PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

# AUGMENTING DEEP LEARNING MODELS USING CONTINUAL AND META LEARNING STRATEGIES

**JULIO ANDRÉS HURTADO GONZÁLEZ**

Thesis submitted to the Office of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor in Engineering Sciences

Advisor:
ALVARO SOTO

Santiago de Chile, January 2022

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

# AUGMENTING DEEP LEARNING MODELS USING CONTINUAL AND META LEARNING STRATEGIES

**JULIO ANDRÉS HURTADO GONZÁLEZ**

Members of the Committee:

| | |
|---|---|
| **ÁLVARO SOTO** | DocuSigned by: Alvaro Soto A. DA096D318D8B4F2... |
| **HANS LÖBEL** | DocuSigned by: Hans Lobel D. D9FDE0F134F849F... |
| **DENIS PARRA** | DocuSigned by: [illegible] 0F3E21932A2E493... |
| **MARCELO MENDOZA** | DocuSigned by: Marcelo Mendoza 22A65369D6944C1... |
| **VINCENZO LOMONACO** | DocuSigned by: Vincenzo Lomonaco 85FC68B1A5D34C4... |
| **IGNACIO LIRA** | DocuSigned by: IGNACIO LIRA 4747E55F02D9401... |

Thesis submitted to the Office of Graduate Studies in partial fulfillment of the requirements for the Degree Doctor in Engineering Sciences

Santiago de Chile, January 2022

## ACKNOWLEDGEMENTS

Many people helped me to complete this thesis, for them my total thanks.

## TABLE OF CONTENTS

## LIST OF FIGURES

# LIST OF TABLES

# RESUMEN

Los modelos de aprendizaje profundo son entrenados con conjuntos de datos finitos con una distribución fija, y se prueban en conjuntos que siguen la misma distribución. Este proceso difiere mucho de cómo aprendemos los humanos, donde nos enfrentamos a diferentes situaciones que debemos aprender a resolver continuamente.

Los modelos de aprendizaje profundo no son capaces de adaptarse continuamente a nuevas tareas o situaciones. Cuando un modelo ya entrenado se enfrenta con una nueva tarea, debe ser re entrenado para adaptarse a los nuevos datos. Este entrenamiento modifica completamente los pesos del modelo para enfocarse en la nueva tarea, causando que el modelo olvide lo previamente aprendido. Este problema es conocido como olvido catastrófico, y es el responsable de que el rendimiento de tareas entrenadas previamente baje drásticamente.

En esta tesis nos enfocamos en dos ideas para aliviar el problema del olvido. La primera idea es aprender pesos que favorezcan la transferencia de conocimiento entre tareas, lo que disminuye la necesidad de modificar los pesos del modelo, reduciendo el olvido. La segunda idea es facilitar la reutilización de los pesos del modelo, es decir, entregar herramientas al modelo para que una nueva tarea utilice la información ya adquirida y la complemente con aprendizaje de la propia tarea.

Las dos grandes contribuciones de esta tesis consisten en dos métodos que utilizan estas ideas para aliviar el problema del olvido catastrófico en problemas de aprendizaje continuo. Estas contribuciones muestran que incentivar la reutilización de los pesos es un factor importante para reducir el olvido.

**Palabras Claves**: Aprendizaje Continuo, Meta Learning, Transferencia de Conocimiento, Olvido Catastrófico.

# ABSTRACT

Deep learning models are trained with finite datasets with a fixed distribution and are tested on sets that follow the same distribution. This process differs significantly from how humans learn, as we are faced with different situations that we must continually learn to resolve.

Deep learning models are not capable of continually adapting to new tasks or situations. When an already trained model is faced with a new task, it must be retrained to adapt to the new data. This new training process completely modifies the model weights to focus on the new task, causing the model to forget what was previously learned. This problem is known as catastrophic forgetting, and it is responsible for drastically lowering the performance of previously trained tasks.

In this thesis, we focus on two ideas to alleviate the problem of forgetting. The first idea is to learn weights that favor the transfer of knowledge between tasks, reducing the need to modify the model weights, reducing forgetting. The second idea is to facilitate the reuse of the model weights, that is, to provide tools to the model so that a new task uses the information already acquired and complements it with learning from the task itself.

The contributions of this thesis consist of two methods that use these ideas to alleviate the problem of catastrophic forgetting in continual learning problems. These contributions show that encouraging the reuse of weights is an essential factor in reducing forgetting.

**Keywords**: Continual Learning, Meta Learning, Machine Learning, Knowledge transfer, Catastrophic forgetting.

# 1. 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 (Baddeley, 1997; McLeod, 2008) 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 molding short-term to long-term memories.

In the case of artificial neural networks (ANN), weights accumulate knowledge. When we train a model, an optimizer adjust its weights to minimize an objective function. However, similar to *retroactive interference*, as a model learns a new task, its weights are overwritten, causing the model to forget previously learned knowledge (Rebuffi, Bilen, & Vedaldi, 2017; Masse, Grant, & Freedman, 2018; Lee, Kim, Jun, Ha, & Zhang, 2017; Shmelkov, Schmid, & Alahari, 2017; Serra, Suris, Miron, & Karatzoglou, 2018).

Unfortunately, in the case of ANNs, task interference is more severe than in the case of humans. This problem is evident for the case of continual learning, *i.e.*, the case when data from new tasks is presented sequentially to the learner and does not have access to previous data. Under this training scheme, when a previously trained model is retrained on a new task, it usually suffers a significant drop in performance in the original task. This problem is known as Catastrophic Forgetting (CF) (Zenke, Poole, & Ganguli, 2017; Rusu et al., 2016; Kirkpatrick et al., 2017).

Previous works had focus on three strategies to tackle CF. The first strategy consists of avoiding the modification of parameters that are key to solve previous tasks. Specifically, when facing a new task, a regularization term ensures that critical parameters are modified as little as possible (Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi, Rohrbach, & Tuytelaars, 2019; Aljundi, Babiloni, Elhoseiny, Rohrbach, & Tuytelaars, 2018; Dhar, Vikram Singh, Peng, Wu, & Chellappa, 2019). In general, these approaches show satisfactory performance in problems that involve few tasks. However, problems such as accumulated drifting in weight values and interference make this approach challenging to scale when the number of tasks increases. The second strategy consists of introducing structural changes to the architecture of a model, either by adding new weights to the model, including binary masks with each new task and so forth (Li & Hoiem, 2017; Mallya & Lazebnik, 2018; Mallya, Davis, & Lazebnik, 2018; W. Hu et al., 2019; Serra et al., 2018; Ebrahimi, Meier, Calandra, Darrell, & Rohrbach, 2020). The main problems with these methods are the extra model complexity. The third strategy are Memory-based methods that mitigate CF by inserting data from past tasks into the current training process (Ebrahimi et al., 2021; Rebuffi, Kolesnikov, Sperl, & Lampert, 2017; Chaudhry, Rohrbach, et al., 2019), using it to retrain previous tasks together with the new one continuously. In general, these methods have the best performance. However, they must have access to data from previous tasks, which is not always feasible.

