchmarks used to test techniques to avoid CF. In particular, we conduct an ablation study to identify the contribution of each component of the proposed method, demonstrating its ability to avoid retroactive interference with previous tasks and to promote knowledge transfer to future tasks.

INDEX TERMS Artificial intelligence, learning (artificial intelligence), machine learning, supervised learning, continual learning.

I. INTRODUCTION

Among the cognitive abilities of humans, memory is one of the most relevant. In effect, the ability to recall past experiences, knowledge, friendships, and emotions, is rooted in the essence of what makes us humans. However, memories are fragile, in particular, studies from cognitive psychology [6], [41] show that there are 3 main mechanisms related to loss of memories: i) retrieval failure, ii) task interference, and iii) lack of consolidation. In terms of retrieval failure, forgetting occurs when long-term memory is no longer accessible because the retrieval cues are no longer present. In terms of task interference, memories can be disrupted by similar memories or related information, leading to *proactive interference* where old information disrupts new learning, or to *retroactive interference* where new knowledge disrupts old information. Finally, in terms of lack of consolidation, memories can be lost during the process of transforming short-term to long-term memories.

In the case of artificial neural networks (ANN), task interference is behind the problem known as Catastrophic Forgetting (CF) [29], [40], [46], [51], [53]. Specifically, as a model learns a new task, the weights of the network are modified. This new learning provokes forgetting old information stored in the weights of the model, causing *retroactive interference*.

The problem of task interference in ANN is more severe than in the case of humans. In effect, when a previously trained model is retrained on a new task, it usually suffers a significant drop in performance in the original task [26], [49], [60]. In contrast to artificial models, humans have a remarkable ability to achieve continual learning without experiencing CF problems. As an example, humans

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

continuously associate previous experience to new situations, strengthening previous memories, and avoiding retroactive interference [25].

Taking inspiration from the robustness of humans to avoid retroactive interference, we develop a learning strategies to face continual learning in ANN, *i.e.*, the case when data for each new task is presented sequentially to the learner, who does not have access to previous data.

In this work, we propose a new method that exploits two complementary learning strategies to mitigate CF in ANN, in particular, deep Convolutional Neural Network (CNN). These learning strategies are based on: i) dynamic allocation of model capacity to avoid interference between tasks, and ii) knowledge transfer from previous to new tasks to foster weight reusability instead of overwriting.

In terms of dynamic allocation of model capacity, there is usually a trade-off between providing flexibility to a model to adapt its parameters to a new task versus keeping relevant parameters unchanged. Previous works have mainly focused on limiting this flexibility by penalizing changes of what they identify as essential weights [2], [3], [26], [60]. In this work, we propose a method to reduce interference by dynamically allocating part of the network capacity to each task. We achieve this by using sparse coding techniques [13] to adaptively manage the number of parameters that are used to learn each new task, while keeping the total model capacity fixed. In contrast to previous approaches [38], we apply group sparse regularization to foster the use of groups of parameters that specialize to learn a task. Specifically, by adding selectors, in the form of binary masks, we are able to freeze groups of parameters that are key to solve a task, reserving the rest of the network to learn new tasks.

In terms of knowledge transfer to new tasks, if a model is able to learn parameters that are useful for past and future tasks, there would be no need to drastically change their values. Following this intuition, we propose a training strategy that fosters learning of patterns that can be useful to support several tasks. In particular, in the context of few-shot learning [16], [45], meta-learning techniques have been used to promote the learning of weights that can be quickly adapted to handle new tasks [10], [43], [48]. Taking inspiration from these ideas, we propose a meta-learning strategy that fosters learning of weights that can be useful to favor a positive transfer of knowledge between tasks, facilitating the acquisition of continual learning skills.

In summary, this work makes the following three main contributions in the context of a continual learning scenario:

- A new learning strategy that uses group sparse regularization to selectively train groups of parameters to learn each task and, at the same time, reserve part of the network capacity to learn future tasks, helping to avoid the problem of retroactive interference.
- A new learning strategy that uses a meta-learning approach to foster knowledge transfer between tasks by learning weights that can be useful to face future tasks.
- Extensive empirical evidence indicating that the combination of the two previous contributions provides a substantial impact to improve the robustness of our model to avoid CF in a continual learning scenario.

The remainder of the manuscript is organized as follows: in Section II we briefly describe previous and related works in continual learning and meta-learning. Section III describes our proposal, first the group sparse regularization and then the meta-learning strategy. Section IV presents our main experiments and results. Finally, Section V details our conclusions.

II. RELATED WORK

A. CATASTROPHIC FORGETTING

In principle, a simple approach to avoid CF is to retrain the model with all the available data, old and new, each time a new task arrives. This approach is, however, unfeasible in many situations, for instance, when old data are no longer available. Even in cases where one can store every dataset, usually the computational cost of re-training using all data can be highly inefficient or prohibitive. Therefore, there is a need for more efficient and flexible solutions.

In the context of sequential learning, previous work has followed two main strategies to prevent CF. The first strategy [2], [3], [12], [26], [60] consists of avoiding the modification of key parameters for previous tasks when learning a new task. Several techniques can be used to identify these critical parameters, such as fisher matrix [26] and per-weight uncertainty [14], among others [2], [3], [10]. Afterwards, when facing a new task, a regularization term ensures that critical parameters are modified as little as possible. By using this strategy, performance on previous tasks does not decrease abruptly when learning new ones. At the same time, the network has flexibility to capture information from new tasks. In general, this approach shows satisfactory performance in problems that involve few tasks, however, when the number of tasks increases, problems such as accumulated drifting in weight values and interference among them, make this approach difficult to scale. Alternatives to decrease the interference between tasks are presented in [35], [39], here the authors propose to completely freeze previously trained weights eliminating interference but inhibiting information transfer between tasks.

The second strategy [15], [21], [32], [37], [38], [51] consists of introducing structural changes to the architecture of the models. [49] proposes cloning a model and adding connections between the layers of previous models to the new one, creating an exchange of information from old to new tasks. As a drawback, the amount of disk space required by the model increases linearly with the number of tasks. A popular approach is to incorporate trainable binary masks that are used to select parameters, either through pruning or learning over a backbone model [37], [38]. A downside of this strategy is that the masks are learned independently of network parameters, leading to suboptimal solutions. Another popular strategy is to use memory replay to recall critical information about previous datasets [21], [47], [48], either through saving elements of previous tasks or training GANs to generate those elements past elements [17], [31], [52]. The main idea is to recreate previous tasks distribution. The main problem with these methods is the need of an efficient method to recall key information from previous tasks and the need to have access to inputs from past tasks. Instead of saving elements of prior dataset [7], [22] proposed saving feature vectors. This solution reduces privacy and memory concerns, as these vectors typically require less memory than complete elements but still need to have access to previous datasets to create the vectors.

A related topic to Continual Learning and Catastrophic Forgetting is Distribution drifts. This field aims to train models able to adapt well to change in the distributions in which they were trained [1], [5]. This scenario is related to the idea of transfer learning [58], and data stream problems [8]. Unlike previous scenarios, the goal of Continual Learning is to acquired new knowledge from new distributions without loss of performance of previously learned task, avoiding interference and CF [30].

In contrast to the methods above, we use sparsity techniques to train group of parameters that are relevant for each task, implementing an efficient dynamic allocation of the available resources. Furthermore, by fostering the recycling of parameters from previous tasks, we implement an efficient use of previous knowledge to support new tasks.

B. META-LEARNING

Metalearning is the ability of "learning to learn" [54], in other words, the ability to discover proper biases or procedural knowledge that can be used to learn new tasks. In effect, the ability to generalize across tasks is at the core of metalearning. In this context, [23], [24], [44] propose different metalearning strategies to tackle a continual learning scenario. The resulting techniques reduce task interference by avoiding conflicts between current and future gradient directions to update weights. In [56] and [18] the authors propose the use of a meta-model to avoid CF. This meta-model operates as a general hyper-network that is conditioned by each task to generate the weights of a task-specific network. In [48], authors combine a replay memory with a metalearning strategy, looking to take advantage of the benefits of both techniques. Training prototypes by class has also been explored [11], these prototypes are adapted as new classes arrive, and classification is made using a distance metric.

In the context of few-shot learning, several works [9], [16], [36], [42], [57] have proposed meta-learning strategies that achieve fast adaptation of weights to model a new task. A popular approach is based on fostering weight values that can be adjusted to model a new task using just a few gradient updates. Taking inspiration from this idea, our method to foster knowledge transfer from previous to new tasks is based on adapting the methods presented in [16], [42] to a continual learning scenario.

III. METHOD DESCRIPTION

In this section, we present our proposed method to avoid the CF problem. As we mentioned, our method consists of two main steps: A) a learning strategy that avoids interference among tasks, and B) a training scheme that fosters weight sharing among tasks. Next, we discuss the technical details behind each of these steps.

A. AVOIDING INTERFERENCE AMONG TASKS

As a first strategy, we directly tackle interference among tasks by introducing a mechanism that prevents new tasks from modifying weights that are relevant to solve previous tasks. As a key observation, we acknowledge that, in the context of a CNN, learning a task consists of finding suitable convolutional filters to correctly map inputs to outputs. As a consequence, avoiding interference among tasks is directly related to avoiding that a new task might modify a filter that is relevant to solve a previous task.

Following the previous observation, we introduce an adaptive mechanism to control the number of convolutional filters that are available to learn new tasks. Specifically, this mechanism avoids interference among tasks by freezing the value of weights associated to convolutional filters that are relevant to solve previous tasks. To be effective, this mechanism has to balance two goals: It must provide the model with enough freedom to learn suitable convolutional filters to solve its current task, while, at the same time, it must also restrict this freedom in order to preserve knowledge from previous tasks. To achieve these goals, we modify the regular loss function used by CNNs, introducing a group sparsity regularization term (*GoSpaR*). This term fosters a sparse learning of convolutional filters, leading to an adaptive use of network resources as our model incrementally learns new tasks. We describe next the mathematical details behind this approach.

1) GROUP SPARSE REGULARIZATION OVER CONVOLUTIONAL FILTERS

In our formulation, we consider a CNN classification model with L layers; where $L-1$ layers are convolutional and the last one, or classification head, corresponds to a fully connected ANN. Furthermore, we consider a set of training examples $\{x_i, y_i\}_{i=1}^{N}$, where x_i refers to input i and y_i to its corresponding label. For such a model, learning can be performed by solving the following optimization problem:

$$\underset{W,\Theta}{\text{argmin}} \; \frac{1}{N}\sum_{i=1}^{N} Loss(x_i, y_i; W, \Theta) + R_{wd}(W, \Theta) \tag{1}$$

where $\Theta = \{\Theta^1, \ldots \Theta^{L-1}\}$ denotes the parameters of the $L-1$ convolutional layers, Θ^l denotes the parameters of convolutional layer l and W represents the parameters of the classification head.

The problem in Equation (1) is divided into two components. The first component (*Loss*) accounts for the difference between the target output y_i and the prediction of the model with parameters $\{W, \Theta\}$. We use cross-entropy as the *Loss* function. The second component is a weight decay regularization term that helps to avoid overfitting, define as:

$$R_{wd}(W, \Theta) = C_{wd}\frac{1}{2}(\|W\|_F^2 + \|\Theta\|_F^2), \tag{2}$$

where $\|\cdot\|_F^2$ denotes squared ℓ_2-norm and C_{wd} is the corresponding regularization constant.

To avoid interference among tasks, we augment Equation (1) by including *GoSpaR* over the convolutional layers, as follows:

$$\underset{W,\Theta}{\text{argmin}}\ \frac{1}{N}\sum_{i=1}^{N} Loss(x_i, y_i; W, \Theta) + C_{wd}\frac{1}{2}\|W\|_F^2 + C_\Theta \sum_{l=1}^{L-1}\Gamma_l(\Theta^l), \tag{3}$$

where C_Θ is a regularization constant. We define Γ_l as:

$$\Gamma_l(\Theta^l) = (1-\beta^l)\frac{1}{2}\|\Theta^l\|_F^2 + \beta^l \sum_{k=1}^{K^l}\|\Theta^l_{k,*}\|. \tag{4}$$

The first term in Equation (4) corresponds to an ℓ_2-norm regularizer, which is weights by coefficient $(1-\beta^l)$. The second term uses an $\ell_{1,2}$-norm to penalize the number of filters used by each layer. Specifically, K^l corresponds to the number of filters in layer l; $\Theta^l_{k,*}$ denotes the weights of layer l associated to filter k; and, finally β^l regulates the importance of setting filters in layer l to zero.

The goal of *GoSpaR* is to minimize the number of groups used by the model, setting to zero groups of unused parameters. A similar group sparsity inducing regularizer has been previously used in applications related to image classification [4], [33], [34], [50], [59], [61]. In our case, we select groups in such a way that, when learning a task, the regularization function fosters the use of a limited number of convolutional filters, leaving network resources available to learn future tasks.

Fig. 1 shows the effect of *GoSpaR* in the operation of a generic layer l of a CNN model. In this case, the regularization sets filter k_i^l to zero which corresponds to filter i in layer l. As a consequence, the corresponding feature map M_i^l can be deactivated using function M, shown in Equation (5). We use ℓ_2-norm to decide if a filter is active or not. After training task t, filters of layer l whose ℓ_2-norm is less than threshold ε are considered as inactive filters, therefore, they are available to be trained by future tasks.

$$M(\Theta^l) = \|\Theta^l_{\cdot,\cdot}\|_2 \leq \varepsilon \tag{5}$$

For each task t and layer l with K^l filters, we keep track of the list of active filters by using a binary mask $m_l^t \in [0, 1]^{K^l}$ that associates a binary coefficient to each filter. A similar procedure has been used before in [37] and [51]. For the initial task $t = 1$, the mask is initialized with all its coefficients equal to 1, representing that all filters are available for training. After training a task t, filters of layer l with ℓ_2-norm greater than ε are marked as used filters, and the corresponding flag is set to zero in the associated mask m_l^t. Afterwards, the resulting mask m_l^t is used to initialize mask m_l^{t+1} to train the next task.