In contrast to these previous strategies, when learning new tasks, humans continuously associate previous experience to new situations, strengthening previous memories, which help to mitigate the forgetting problem (Karpicke, 2016 (accessed June, 2020)). Ideally, we require a model capable of reusing previously acquired knowledge, minimizing the need to modify the weights drastically. Taking inspiration from this mechanism, in this thesis, we propose two methodologies focusing on training reusable knowledge.

As a first contribution of this thesis (Hurtado, Lobel, & Soto, 2021), 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.

By dynamically allocating part of the model capacity to each task, we can separate what was learned into different subgroups of parameters, helping avoid interference. We achieve this by using sparse coding techniques (Donoho & Elad, 2003) to adaptively manage the number of parameters that are used to learn each new task, while keeping the total model capacity fixed. We apply group-sparse regularization to foster groups of parameters that specialize in learning a task. Then, we freeze those parameters that are key to solving a task by adding selectors in the form of binary masks, reserving the rest of the network to learn new tasks.

Regarding knowledge transfer to new tasks, and following the intuition that if a model can learn useful parameters for past and future tasks there would be no need to change its values drastically. We propose a training strategy that fosters learning patterns that can be useful to support several tasks. In particular, in the context of few-shot learning (Ravi & Larochelle, 2017; Finn, Abbeel, & Levine, 2017), metalearning techniques have been used to promote the learning of weights that can be quickly adapted to handle new tasks (Riemer et al., 2019; Chaudhry, Ranzato, Rohrbach, & Elhoseiny, 2019; Raghu, Raghu, Bengio, & Vinyals, 2020). Taking inspiration from these ideas, we propose a metalearning strategy that fosters the learning of weights that can be useful to favor a positive transfer of knowledge between tasks, facilitating the acquisition of continual learning skills.

As a second main contribution of this thesis (Hurtado, Raymond-Saez, & Soto, 2021), we propose MetA Reusable Knowledge or MARK. A new model, based on a metalearning approach that fosters weight reusability among tasks instead of mitigating weight overwriting or learning independent weights for different tasks. In particular, we envision these weights as a common Knowledge Base (KB) that is not only used to learn new tasks, but also enriched with new knowledge as the model learns new tasks. In this sense, the KB behind MARK is not given by an external memory that encodes information in its vectors, but by a *trainable model that encodes shared information in its weights*. As a complementary mechanism to query this KB, MARK also includes trainable masks to execute a selective addressing scheme to query the KB.

Consequently, to build and query its shared KB, MARK exploits two complementary learning strategies. On the one hand, a metalearning technique provides the key mechanism to meet two goals: i) encourage weight updates that are useful for multiple tasks, and ii) enrich the KB with new knowledge as the model learns new tasks. On the other hand, a set of trainable masks provides the key mechanism to selectively choose from the KB relevant weights to solve each task.

In terms of its internal operation, MARK works by first forcing the model to reuse current knowledge via functions that detect the importance of each pattern learned in the KB. Later, it expands its knowledge if past knowledge is not sufficient to perform a task successfully.

In summary, in this thesis, we propose two solutions for the Continual Learning scenario, focusing on finding representations that can be reusable across tasks. We achieve this by providing models with two fundamental tools. First, we train weights that can be transferable between tasks. Second, we encourage the models to reuse previously learned weights by learning functions to select relevant elements from previously learned weights.

## 1.1. Continual Learning

We have mentioned the the problem of Catastrophic Forgetting in Continual Learning scenarios (CL), but what it is CL? Continual Learning studied the problem where a model is trained in a stream of non-stationary datasets or changing environments (Parisi, Kemker, Part, Kanan, & Wermter, 2019; Hadsell, Rao, Rusu, & Pascanu, 2020; Delange et al., 2021). The main goal of this scenario is to have a model that can continuously accumulate knowledge from new data without negatively interfering with what has been learned in the past.

The problem is that Deep Learning models are normally trained in stationary environment, where the distribution of the training data does not change over time. This causes that when faced with a CL scenario, the new tasks modify the weights of the model, which cause *retroactive interference*. As mentioned, this interference can drastically lowers the performance of previously learned tasks by overwriting the weights of the model to learn the new task.

Studying models that can train in CL scenarios can bring several benefits. For example, there are many applications with a constant flow of new data, such as recommender systems or household robots. In the second example, household robots can be trained with with essential and generic functions but it may need to continuously learn new functions or tasks that should be applied in specific situations. Models capable of continuous acquiring new skills can learn this specific functions without forgetting the based jobs they were trained in. Studying this problem can also bring positive consequences in stationary environments by improving learning efficiency, by enabling knowledge transfer between related tasks, or not needing to re-learn with the complete dataset, as new data arrive (Hadsell et al., 2020).

There are several characteristics desired for models in CL scenarios:

(i) Models shall require minimal access to data of previous tasks, since it does not have infinite storage capacity, nor this data are guaranteed to be accessible.
(ii) Negative interference between tasks must be eliminated or reduced as much as possible.
(iii) Models must maintain a degree of plasticity to learn newly available tasks.
(iv) What is learned by the model must be transferable to past and future tasks, seeking that what is learned to be reusable across tasks.

Following previous work on CL (Van de Ven & Tolias, 2019), in this thesis we consider a task incremental scenario, where task identity is provided at training time. Each task $t$ consists of a new data distribution $D^t = (X^t, Y^t, T^t)$, where $X^t$ denotes the input instances, $Y^t$ denotes the instance labels, and $T^t$ is a task ID. This task ID is required during training and inference. The goal is to train a classification model $f : X \longrightarrow Y$ using data from a sequence of $T$ tasks: $D = \{D^1, ..., D^T\}$. Following the usual setup for CL (Van de Ven & Tolias, 2019), each task is presented sequentially to the model. Our scenario allows for unlimited use of data from the current task, but after switching to a new task, this data is no longer available. Furthermore, there is no intersection between the classes of different tasks, but they share a similar domain. This domain intersection is key to take advantage of common patterns among the tasks.

## 1.2. Hypothesis and Objectives

Considering the previous definition, and the limitations of current methods, in this thesis, we proposed the following hypothesis: By using sparse regularizers and training methods that favor generalization, it is possible to obtain a model that learns multiple tasks sequentially without losing performance, and that the generic learned weights can be used across tasks, with the help of functions that provide specialization.

The proposed hypothesis is motivated by three related questions. The first question is the holy grail of questions in CL: How can we mitigate or avoid the problem of CF? Our first objective is to use sparse regularizations to split our model into two groups of parameters: (1) weights that are been trained in the current task, and (2) weights that are set to zero because of the regularization. As proposed in our first method (Hurtado, Lobel, & Soto, 2021), we create binary mask to select does two groups after training a task, this mask helps to freeze does trained weights. Here, we experiment with the trade-off between plasticity and stability by freezing a percentage of the weights.

The second question is: How can we train weights that are usable across tasks? Ideally, previously trained weight are useful for future tasks. Our second objective is to propose new learning strategies to encourage transferable weights. As mentioned, we propose to use meta-learning strategies. It has been shown that models trained with these strategies require slight modifications in their weights to adapt to new tasks in few-shot learning problems (Raghu et al., 2020).

The third question is: How do we encourage the model to reuse previous learned knowledge? It is not enough to have reusable information, but we also need that the model learn to use this transferable weights. Our third objective is to provide these models with the ability to select relevant information for each task, learning to use the information acquired in the past.

The rest of the document is divided into the following sections. In Chapter 2 we present Related Work in the area of Continual Learning and Metalearning. In Chapter 3 and 4 we present our proposals in more details, with their corresponding contributions, methods, and results. Finally, we close the thesis with the conclusion and future work.

# 2. RELATED WORK

## 2.1. Catastrophic Forgetting

A simple approach to avoid Catastrophic Forgetting (CF) is to re-train the model with all the available data, old and new, each time a new task arrives. However, this approach is unfeasible in many situations, for instance, when old data are no longer available. Even when one can store every dataset, 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 continual learning, previous work has followed three main strategies to prevent CF. The first strategy (Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2019, 2018; Dhar et al., 2019) 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 (Kirkpatrick et al., 2017), per-weight uncertainty (Ebrahimi, Elhoseiny, Darrell, & Rohrbach, 2019), among others (Aljundi et al., 2019, 2018; Chaudhry, Ranzato, et al., 2019; Saha & Roy, 2021). Afterward, 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 the 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, make this approach challenging to scale. Alternatives to decrease the interference between tasks are presented in (Lomonaco, Maltoni, & Pellegrini, 2020; Masana, Tuytelaars, & van de Weijer, 2020), here the authors propose to completely freeze previously trained weights eliminating interference but inhibiting information transfer between tasks.

The second strategy consists of introducing structural changes to the architecture of the models (Li & Hoiem, 2017; Mallya & Lazebnik, 2018; Mallya et al., 2018; W. Hu et al., 2019; Serra et al., 2018; Ebrahimi et al., 2020). Works like (Rusu et al., 2016; Fernando et al., 2017) propose 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, learning over a backbone model (Mallya & Lazebnik, 2018; Mallya et al., 2018), or using the Lottery Tickets Hypothesis (Wortsman et al., 2020). A downside of this strategy is that the masks are learned independently of network parameters, leading to suboptimal solutions.

The third strategy is to use Memory-Based methods to recall critical information about previous datasets, either through using elements of previous tasks (W. Hu et al., 2019; Riemer et al., 2019; Rebuffi, Kolesnikov, et al., 2017; Ebrahimi et al., 2021), minimizing gradient interference (Lopez-Paz & Ranzato, 2017; Chaudhry, Ranzato, et al., 2019), or training GANs to generate past elements (Lesort, Caselles-Dupré, Garcia-Ortiz, Stoian, & Filliat, 2019; Shin, Lee, Kim, & Kim, 2017; Hayes, Kafle, Shrestha, Acharya, & Kanan, 2020; Kemker & Kanan, 2018). The main idea is to recreate previous tasks distribution. The main problem with these methods is the need for 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, (Iscen, Zhang, Lazebnik, & Schmid, 2020; L. Caccia, Belilovsky, Caccia, & Pineau, 2020) proposed saving feature vectors. This solution reduces privacy and memory concerns, as these vectors typically require less memory than complete elements but still need 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 (Arjovsky, Bottou, Gulrajani, & Lopez-Paz, 2019; Ahuja, Shanmugam, Varshney, & Dhurandhar, 2020). This scenario is related to the idea of transfer learning (Weiss, Khoshgoftaar, & Wang, 2016), and data stream problems (Cao, Ming, Xu, Zhang, & Wang, 2019). Unlike previous scenarios, the goal of Continual Learning is to acquire new knowledge from new distributions without loss of performance of previously learned task, avoiding interference and CF (Lesort, Caccia, & Rish, 2021).

## 2.2. MetaLearning

Metalearning is the ability of “learning to learn” (Thrun & Pratt, 1998), in other words, the ability to discover proper biases or procedural knowledge that can be used to learn new tasks. In the context of few-shot learning, several works (Lopez-Paz & Ranzato, 2017; Chaudhry, Dokania, Ajanthan, & Torr, 2018; Vuorio, Cho, Kim, & Kim, 2018; Finn et al., 2017; Nichol, Achiam, & Schulman, 2018) have proposed metalearning strategies due to the ability to find features able to adapt to new tasks quickly. As noticed by (Raghu et al., 2020), by training a model with a metalearning strategy, we obtained a good initialization for future meta-tasks (few-shot learning), meaning that with minor modifications to the weights, we can found a proper solution. A popular approach fosters weight values that can be adjusted to model a new task using just a few gradient updates. Taking inspiration from this idea, both of our methods foster knowledge transfer from previous to new tasks based on adapting the methods presented in (Finn et al., 2017; Nichol et al., 2018) to a continual learning scenario.

There are different scenarios in Continual Learning where meta-learning is applied. The first is Meta-Continual Learning, where a model is pre-trained with a metalearning strategy to generate a good starting point, which allow minimizing forgetting in future sequence tasks (Javed & White, 2019; Beaulieu et al., 2020). The second is Continual-Meta Learning, where the objective is to train the model so that it can quickly remember what it has learned in the past through local adaptations (He et al., 2019; Z. Wang, Mehta, Póczos, & Carbonell, 2020). Finally, OSAKA (M. Caccia et al., 2020) mix previous scenarios by needing a model that quickly solves a new task and has fast adaptation.

The scenarios described may be applicable to different environments. In the context of Sequence Learning, (Javed & White, 2019; Rajasegaran, Khan, Hayat, Khan, & Shah, 2020; K J & N Balasubramanian, 2020; Beaulieu et al., 2020) propose different metalearning strategies. The resulting techniques reduce task interference by avoiding conflicts between current and future gradient directions to update weights. Ina more classic Continual Learning environment, some works (M. Caccia et al., 2020; He et al., 2019; Riemer et al., 2019) propose a change in the learning strategy, either by using a meta-model or by combining it with a Memory-Based method, looking to take advantage of the benefits of both techniques.