During training, binary masks m_l^t are used to prevent interference from new tasks by freezing weights that are relevant to previous tasks. During test, binary masks are used to identify the list of active filters for each task. Using this information, we let a task to use only the filters that were available during its training, avoiding potential interference from filters that were learned by subsequent tasks.

One drawback of the previous training scheme is that it is not trivial for new tasks to use knowledge acquired during previous tasks. This is mainly due to normalization problems associated to the independent training of filters that are being freezed from previous tasks. To account for this limitation, we take inspiration from [20] to include a task-specific function that learns to combine the outputs of all the convolutional layers available to the task. Next, we present the details behind this idea.

2) CALIBRATION OF FILTERS FROM DIFFERENT TASKS

In a standard single-task learning setting, when a model learns a task, it calibrates the relevance of each weight and filter in the context of the rest of the weights and filters that are being concurrently trained. However, in our case, we have multiple tasks that are being learned sequentially. In our setting, each new task can train weights of unused filters, but it can also use previously learned filters that it cannot modify. Thus, the model has to learn how to combine both sources of information.

To facilitate the combination of filters from different tasks, we introduce a normalization step to calibrate the outputs of all the convolutional layers available to a task. Taking inspiration from the mechanism behind the Squeeze and Excitation Network (SE) [20], for each task we learn a normalization function that scales the activation maps of each convolutional layer.

Specifically, the Calibration of Filters *(CaFil)* function (O_l) is defined in Equation (6), were each activation map M_i^l is weighted by the outputs of the normalization function Φ_{lt}, via an element-wise multiplication $(\cdot)$. This multiplication helps to strengthen or weaken the activation maps on M^l.

$$O_l(\theta_l, M^l) = M^l \cdot \Phi_{lt}(M^l) \tag{6}$$

Function Φ_{lt} encodes the specialization and normalization weights for layer l and task t, by squeezing and aggregating the representation M^l to find the corresponding values, according to:

$$\Phi_{lt}(M^l; W_{tl}^{1,2}) = \sigma(W_{tl}^2\ \rho(W_{tl}^1 M^l)), \tag{7}$$

where ρ and σ represent the ReLU and Sigmoid activation functions, respectively. Weights W_{tl}^1 and W_{tl}^2 are learned during training.

By adding functions Φ_{lt}, we seek to balance the outputs of all the convolutional layers. Furthermore, by having a specific

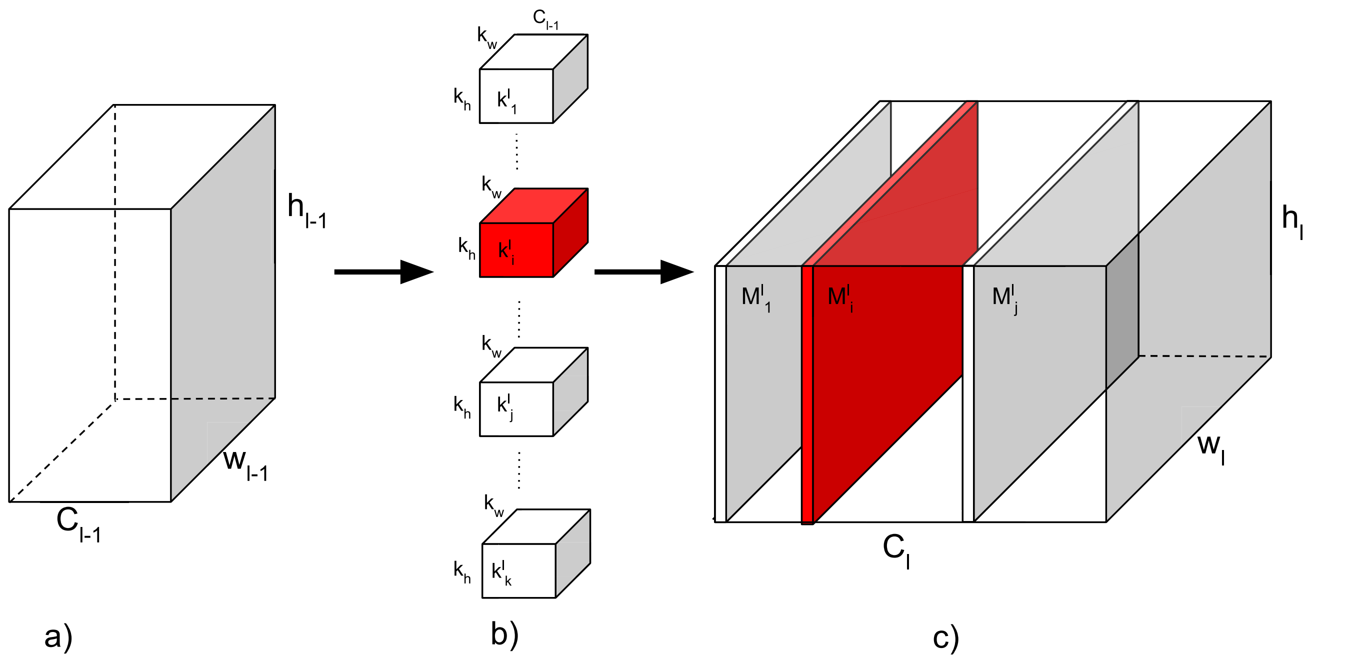


FIGURE 1. (a) Activation maps at convolutional layer $l-1$ of size $h_{l-1} \times w_{l-1} \times C_{l-1}$. (b) At layer l, these activation maps are processed by convolutional filters k^l_i, $i \in [1, \ldots, K]$, of size $k_h \times k_w \times C_{l-1}$. (c) In this case, *GoSpaR* sets filter k^l_i to zero, therefore, the corresponding activation map M^l_i in layer l can be deactivated.

function per task, we also seek task specialization in the use of filters. By achieving both goals, the model learns to combine knowledge from previous and current tasks.

3) FILTER TRAINING

The training process of a task t is shown in Algorithm 1. For each batch $\{x^B_i, y^B_i\}$, we obtain the corresponding predictions $\hat{y}^B_i$ using the current weights Θ and W_t, i.e., weights of convolutional layers and classification head for task t, respectively. Then, given the classification ($Loss$) and the regularization (Γ) terms, we obtain the gradient for both sets of weights (g_Θ, g_{W_t}). Afterwards, we multiply gradients of Θ by the corresponding binary masks m^t to set the selected gradients to zero. Finally, we update the weights of free filters and classifier with a learning rate α.