Others methods use different approaches to combine metalearning and Continual Learning. For example, training prototypes by class has been explored by (De Lange & Tuytelaars, 2020), these prototypes are adapted as new classes arrive, and classification is made using a distance metric. Others train functions as components in the network, either by using Hypernetworks (von Oswald, Henning, Sacramento, & Grewe, 2020), Deep Artificial Neurons (DANs) (Camp, Mandivarapu, & Estrada, 2020), or Compositional Structures (Mendez & Eaton, 2021), so that the network components are more flexible during future training processes.

# 3. OVERCOMING CATASTROPHIC FORGETTING USING SPARSE CODING AND META LEARNING

In this chapter, we present our first proposal to avoid the CF problem. First, we discuss the technical details behind each of the components conforming the proposal. Then, we present the results, followed by an ablation study to better understand the contribution of each component of the proposal.

## 3.1. Method Description

One solution to avoid interference between tasks is to freeze all weights after training the first task. However, by freezing the weights of the model, we lost two fundamental abilities for a continual learning scenario: 1) The flexibility to learn new patterns, and 2) The positive transfer of knowledge between tasks. These restrictions confront us with two challenges: i) How we freeze weights but at the same time keep the flexibility to learn new patterns. ii) How we foster weights to learn new valuable knowledge across tasks.

To address the first challenge, we use a learning strategy that selectively freezes groups of parameters instead of the entire model. Specifically, we propose a regularization mechanism that minimizes the number of parameters used to learn a task. Then, after learning a task, those weights are frozen, avoiding being modified in future training. An advantage of this approach is that it provides flexibility to learn new patterns in those unfrozen weights.

To address the second challenge, we propose a new learning strategy that fosters weight sharing among tasks, focusing on two components. The first component focuses on improving the communication between groups of parameters learned in different tasks, normalizing and weighing the importance of each group depending on the task. The second component is a new learning strategy based on metalearning that fosters knowledge transfer between tasks, by learning weights that can be helpful to future tasks. Next, we describe the details behind each component.

### 3.1.1. 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 on 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.

#### 3.1.1.1. 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:

$$\operatorname*{argmin}_{W,\Theta} \frac{1}{N}\sum_{i=1}^{N} Loss(x_i, y_i; W, \Theta) + R_{wd}(W, \Theta) \qquad (3.1)$$

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

The problem in Equation (3.1) 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:

$$R_{wd}(W, \Theta) = C_{wd}\frac{1}{2}(\|W\|_F^2 + \|\Theta\|_F^2), \tag{3.2}$$

where $\|\cdot\|_F^2$ denotes squared $\ell_2$-norm and $C_{wd}$ is the corresponding regularization constant.

To avoid interference among tasks, we augment Equation (3.1) by including *GoSpaR* over the convolutional layers, as follows:

$$\underset{W,\Theta}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} Loss(x_i, y_i; W, \Theta) + C_{wd}\frac{1}{2}\|W\|_F^2 + C_\Theta \sum_{l=1}^{L-1} \Gamma_l(\Theta^l), \tag{3.3}$$

where $C_\Theta$ is a regularization constant. We define $\Gamma_l$ as:

$$\Gamma_l(\Theta^l) = (1-\beta^l)\frac{1}{2}\|\Theta^l\|_F^2 + \beta^l \sum_{k=1}^{K^l} \|\Theta^l_{k,*}\|. \tag{3.4}$$

The first term in Equation (3.4) corresponds to an $\ell_2$-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 $\beta^l$ 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 (Lobel, Vidal, & Soto, 2015; Zhou, Alvarez, & Porikli, 2016; Alvarez & Salzmann, 2016; Wen, Wu, Wang, Chen, & Li, 2016; Scardapane, Comminiello, Hussain, & Uncini, 2017; Lobel, Vidal, & Soto, 2020). 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.

Figure 3.1 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 (3.5). We use $\ell_2$-norm to decide if a filter is active or not. After training task $t$, filters of layer $l$ whose $\ell_2$-norm is less than threshold $\varepsilon$ are considered as inactive filters, therefore, they are available to be trained by future tasks.

$$M(\Theta^l) = \|\Theta^l_{\cdot,\cdot}\|_2 \leq \varepsilon \tag{3.5}$$

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 (Mallya et al., 2018) and (Serra et al., 2018). The first trains binary masks over a fixed pre-trained model, and the second trains gates functions that output non-binary masks.

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 $\ell_2$-norm greater than $\varepsilon$ 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

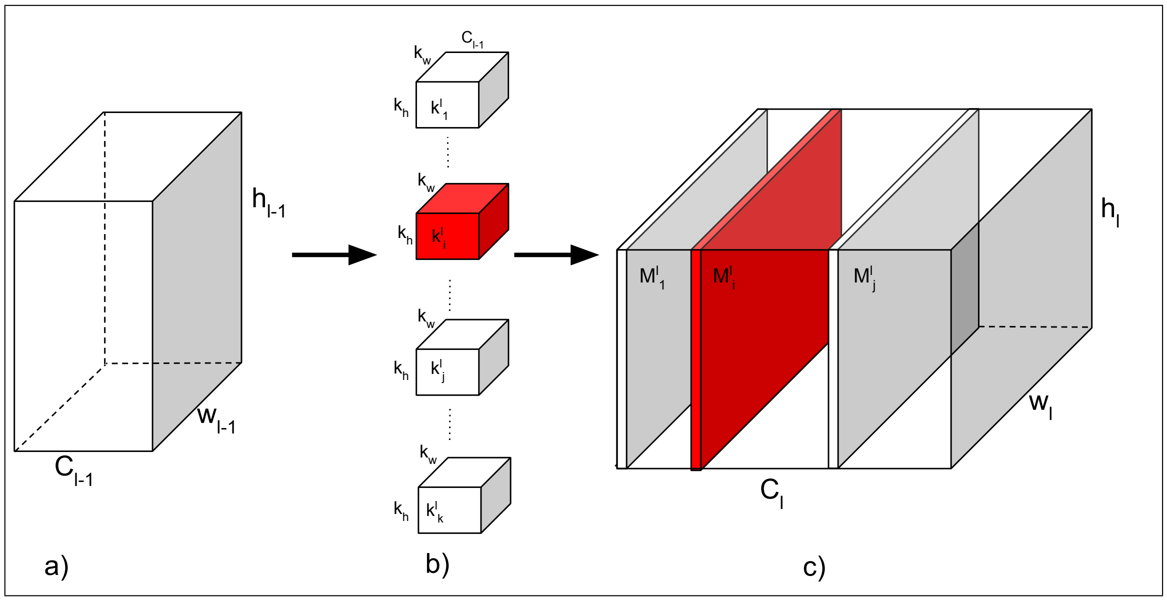


Figure 3.1. (a) Activation maps at convolutional layer $l-1$ of size $h_{l-1} \times w_{l-1} \times C_{l-1}$, (b) At layer $l$, these activation maps are processed by convolutional filters $k_i^l$, $i \in [1, \dots, K]$, of size $k_h \times k_w \times C_{l-1}$. (c) In this case, *GoSpaR* sets filter $k_i^l$ to zero, therefore, the corresponding activation map $M_i^l$ in layer $l$ can be deactivated.

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 (J. Hu, Shen, & Sun, 2018) 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.

#### 3.1.1.2. 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 can not 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 (J. Hu et al., 2018), 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 (3.6). Here, each activation map $M_i^l$ is weighted by the outputs of the normalization function $\Phi_{lt}$, via an element-wise multiplication $(\cdot)$. This multiplication helps to strengthen or weaken the activation maps on $M^l$.

$$O_l(\theta_l, M^l) = M^l \cdot \Phi_{lt}(M^l) \tag{3.6}$$

Function $\Phi_{lt}$ 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:

$$\Phi_{lt}(M^l; W_{tl}^{1,2}) = \sigma(W_{tl}^2 \rho(W_{tl}^1 M^l)), \tag{3.7}$$

where $\rho$ and $\sigma$ represent the ReLU and Sigmoid activation functions, respectively. Weights $W_{tl}^1$ and $W_{tl}^2$ are learned during training.

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

#### 3.1.1.3. Filter training

The training process of a task $t$ is shown in Algorithm 1. For each batch $\{x_i^B, y_i^B\}$, we obtain the corresponding predictions $\hat{y}_i^B$ using the current weights $\Theta$ and $W_t$, i.e., weights of convolutional layers and classification head for task $t$, respectively. Then, given the classification ($Loss$) and the regularization ($\Gamma$) terms, we obtain the gradient for both sets of weights ($g_\Theta, g_{W_t}$). Afterwards, we multiply gradients of $\Theta$ 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 $\alpha$.

```
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 $x_i^B, y_i^B$ in $D_t$ do
      $\hat{y}_i^B = f(x_i^B, W_t, \Theta)$
      # Get gradients
      $g_\Theta, g_{W_t} \leftarrow \nabla(Loss(\hat{y}_i^B, y_i^B) + \Gamma(\Theta))$
      # Freeze used weights
      $g_\Theta \leftarrow m^t \cdot g_\Theta$
      # Update weights
      $\Theta \leftarrow \Theta - \alpha \cdot g_\Theta$
      $W_t \leftarrow W_t - \alpha \cdot g_{W_t}$
```

### 3.1.2. 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 metalearning scenarios (Vinyals, Blundell, Lillicrap, kavukcuoglu, & Wierstra, 2016; Ravi & Larochelle, 2017; Finn et al., 2017). In effect, the ability to generalize across tasks is at the core of metalearning. In this context, (Vinyals, Blundell, Lillicrap, kavukcuoglu, & Wierstra, 2016) propose a metalearning 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, (Finn et al., 2017) propose an optimization method to find network weights that can be quickly adapted to model new tasks. Taking inspiration from (Finn et al., 2017), we adapt its metalearning 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 3.1.1. 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 (Finn et al., 2017), 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 $\Theta_e$, and obtain a new set of parameters $\Theta_s^u$, after iterating $s$ times over mini-task $u$, expressed in Equation (3.8):

$$\Theta_{i+1}^{u} = \Theta_{i}^{u} - \alpha \nabla_{\Theta_{i}^{u}} Loss_{u}(f_{\Theta_{i}^{u}}(x_{u}^{Tr}), y_{u}^{Tr}), \tag{3.8}$$

where $f_{\Theta_i^u}(x_u^{Tr})$ represents the output of the model $f$ with parameters $\Theta_i^u$ 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 $\alpha$ is the learning rate of the inner loop. In our case, the $Loss$ of every mini-task is cross-entropy.

After learning $U$ models, we update parameters $\Theta_e$ 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 metatraining validation set. As shown in Equation (3.9), 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 $\beta$:

$$\Theta_e \leftarrow \Theta_e - \beta \nabla_{\Theta_e} \sum_{u}^{U} Loss_u(f_{\Theta_s^u}(x_u^{Va}), y_u^{Va}) \tag{3.9}$$

By adding the meta-learning strategy, the complete training process of a task is described by Algorithm 2. First, we train the model with Algorithm 1 for a few epochs to adapt it to the new task. Afterward, we start training the model with the meta and traditional training strategy in an alternate way. We repeat this process for a given number of epochs.

It is important to mention that during metatraining, 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 metalearning, *GoSpaR* is not being used, since the objective of the metalearning strategy differs from the goals of the method proposed in Section 3.1.1. 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.