Algorithm 1: TaskTraining

```
**Input**: Data ($D_t$), Model ($f$), Weights ($W_t$,$\Theta$),
Binary Mask ($m^t$), Loss Function ($Loss$)
**Output**: Trained Weights ($W_t$,$\Theta$)
for $x^B_i, y^B_i$ in $D_t$ do
    $\hat{y}^B_i = f(x^B_i, W_t, \Theta)$
    # Get gradients
    $g_\Theta, g_{W_t} \leftarrow \nabla(Loss(\hat{y}^B_i, y^B_i) + \Gamma(\Theta))$
    # Freeze used weights
    $g_\Theta \leftarrow m^t \cdot g_\Theta$
    # Update weights
    $\Theta \leftarrow \Theta - \alpha \cdot g_\Theta$
    $W_t \leftarrow W_t - \alpha \cdot g_{W_t}$
```

B. FOSTERING WEIGHT SHARING AMONG TASKS

So far, our proposed model has the ability to mitigate the problem of interference among tasks. However, it still lacks a mechanism to encourage sharing of weights among tasks. As we mentioned, a highly desirable feature is to foster knowledge sharing between tasks, *i.e.*, learning filters that can be useful to solve several tasks. In an incremental learning scenario, this reduces to learn filters that, besides the current task, can also support useful representations to solve future tasks.

The previous observation highlights a close relation between incremental and meta-learning scenarios [16], [45], [55]. In effect, the ability to generalize across tasks is at the core of meta-learning. In this context, [55] proposes a meta-learning strategy known as Episodic Training *(ET)* that consists of sampling from a task pairs of support and query sets to simulate training data from a large number of mini-tasks. This mini-tasks are then used to bias the learner to improve its ability to generalize to future tasks. Following this strategy, [16] proposes an optimization method to find network weights that can be quickly adapted to model new tasks. Taking inspiration from [16], we adapt its meta-learning strategy to the case of continual learning, specifically to foster weight sharing among tasks.

In our implementation, when training each new task, we alternate between using regular and *ET* during a predefined number of iterations. In the case of regular training, we apply the learning strategy described in Section III-A. In the case of *ET*, we create a set of U mini-tasks, where each mini-task consists of randomly sampling from the current task a set of H classes and h training instances per class. This allows us to create classes independent of the main task, finding weights not specific to it. Specifically, we consider a training batch $\{x_u^{Tr}, y_u^{Tr}\} \sim P(D_t)$ for mini-task u, where x_u^{Tr} refers to the inputs and y_u^{Tr} to the corresponding labels.

Following [16], our method for *ET* consists of two nested loops, an inner and an outer loop. The inner loop is in charge of training a model for the current mini-task while the outer loop is in charge of updating weights following a gradient direction that leads to fast adaptation to new mini-tasks. Specifically, for each mini-task u, we take as the initial value a copy of the model parameters in epoch e, namely Θ_e, and obtain a new set of parameters Θ_s^u, after iterating s times over mini-task u, expressed in Equation (8):

$$\Theta_{i+1}^u = \Theta_i^u - \alpha \nabla_{\Theta_i^u} Loss_u(f_{\Theta_i^u}(x_u^{Tr}), y_u^{Tr}), \tag{8}$$

where $f_{\Theta_i^u}(x_u^{Tr})$ represents the output of the model f with parameters Θ_i^u when training mini-task u in the i step of the inner loop, we define $\Theta_0^u = \Theta_e$. $Loss_u$ corresponds to the loss function of task u and α is the learning rate of the inner loop. In our case, the *Loss* of every task is cross-entropy.

After learning U models, we update parameters Θ_e of the original model, sampling a new set of mini-task x_u^{Va}. We accumulate the loss of all the mini-tasks as the sum of all the losses given the meta-training validation set. As shown in Equation (9), we accumulate this sum by considering the model trained by s inner loop steps respective to mini-task u and weighted by the outer loop learning rate β:

$$\Theta_e \leftarrow \Theta_e - \beta \nabla_{\Theta_e} \sum_u^U Loss_u(f_{\Theta_s^u}(x_u^{Va}), y_u^{Va}) \tag{9}$$

By adding the meta-learning strategy, the complete training process of a task is described by Algorithm 2, where we first have a few epochs (5 epochs) of adapting the task using Algorithm 1. Afterwards, we start training the model with the meta-training strategy for a few iterations, to then train the model with traditional training for one epoch. We repeat this process until we obtain a suitable level of accuracy.

It is important to mention that during meta-training, the goal is to adapt convolutional filters so they are useful for other tasks. For this reason, neither the task-specific functions nor the task classifier of the current task are modified during this process. Furthermore, during meta-learning, *GoSpaR* is not being used, since the objective of the meta-learning strategy differs from the goals of the method proposed in Section III-A. In this sense, the interleaved application of regular and *ET* steps complement each other, leading to a novel and useful method to train models under a continual learning scenario.

In this work, we introduce three key mechanisms or components to help avoiding the CF problem: i) Group-Sparse Regularization (*GoSpaR*), ii) Calibration of Filters (*CaFil*), and iii) Episodic Training (*ET*). While the first helps to reduce interference between tasks, the other two encourage weight-sharing among tasks. Given these components, we named our whole model *GoCaT*.

Algorithm 2: MetaTraining

```
Input: Data ($D_t$), Model ($f$), Weights ($W_t$,$\Theta$),
Binary Mask ($m^t$), Loss Function ($Loss$),
Learning Rates ($\alpha$, $\beta$),
Hyper-parameters ($U$, $s$, $epochs$, $\tau$)
Output: Trained Weights ($W_t$,$\Theta$)
for $e$ in $epochs$ do
    if $e > \tau$ then
        for $u$ in $[1, 2, \dots U]$ do
            $x_u^{Tr}, y_u^{Tr} \sim P(D_t)$
            $x_u^{Va}, y_u^{Va} \sim P(D_t)$
            for $i$ in $[1, 2, \dots s]$ do
                $\Theta_{i+1}^u \leftarrow \Theta_i^u - \alpha \nabla_{\Theta_i^u} Loss_u(f_{\Theta_i^u}(x_u^{Tr}), y_u^{Tr})$
        $\Theta_e \leftarrow \Theta_e - \beta \nabla_{\Theta_e} \sum_u^U Loss_u(f_{\Theta_s^u}(x_u^{Va}), y_u^{Va})$
    $\Theta_{e+1} \leftarrow$ TaskTraining($\Theta_e$)
```

IV. EXPERIMENTAL EVALUATION

We start discussing the datasets and baselines that we use in our experiments. Afterwards, we explain implementation details behind our model. Finally, we present our main results and an ablation study of the key parts of the proposed method.

A. DATASET

We test our method using 3 popular benchmarks used to test continual learning approaches. All of them correspond to visual recognition applications. First, we consider the so-called 5-Dataset [15], which consists of sequentially training a model using data from 5 datasets: CIFAR10, not-MNIST (nMNIST), SVHN, MNIST, and Fashion-MNIST (fMNIST), not necessarily in that order. To maintain consistency with the number of channels of the input, for grayscale images, we repeat the channel three times to simulate having three channels. As the second scenario, we use 20-Split CIFAR100 dataset [27], which consists of dividing the CIFAR100 dataset into 20 different tasks, each with only 5 different classes. Finally, we use the so-called Permuted MNIST (P-MNIST) dataset, which consists of training using a modified version of the MNIST dataset [28], where each task is a new random permutation of the pixels in each image. Table 1 shows a summary of each dataset.

B. BASELINES

As a first baseline, we compare our results with a strategy based on sequential learning without considering any modification to a regular training scheme. We refer to this strategy as *SGD*, since it only applies Stochastic Gradient Descent during training without considering previous tasks or any

TABLE 1. Size and details of the datasets used in our experiments.

	CIFAR10	nMNIST	SVHN	MNIST	fMNIST	CIFAR100
Train	42500	15526	62269	51000	51000	42500
Validation	7500	2739	10988	9000	9000	7500
Test	10000	459	26032	10000	10000	10000
Color images	Yes	No	Yes	No	No	Yes

particular regularization to avoid CF. This baseline provides us with a lower bound in terms of accuracy. As a second baseline, we consider a multi-task training scenario, where all tasks are learned simultaneously. We refer to this strategy as Joint-Training (JT). This baseline provides us with an optimistic or upper-bound scenario where the learner has access to all the data during its training process.

Besides the two previous baselines, we also compare our results against recent works that also tackle the CF problem. Specifically, we consider works that focus on using regularization techniques to avoid CF, such as, Elastic Weight Consolidation (*EWC*) [26] and Synaptic Intelligence (*SI*) [60]. We also compare our approach to Hard Attention to the Task (*HAT*) [51] and Adversarial Continual Learning (*ACL*) [15]. The first one uses gate functions per task to reduce forgetting, and the second approach uses extra functions per task with an adversarial training strategy. For the first two methods, we use the implementations described in [19]. For *HAT* and *ACL*, we use the original implementation of the authors. For a fairer comparison between works, the same base model is used, adding only the corresponding methods over it. This ensures a similar amount of parameters used by each method, changing only techniques to avoid CF. In all cases, we perform a search for the best hyper-parameters for every dataset.

Following previous works, we use two metrics to compare all methods. First, Mean Accuracy (Mean Acc) measures the average accuracy obtained in each task at the end of the final training process, as is shown in Equation 10, where $Acc_{T,t}$ is the accuracy of task t after training task T. Second, Backward Transfer (BWT) measures the performance impact that learning a new task produces over previous tasks. Specifically, a negative BWT score indicates that the model is forgetting more than what is learning from a new task. On the contrary, a positive BWT score indicates that the model is improving its overall learning when it is trained using a new task. Equation 11 indicates how to compute BWT, where $Acc_{t,t}$ indicates the accuracy obtained by task t at the end of training task t.

$$Acc = \frac{1}{T}\sum_{t=1}^{T} Acc_{T,t} \tag{10}$$

$$BWT = \frac{1}{T-1}\sum_{t=1}^{T-1} Acc_{T,t} - Acc_{t,t} \tag{11}$$

C. IMPLEMENTATION DETAILS

For all experiments and methods, we use the following architecture. This consists of 4-blocks of convolutional layers that are common to all tasks. Each block consists of a convolutional layer with 32 filters and a kernel size of 3, batch normalization, ReLU activations, and a max-pooling layer. These block are followed by a task specific classification layer. As explained in Section III-A1, the main reason for using this architecture is that our method selects relevant convolutional filters for each task. For this reason, we need a model that only has these types of layers. Instead of creating a new architecture, we use similar architecture to the one used in [16], [45]. Additionally to the architecture, we add method-specific functions, like the CaFil function described in Section III-A2, or the gate functions describe in HAT or the adversarial block from ACL.

For the optimization process, we train each task for 50 epochs, using an SGD optimizer with a learning rate of 0.003 and a batch size of 64. During meta-training, we sample 25 elements and create 5 classes with them for each mini-task. Code available at https://github.com/JuliousHurtado/Meta-Iteration.

As in previous works [26], [51], we assume that we do not have access to the total number of classes of the complete scenario, these are revealed as the new task arrives. For this reason, a new classifier is initialized for each task with the corresponding class number. At inference time, we have access to the ID of the task that we are testing.

D. RESULTS

1) 5-DATASET

We start by presenting our results in the 5-Dataset sequence. As a relevant feature, tasks in this sequence are highly dissimilar because images in each dataset are coming from different scenarios, with different scales, lighting, colors, and other variations. In our test, we train each method 3 times using different task orders.

Table 2 resumes our main results in terms of Mean Acc and BWT metrics. By training without restriction, the SGD model obtains close to 27% accuracy. Also, the value of BWT score indicates that learning a new task catastrophically interferes with what has been learned in previous tasks. When using our approach, we obtain 63, 7% of average accuracy and low variance between runs, outperforming all the alternative methods by a large margin. This illustrates the positive effect of the mechanisms that we propose to prevent CF. Furthermore, by considering BWT score in Table 2, we observe that our method is the only one with a positive score, indicating effective incremental learning during the sequence of tasks.

As expected, JT obtains the best performance for this scenario, since it has available all the data during training.

TABLE 2. Results using the 5-Dataset. Showing the mean Accuracy, BWT and memory required for each method.

	Mean Acc. (std.)	BWT	KB
SGD	27.4% (0.01%)	-0.6067	131.5
EWC	44.1% (0.11%)	-0.1054	131.5
SI	42.6% (0.02%)	-0.0718	131.5
HAT	59.2% (0.04%)	-0.1182	145.0
ACL	56.4% (0.02%)	-0.2484	491.1
GoCaT	**63.6%** (0.04%)	**0.0007**	147.0
JT	75.18% (-)	-	131.5

Regarding memory usage, our approach needs to store the specialization functions and the binary mask per task, therefore, there is an small overhead in memory requirement. In this sense, as we can observe from the third column of Table 2, the proposed method uses a similar amount of memory than HAT, which uses gate functions per task.

2) 20-SPLIT CIFAR100

In the case of CIFAR100, the original dataset is divided into 20 different tasks. In contrast to the 5-Dataset sequence, here the sequence of tasks has very similar images, all in color and same dimensions. Therefore, we expect a high degree of knowledge sharing among tasks.

Table 3 resumes our main results. As tasks are more related to each other, it can be seen that the difference in average accuracy between SGD and the best method is reduced. Again our method outperforms the baselines and alternative approaches. In particular, our method outperforms the runner-up, HAT, by 4% in terms of average accuracy. HAT and our method achieve positive learning concerning the BWT score, showing that both methods are able to exploit the close relation among the training tasks.

TABLE 3. Results using the 20-Split CIFAR100 dataset.

	Mean Acc.	BWT	KB
SGD	34.9%	-0.4591	169.0
EWC	37.8%	-0.4542	169.0
SI	43.4%	-0.2363	169.0
HAT	55.7%	0.0066	182.0
ACL	58.8%	-0.1880	874.6
GoCaT	**59.9%**	**0.0078**	195.0
JT	60.1%	-	169.0

Given the close relation among the tasks, it is noteworthy that our method is the only one that reaches a performance highly similar to the upper-bound given by JT. This illustrates the relevance of the proposed strategy to share knowledge among the training tasks.

3) P-MNIST

The last scenario to test our method is the Permuted MNIST dataset. This dataset consists of 10 tasks that are created by applying 10 random permutations to the pixels of the images in MNIST dataset. Due to the random permutations, patterns to identify each class change significantly among tasks. Therefore, it is expected a low level of pattern sharing among tasks.