**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, ... 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, ... 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($\Theta_e$)
```

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*.

## 3.2. Experimental evaluation

In this section, 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.

### 3.2.1. 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 (Ebrahimi et al., 2020), which consists of sequentially training a model using data from 5 datasets: CIFAR10, not-MNIST (nMNIST), SVHN, MNIST, and

Table 3.1. 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 |

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 (Krizhevsky & Hinton, 2009), 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 (LeCun, Bottou, Bengio, & Haffner, 1998), where each task is a new random permutation of the pixels in each image. Table 3.1 shows a summary of each dataset.

### 3.2.2. 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 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*) (Kirkpatrick et al., 2017) and Synaptic Intelligence (*SI*) (Zenke et al., 2017). We also compare our approach to Hard Attention to the Task (*HAT*) (Serra et al., 2018) and Adversarial Continual Learning (*ACL*) (Ebrahimi et al., 2020). 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 EWC and SI, we use the implementations described in (Hsu, Liu, Ramasamy, & Kira, 2018). 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 3.10, 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 3.11 indicates how to compute BWT, where $Acc_{t,t}$ indicates the accuracy obtained by task $t$ at the end of training task $t$.

$$Acc = \frac{1}{T}\sum_{t=1}^{T} Acc_{T,t} \tag{3.10}$$

$$BWT = \frac{1}{T-1}\sum_{t=1}^{T-1} Acc_{T,t} - Acc_{t,t} \tag{3.11}$$

### 3.2.3. Implementation details

For all experiments and methods, we use the same 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 blocks are followed by a task specific classification layer. As explained in Section 3.1.1.1, 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 kind of layers. Instead of creating a new architecture, we use similar architecture to the one used in (Ravi & Larochelle, 2017; Finn et al., 2017). In top to the architecture, we add method-specific functions, like the CaFil function described in Section 3.1.1.2, 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 metatraining, we sample $25$ elements and create $5$ classes with them for each mini-task.

As in previous works (Serra et al., 2018; Kirkpatrick et al., 2017), 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.

### 3.2.4. Results

#### 3.2.4.1. 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 3.2 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

Table 3.2. 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 |

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 3.2, 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. 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 3.2, the proposed method uses a similar amount of memory than HAT, which uses gate functions per task.

#### 3.2.4.2. 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 3.3 summarizes 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

Table 3.3. 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 |

is reduced. Again our method outperforms the baselines and alternative approaches. In particular, our method outperforms HAT by $4\%$ in terms of average accuracy and ACL by more than $1\%$. 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.

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.

#### 3.2.4.3. 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 3.4. 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

Table 3.4. 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 |

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.

### 3.2.5. Ablation study

In this section, we perform an ablation study over the main components of our proposal. Furthermore, we also analyze the impact of the metalearning 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.

#### 3.2.5.1. 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 3.5 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 3.5, Figure 3.2 shows the evolution of the accuracy for individual tasks. The vertical lines in each sub-figure 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 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 3.5 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 $4\%$.

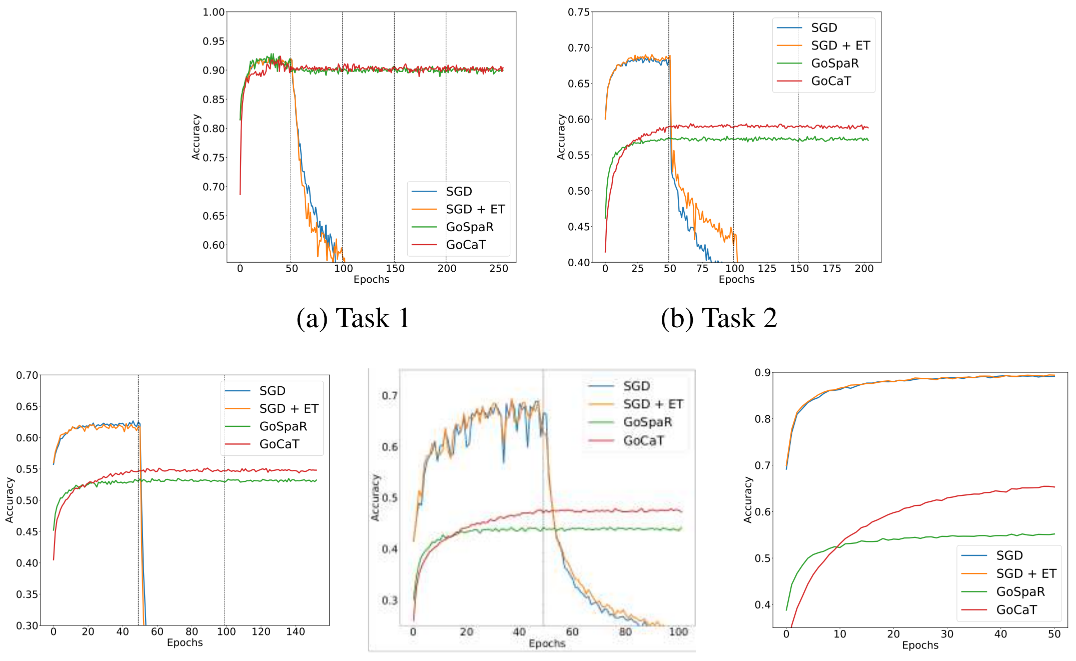
To verify how the combination of components reduces the gap, we check Figure 3.2. We can observe that GoCaT improves the accuracy by reducing the gap in all tasks, while at the same time avoiding task interference. Despite not closing the gap completely, we can see that the proposed method encourages knowledge transfer between tasks. By better using what is learned in the past, the model improve accuracy and keep the performance for each task.

#### 3.2.5.2. Metalearning strategy:

The main goal of adding a metalearning strategy is to provide model flexibility to learn patterns that can be useful to several tasks. To achieve this, the core of the metalearning 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 metalearning iterations and the control of task interference by freezing the values of relevant parameters.

Table 3.5. 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 3.2. 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.

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.

Table 3.6 shows mean accuracy and BWT scores for GoCaT when we change the number of iterations of the outer loop of the metalearning 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.

Table 3.6. Performance of GoCaT when we change the number of updates in the outer loop of the metalearning 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 |

### 3.2.5.3. 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 3.3 shows the average time in seconds that different methods take in train and test. Regularization methods (EWC, SI, and GoSparR) take similar times compare 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.

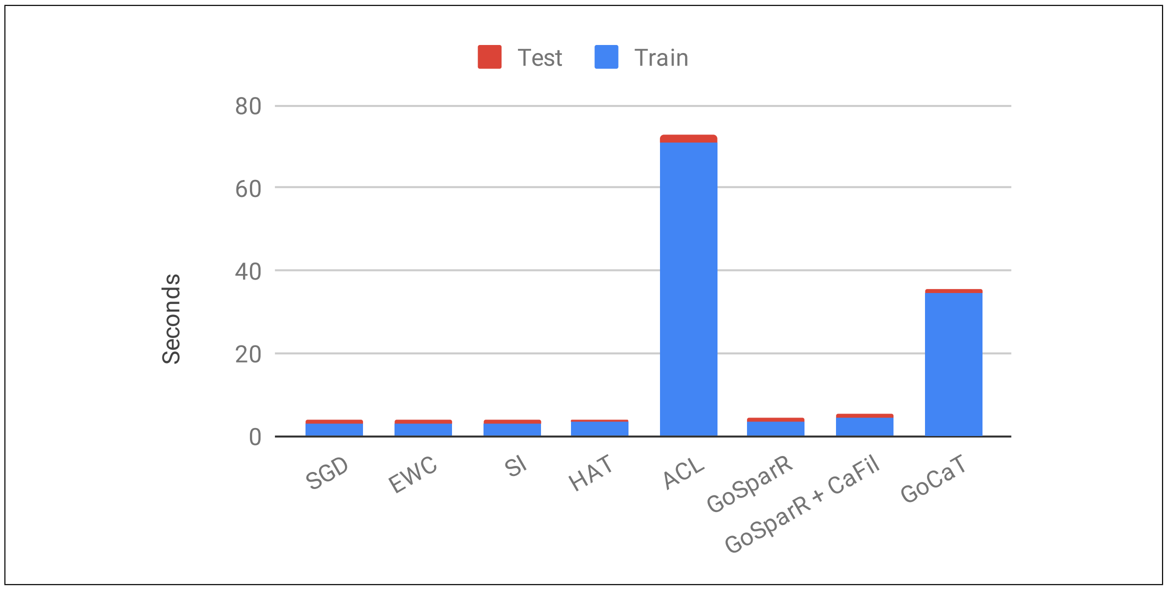


Figure 3.3. Time in seconds that each method takes to run one epoch in train (blue) and test (red)

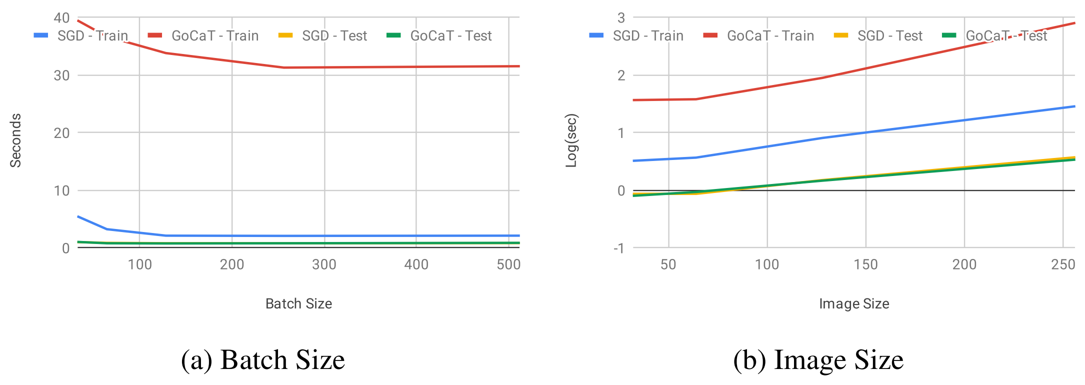


Figure 3.4. The time it takes each method in two different scenarios. **(a)** The number of seconds it takes to train (and test) with different batch sizes. **(b)** Time in log(seconds) it takes to train (and test) one epoch with different image sizes.

Taking SGD as a baseline, we perform two experiments to check the complexity of GoCaT: 1) Change the batch size and 2) Change image size. By changing the batch size, both methods decrease the time it takes to train, as shown in Figure 3.4a. 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 we increase the input size, the time it takes to train goes up for both methods, as shown in Figure 3.4b. In the case of SGD, it goes up almost 9 times when we resize the images from 32x32x3 to 256x256x3 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 metalearning strategy.

#### 3.2.5.4. Number of epochs:

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.

To verify the amount of epochs needed, we training the model for 100 epochs and check the accuracy every 25 epochs. By training for 25 epochs, the model achieves an accuracy of $58.43\%$ with a BWT $0.002$. By adding 25 epochs, 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.