Similar to the previous scenarios, our approach outperforms the alternative methods, obtaining an average accuracy of 65%, as is shown in Table 4. However, given the large difference among the training tasks, in this case there is a large gap with respect to the upper-bound given by JT. Actually, for this dataset, we can observe that any of the sequential learning methods is able to obtain positive score in terms of BWT. This illustrates the difficulty of sharing visual patterns between tasks for this dataset.

TABLE 4. Results obtained in the sequence of 10 different permutations of the MNIST dataset.

	Acc	BWT	KB
SGD	25.2%	-0.7781	162.0
EWC	39.5%	-0.3859	162.0
SI	40.6%	-0.5308	162.0
HAT	54.8%	-0.3553	174.0
ACL	32.5%	-0.7100	766.0
GoCaT	**65.1%**	**-0.0154**	181.0
JT	92.9%	-	162.0

E. ABLATION STUDY

In this section, we perform an ablation study over the main components of our proposed method. Furthermore, we also analyze the impact of the meta-learning strategy in terms of the trade-off between model flexibility to adjust parameters and interference between tasks. Finally, we carry out a study of the complexity of the model. All these experiments are carried out using the 5-Dataset as benchmark.

1) COMPONENTS ANALYSIS

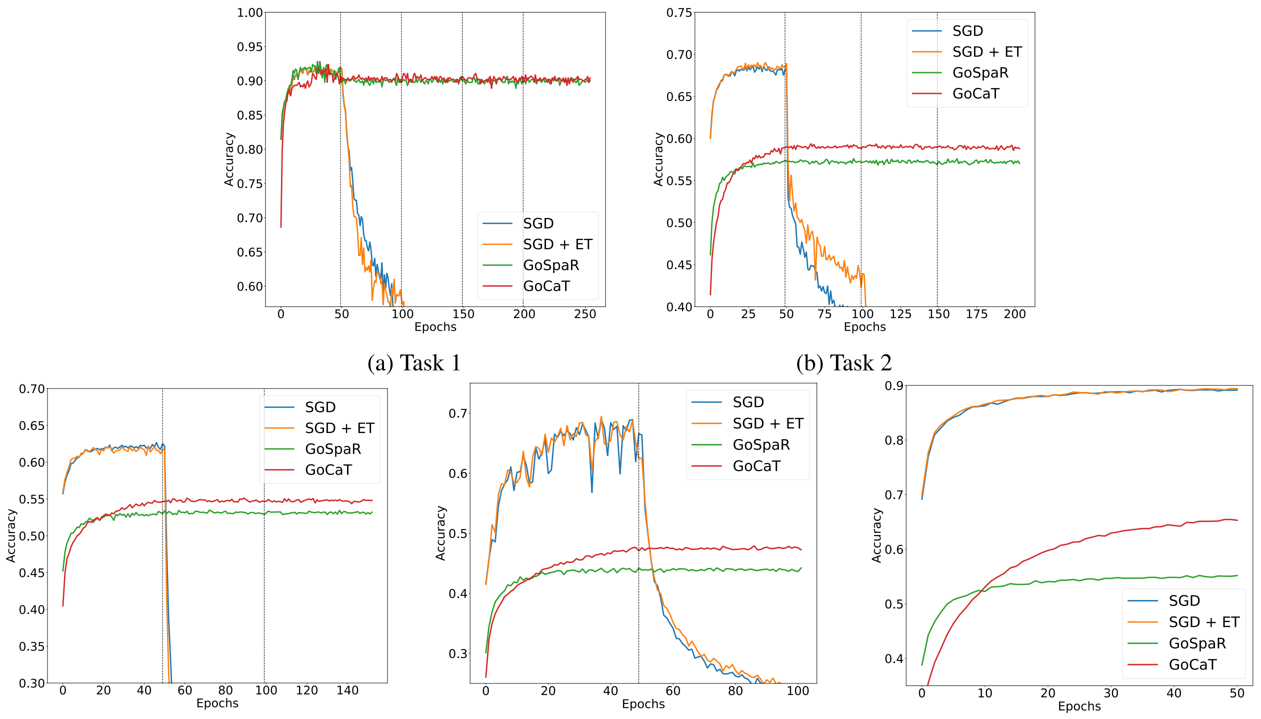
This section compares the effect of introducing each of the three components that we are proposing. As a baseline, we use a model that does not incorporate any technique to avoid forgetting (SGD).

Table 5 resumes the results of our analysis. Each column represents the accuracy obtained for each task at the end of the last training process. As expected, SGD only obtains a good performance when tested in the last task, indicating that it suffers from a drastically forgetting of previous tasks. By applying GoSpaR, we manage to preserve the accuracy of previous tasks, demonstrating its positive effect.

Similar to Table 5, Fig. 2 shows the evolution of the accuracy for individual tasks. The vertical lines in each subfigure indicate the transition to a new task. The abrupt loss of accuracy in SGD after the transition to a new task reflects the catastrophic forgetting. Instead, by adding the GoSpaR, we can preserve performance for trained tasks, showing the effectiveness of the proposed component. Despite the flexibility to learn unused filters, there is a gap between SGD

TABLE 5. **Accuracy obtained by different components that we propose. This table show the performance in each task after training the complete sequence of the 5-Dataset benchmark. The final column indicates the average accuracy.**

	Task 1	Task 2	Task 3	Task 4	Task 5	Acc
SGD	6.8%	14.1%	10.8%	24.0%	89.1%	28.9%
GoSpaR	89.9%	57.0%	53.2%	44.2%	55.2%	59.9%
CaFil	9.8%	16.7%	10.3%	20.2%	89.2%	29.2%
ET	10.2%	8.7%	13.9%	24.1%	**89.4%**	29.3%
GoSpaR + CaFil	91.7%	58.4%	53.8%	46.8%	61.3%	62.1%
GoCaT	**91.9%**	**58.8%**	**54.8%**	**47.3%**	65.3%	**63.6%**



(a) Task 1 (b) Task 2

(c) Task 3 (d) Task 4 (e) Task 5

FIGURE 2. **Each figure represents the accuracy over the epoch for each task. The vertical line denotes the transition to training a new task. In each Task, the first 50 epochs shows the training using data from the current Task, then its shows how the accuracy evolves when training the model in the following tasks.**

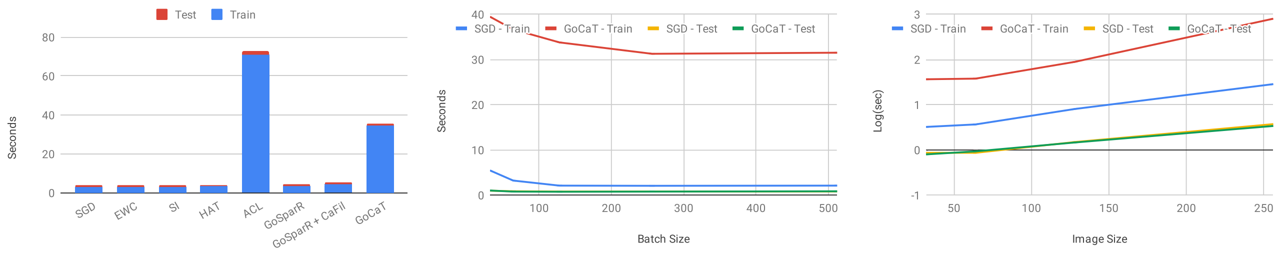
and GoSpaR at the end of the training process for each task, which favors SGD. However, this advantage is quickly lost due to CF.