# 4. OPTIMIZING REUSABLE KNOWLEDGE FOR CONTINUAL LEARNING VIA METALEARNING

In this chapter, we present our second proposal to avoid the CF problem. Similar to our previous proposal, our primary motivation is to train weights to foster transferability between tasks. However, in MARK (MetA Reusable Knowledge), we provide complete flexibility to the model to train a task, without freezing weights. The following sections explain how we obtained a model with reusable knowledge and how we can learn to reuse this knowledge for each task. At the end of the section, we present results and an ablation study.

## 4.1. Method Description

When learning tasks sequentially, humans build upon previous knowledge, leading to incremental learning. In contrast, in a similar scenario, ANNs devote all of their resources to the current task, leading to the problem of CF. Taking inspiration from the behavior of humans, an appealing idea is to provide the model with a Knowledge Base (KB) that, as the model faces new tasks, incrementally captures relevant knowledge. Using this shared KB, the model can associate previous experience to new situations, mitigating the CF problem. To implement this idea, we have to address two main challenges: i) How do we build this KB incrementally?, and ii) How do we query this KB to access relevant pieces of knowledge?

To address the first challenge, we leverage metalearning in order to train a KB from data. Specifically, we use a metalearning strategy known as episodic training (Vinyals, Blundell, Lillicrap, Wierstra, et al., 2016). This strategy consists of sampling from a task pairs of support and query sets to simulate training data from a large number of mini-tasks. These mini-tasks are then used to bias the learner to improve its ability to generalize to future tasks. In turn, this generalization leads to our goal: to capture relevant knowledge that can be reused to face new tasks.

To address the second challenge, on top of the KB, we train mask-generating functions for each task. These masks provide a suitable mechanism to selectively access the information encoded in the weights of the KB. We can envision these masks as a query that is used to access the intermediate activations of the KB. Following (Perez, Strub, de Vries, Dumoulin, & Courville, 2018), these masks depend solely on each specific task and input. In this way, given an input, MARK uses the corresponding mask to query the KB generating a feature vector. This vector is then used by a task dependent classification head to output a prediction.

### 4.1.1. Model Architecture

Since we need a trainable KB and a mechanism that can query these weights, we need an architecture composed of several modules. Figure 4.1 shows a schematic view of the operation behind MARK, where the main modules are:

- **Feature Extractor** ($F^t$): This module is in charge of providing an initial embedding for each input $X_i$, i.e., $F^t$ takes input $X_i$ and outputs a vector representation $F_i^t$. In our case, we test our method using visual recognition applications, therefore, $F^t$ is given by a convolutional model. It is important to note that model $F^t$ can be shared among tasks or specific to each task.
- **Knowledge Base** ($KB$)**:** This is the main module behind MARK. It is in charge of accumulating relevant knowledge as the model faces new tasks. Since we work with images benchmarks, we use a convolutional architecture with $B$ blocks. This part of the model is shared across tasks.
- **Mask-Generating functions** ($M^t$)**:** These modules take as an input a feature vector $F_i^t$ and produce a task-dependent mask $M_i^t$ for each block of the KB. Each mask consists of a set of non-binary scalars, one for each channel of the KB blocks, that multiply each channel's activation. These masks are critical to select what knowledge is relevant to each instance and task. In our implementation, we use fully connected layers.

- **Classifier ($C^t$):** These modules correspond to task-dependent classification heads. Its input $F^t_{i,KB}$ is given by a flattening operation over the output of the last block of the KB. Given the task ID of an input $X_i$, the corresponding head outputs the model prediction. In our implementation, we use fully connected layers.

As shown in Figure 4.1, the flow of information in MARK is as follows. Input $X_i$ goes into $F^t$ to extract the representation $F^t_i$. This representation is then used by $M^t$ to produce the set of masks that condition each block in the KB. The same input $X_i$ enters the mask-conditioned KB leading to vector $F^t_{i,KB}$ used by the classification head. Finally, classifier $C^t$ generates the model prediction, where $t$ is the task ID associated to input $X_i$.

### 4.1.2. Training Process

Algorithm 3 describes the training process behind MARK. The first step consists of initializing the KB by training it end-to-end on the first task, without using metalearning and mask functions. In other words, we perform the KB initialization using the regular training procedure of a convolutional neural network for a classification task. After this, we train MARK sequentially on each task by alternating three main steps:

(i) **KB Querying.** We train task-dependent mask-generating functions that are used to query the KB using vector $F^t_i$. Also, we concurrently train the task classifier for the current task. Notice that, leaving aside the KB initialization, in this step each new task is only trained to reuse accumulated knowledge from previous tasks.

(ii) **KB Update**. We use a metalearning strategy to update the weights in the KB. This scheme allows fostering KB updates that favor the acquisition of knowledge that can be reused to face new tasks.

(iii) **KB Querying.** After updating the KB using knowledge from the current task, we repeat the querying process to finetune the mask-generating functions and

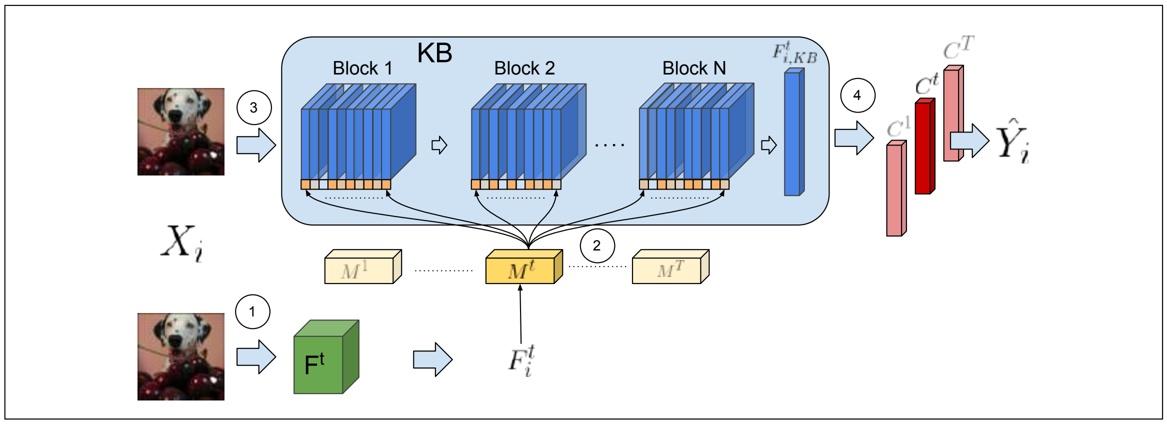


Figure 4.1. Given an input $X_i$ from task $t$, (1) We use a feature extractor $F^t$ to obtain $F^t_i$. (2) $F^t_i$ is then passed to mask function $M^t$ to generate mask $M^t_i$. Afterwards, (3) the same input $X_i$ enters the KB, which has intermediate activations modulated by $M^t_i$. Finally, (4) the modulated features go through a task-dependent classifier $C^t$ that performs the class prediction for $X_i$.

task classifier to use new knowledge. Notice that during this step, the KB is kept fixed.

The intuition behind the application of the three previous steps is as follows. We initially query the KB using the accumulated knowledge from previous tasks. This forces mask functions and classifiers to reuse the available knowledge. When that knowledge is exhausted, we proceed to add knowledge from the current task into the KB. Finally, we take advantage of this newly updated KB to obtain our final mask-functions and classifiers for a given task. We describe next the main steps behind the process to query and update the KB.

#### 4.1.2.1. KB Querying

Once we obtain feature vectors through feature extractor $F^t$, the model can learn which modules in the KB can best solve the current task. In this training stage, the model trains functions to learn how to use the knowledge available in the KB, focusing solely on reusing