Our two extra components, CaFil and ET, help reduce the positive gap for SGD over GoSpaR at the end of the training process for each task. Table 5 shows that, used in isolation, CaFil and ET achieve only slightly better results than SGD. However, the goal of these components is to improve communication between frozen and learnable weights, fostering knowledge transfer among tasks. Therefore, when CaFil is applied together with GoSpaR, average accuracy rises about 2%, confirming the advantage of applying this normalization and specialization on the filters. Furthermore, accuracy improves even more when adding ET, reaching an improvement of 4%.

To verify how the combination of components reduces the gap, we check Fig. 2. We can observe that GoCaT improves the accuracy by reducing the gap in all tasks, while at the same time avoiding task interference. Despite not closing the gap completely, we can see that the proposed method encourages knowledge transfer between tasks. By better using what is learned in the past, the model improve accuracy and keep the performance for each task.

2) META-LEARNING STRATEGY

The main goal of adding a meta-learning strategy is to provide model flexibility to learn patterns that can be useful to several tasks. To achieve this, the core of the meta-learning strategy is to select weight values that can be quickly adapted to new tasks. In our experiments, we notice a trade-off between the flexibility provided to the meta-learning iterations and the control of task interference by freezing the values of relevant parameters. Specifically, the model does not learn to adapt previously learned groups of parameters by allowing low flexibility. Conversely, by allowing too much flexibility, the model suffers from CF.



(a) Train and Test Time (b) Batch Size (c) Image Size

FIGURE 3. The time it takes each method in different scenarios. (a) Shows the time in seconds that each method takes to run one epoch in train (blue) and test (red). (b) The number of seconds it takes to train (and test) with different batch sizes. (c) Time in log(seconds) it takes to train (and test) one epoch with different image sizes.

TABLE 6. Performance of GoCaT when we change the number of updates in the outer loop of the meta-learning scheme.

Iterations	10	25	50	100	250
Acc	61.7%	**63.7%**	62.4%	61.5%	60.2%
BWT	-0.0015	**0.0007**	-0.0029	-0.0087	-0.0187

Table 6 shows mean accuracy and BWT scores for GoCaT when we change the number of iterations of the outer loop of the meta-learning strategy. It can be seen that by giving too little flexibility, the model does not learn to adapt previously learned groups of parameters, but by giving too much flexibility, the model suffers from CF.

3) COMPLEXITY

The complexity of the model is related to how long it takes an input to go through the model. This means the amount of time it can take an element, in test time, to go through the model. However, in some methods, the additional complexity is not in the model but in the training process, because of the changes in the training strategy. To check the complexity of our proposal, we carried out several experiments to verify the response times of our method, both in training and test. The results are the average of training each method for 50 epochs with 3 different seeds.

Figure 3a shows the average time in seconds that different methods take in train and test. Regularization methods (EWC, SI, and GoSparR) take similar times to the baseline (SGD), showing that complexity added by regularizations technique does not affect the training process. In contrast, when adding the calibration functions (CaFil), the training process takes slightly longer per epoch. When changing the training strategy, via ACL or ET, the training time goes up dramatically. Nevertheless, despite the increases in training times, the costs of performing inference in GoCaT does not increase.

Taking SGD as a baseline, we perform two more experiments to check the complexity of GoCaT: 1) Change the batch size and 2) Change image size. In the case of changing the batch size, both methods decrease the time it takes to train, as shown in Fig. 3b. SGD decreased 3 seconds between changing the batch size from 32 to 128. On the other hand, GoCaT decreased by almost 8 seconds when changing the batch size from 32 to 256. In both cases, the test time remains similar despite the increase in batch size, showing that the complexity of our proposal is in the training strategy and not in other components.

As shown in Fig. 3c, as we increase the input size, the time it takes to train goes up for both methods. In the case of SGD, it goes up almost 9 times when we resize the images from $32 \times 32 \times 3$ to $256 \times 256 \times 3$ pixels. On the other hand, our proposal increases the training time by almost 21 times. Similar to the batch size experiment, both SGD and GoCaT increase in similar proportions the time during test, confirming that the complexity of our proposal is in the meta-learning strategy.

4) NUMBER OF EPOCHS

The number of epochs shows the number of times the model goes through the whole dataset. In a model that does not suffer from overfitting, with a greater number of epochs, the accuracy of the model should increase. However, training for more epochs brings a higher computational cost. A valid question is how much we can raise the number of epochs so that the benefit in accuracy exceeds the associated computational cost.

Training the model for 25 epochs achieves an accuracy of 58.43% with a BWT 0.002. By training the model for 50 epochs total, the accuracy increase to 63.70%, and keeping a positive BWT. From this point, we notice that the accuracy still increases when training for more epochs (64.47% with 75 epochs and 65.16% with 100 epochs), but the gain in performance is low in comparison with the extra computational resources.

V. CONCLUSION

This paper introduces GoCaT, a new approach to a continual learning scenario that exploits two complementary strategies. The first strategy avoids catastrophic overwriting of weight values by using a group sparse regularization that reserves part of the model to learn each task. The second strategy fosters weight sharing among tasks by using a meta-learning approach to encourage learning of weights that are expected to be useful to solve future tasks.

Our results show that GoCaT is indeed effective to avoid interference across tasks during the sequential learning process. Specifically, our experiments demonstrate that the group sparse coding (GoSpaR) and binary masks are effective to dynamically allocate the network resources in order to avoid CF. Furthermore, by adding a suitable normalization function to the activation maps (CaFil), our model is able to correctly combine knowledge from previous and current tasks outperforming all alternatives approaches by a large margin.

In terms of the use of the meta-learning strategy, we observe a positive impact in knowledge transfer from previous to new tasks. This is clear for the case of 5-Dataset and 20-Split CIFAR100 datasets, where our proposed method achieves positive BWT scores. In contrast, in the case of the P-MNIST dataset, where the sequential tasks do not share common patterns, the BWT score is negative indicating an expected operation of the proposed strategy.

As future work, our findings suggest the relevance of dedicating part of the network resources to learn task-specific knowledge as well as dedicating part of the network to learn global knowledge that can be useful to solve various tasks. In this work, we achieve this using a single network, however, we can think of a strategy that combines several networks that can be jointly trained. As an example, one can have a global network acquiring inter-task knowledge and a set of local networks acquiring intra-task knowledge.

REFERENCES

[1] K. Ahuja, K. Shanmugam, K. Varshney, and A. Dhurandhar, "Invariant risk minimization games," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 145–155.

[2] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 139–154.

[3] R. Aljundi, M. Rohrbach, and T. Tuytelaars, "Selfless sequential learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.

[4] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1–9.

[5] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," 2019, *arXiv:1907.02893*. [Online]. Available: http://arxiv.org/abs/1907.02893

[6] A. D. Baddeley, *Human Memory: Theory and Practice*. London, U.K.: Psychology Press, 1997.

[7] L. Caccia, E. Belilovsky, M. Caccia, and J. Pineau, "Online learned continual compression with adaptive quantization modules," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1240–1250.

[8] W. Cao, Z. Ming, Z. Xu, J. Zhang, and Q. Wang, "Online sequential extreme learning machine with dynamic forgetting factor," *IEEE Access*, vol. 7, pp. 179746–179757, 2019.

[9] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2018, pp. 532–547.

[10] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-GEM," in *Proc. Int. Conf. Learn. Represent.*, Sep. 2019, pp. 1–5.

[11] M. De Lange and T. Tuytelaars, "Continual prototype evolution: Learning online from non-stationary data streams," 2020, *arXiv:2009.00919*. [Online]. Available: http://arxiv.org/abs/2009.00919

[12] P. Dhar, R. V. Singh, K.-C. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5138–5146.

[13] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization," *Proc. Nat. Acad. Sci. USA*, vol. 100, no. 5, pp. 2197–2202, Mar. 2003.

[14] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning in Bayesian neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2019, pp. 1–4.

[15] S. Ebrahimi, F. Meier, R. Calandra, T. Darrell, and M. Rohrbach, "Adversarial continual learning," in *Proc. Eur. Conf. Comput. Vis.*, Mar. 2020, pp. 1–20.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2017 pp. 1126–1135.

[17] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "Remind your neural network to prevent catastrophic forgetting," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020.

[18] X. He, J. Sygnowski, A. Galashov, A. A. Rusu, Y. W. Teh, and R. Pascanu, "Task agnostic continual learning via meta learning," 2019, *arXiv:1906.05201*. [Online]. Available: https://arxiv.org/abs/1906.05201

[19] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira, "Re-evaluating continual learning scenarios: A categorization and case for strong baselines," in *Proc. NeurIPS Continual Learn. Workshop*, 2018, pp. 1–12.

[20] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7132–7141.

[21] W. Hu, L. Zhou, L. Bing, T. Chongyang, T. Zhengwei, M. Jinwen, Z. Dongyan, and Y. Rui, "Overcoming catastrophic forgetting for continual learning via model adaptation," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[22] A. Iscen, J. Zhang, S. Lazebnik, and C. Schmid, "Memory-efficient incremental learning through feature adaptation," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020.

[23] K. Javed and M. White, "Meta-learning representations for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–11.

[24] K. J. Joseph and V. N. Balasubramanian, "Meta-consolidation for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 14374–14386.

[25] J. D. Karpicke. (Jun. 2016). A Powerful Way to Improve Learning and Memory. Psychological Science Agenda. [Online]. Available: http://www.apa.org/science/about/psa/2016/06/learning-memory

[26] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, and D. Hassabis, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, 2017.

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Univ. Toronto, Toronto, ON, Canada, 2009.

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

[29] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4652–4662.

[30] T. Lesort, M. Caccia, and I. Rish, "Understanding continual learning settings with data distribution drift analysis," 2021, *arXiv:2104.01678*. [Online]. Available: http://arxiv.org/abs/2104.01678

[31] T. Lesort, H. Caselles-Dupre, M. Garcia-Ortiz, A. Stoian, and D. Filliat, "Generative models from the perspective of continual learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

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

[33] H. Lobel, R. Vidal, and A. Soto, "Learning shared, discriminative, and compact representations for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 11, pp. 2218–2231, Nov. 2015.

[34] H. Lobel, R. Vidal, and A. Soto, "CompactNets: Compact hierarchical compositional networks for visual recognition," *Comput. Vis. Image Understand.*, vol. 191, Feb. 2020, Art. no. 102841.

[35] V. Lomonaco, D. Maltoni, and L. Pellegrini, "Rehearsal-free continual learning over small non-I.I.D. batches," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 989–998.

[36] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6470–6479.

[37] A. Mallya, D. Dillon, and L. Svetlana, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018.

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

[39] M. Masana, T. Tuytelaars, and J. van de Weijer, "Ternary feature masks: Zero-forgetting for task-incremental learning," 2020, *arXiv:2001.08714*. [Online]. Available: http://arxiv.org/abs/2001.08714

[40] N. Y. Masse, G. D. Grant, and D. J. Freedman, "Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization," *Proc. Nat. Acad. Sci. USA*, vol. 115, no. 44, pp. E10467–E10475, Oct. 2018.
[41] S. A. McLeod. (2008). *Forgetting*. Accessed: Sep. 30, 2019. [Online]. Available: https:www.simplypsychology.orgforgetting.html
[42] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*. [Online]. Available: http://arxiv.org/abs/1803.02999
[43] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? Towards understanding the effectiveness of MAML," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–21.
[44] J. Rajasegaran, S. Khan, M. Hayat, F. S. Khan, and M. Shah, "ITAML: An incremental task-agnostic meta-learning approach," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 13588–13597.
[45] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Represent.*, 2017.
[46] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–11.
[47] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "ICaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2001–2010.
[48] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," in *Proc. Int. Conf. Learn. Represent.*, 2019.
[49] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*. [Online]. Available: http://arxiv.org/abs/1606.04671
[50] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, Jun. 2017.
[51] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4548–4557.
[52] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. NeurIPS*, 2017, pp. 2994–3003.
[53] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 3400–3409.
[54] S. Thrun and L. Pratt, *Learning To Learn*. Springer, 1998.
[55] O. Vinyals, C. Blundell, T. Lillicrap, K. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3637–3645.
[56] J. V. Oswald, C. Henning, J. Sacramento, and B. F. Grewe, "Continual learning with hypernetworks," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–6.
[57] R. Vuorio, D.-Y. Cho, D. Kim, and J. Kim, "Meta continual learning," 2018, *arXiv:1806.06928*. [Online]. Available: http://arxiv.org/abs/1806.06928
[58] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big data*, vol. 3, no. 1, pp. 1–40, 2016.
[59] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1–9.
[60] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3987–3995.
[61] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact CNNs," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 662–677.

JULIO HURTADO received the B.S. degree in computer engineering from Universidad Técnica Federico Santa María, Santiago, Chile. He is currently pursuing the Ph.D. degree in computer science with the Universidad Catolica de Chile, PUC. His research interests include transfer learning, continual learning, meta learning, and generalization in deep learning.

HANS LOBEL (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science engineering from the Pontificia Universidad Católica de Chile, Santiago, Chile. Since 2016, he has been an Assistant Professor with the Department of Transport Engineering and Logistics and the Department of Computer Science, Pontificia Universidad Católica de Chile. His research interests include visual recognition, machine learning, optimization, intelligent transportation systems, and big data.

ALVARO SOTO (Member, IEEE) received the Ph.D. degree in computer science from Carnegie Mellon University, in 2002. Afterwards, he joined the Computer Science Department, Universidad Catolica de Chile, PUC, where he is currently an Associate Professor. His research interest includes studying different aspects behind the creation of cognitive machines.