```
Algorithm 3: MARK - Training Process
Components:
    • $D^t$: Dataset for task $t$.
    • $F^t$: Feature extractor for task $t$.
    • $KB$: Knowledge Base.
    • $M^t$: Mask-generating function for task $t$.
    • $C^t$: Classifier for task $t$.
$KB \leftarrow Train\{KB(D^1)\}$ /*Init KB */
for task $t \leftarrow 1$ to $T$ do
    /*1. Obtain feature vectors $F_i^t$*/
    $F_i^t \leftarrow \{F^t(D_i^t)\}, i \in \{1, \ldots, |D^t|\}$
    /*2. Train $M^t$ and $C^t$ using current KB*/
    $Train\{M^t$ & $C^t\}$
    /*3. Update KB using metalearning*/
    $KB \leftarrow$ KB-Update   /*Algorithm 4*/
    /*4. Finetune $M^t$ and $C^t$ with updated KB*/
    $Train\{M^t$ & $C^t\}$
end
Output: Trained modules $KB$, $M^t$, $C^t$.
```

```
Algorithm 4: KB-Update
Components:
    • $KB$: Knowledge Base.
    • $C^k$: Temporal Classifier for batch $k$
    • $E_{outer}$: number of training updates for KB (outer loop)
    • $E_{inner}$: number of inner loop epochs.
for e in $E_{outer}$ do
    Generate K batches of data from $D^t$
    for $k$ in $K$ do
        $KB^k \leftarrow KB$ /*Copy of $KB$*/
        $Initialize(C^k)$
        Train $KB^k$ and $C^k$ with batch $k$ for $E_{inner}$ epochs.
    end
    $\nabla KB \leftarrow \frac{1}{E_{inner}} \sum_k^K \gamma_k (KB^k - KB)$
    $S_F \leftarrow KB - \alpha \nabla KB$
end
Output: $KB$   /*output updated $KB$*/
```

knowledge from previous tasks, *without modifying the KB*. In particular, in this step, we only train $M^t$ and $C^t$. Both are trained end-to-end, while keeping the KB weights frozen.

As we generate masks for each intermediate activation of our model, strictly speaking, we have a total of $B$ mask-generating functions. However, to facilitate the notation, we subsume all such functions under the term $M^t$ and think of its output as the concatenation of the results of $B$ functions. Equation 4.1 shows the function $M^t$, where a mask $M_i^t$ is obtained given an input $X_i$ from task $t$. Our implementation of these functions consists of a linear function with parameters $W^{t,M}$, and an activation function $\rho$, which we implement as a ReLU.

$$M_i^t = M^t(F_i^t) = \rho((W^{t,M})^T F_i^t) \tag{4.1}$$

Masks generated in this process are encouraged to have two effects: first, to give a signal of how important a specific module from the KB is for the current input; second, to make sure that gradient updates are done where they really matter. Suppose an activation map is irrelevant for a particular task. In that case, the value of the corresponding mask will be zero, making the gradient update associated with that activation being zero as well.

#### 4.1.2.2. KB Update

The purpose of this training step is to add new knowledge *from the current task* to the KB. As a further goal, we aim to enrich the KB with features that are highly general, i.e., they can be used to solve several tasks. To achieve this, we use metalearning as a way to force the model to capture knowledge that can be reused to face new tasks.

Figure 4.2 shows a schematic view of the metalearning procedure that we use to train MARK. This procedure consists of an adaptation of the training process presented in (Nichol et al., 2018). Specifically, we create a set of $K$ mini-tasks randomly sampling from the current task, where each mini-task consists of a set of $H$ classes and $h$ training instances per class. This procedure allows us to create a mini-task that differs from the main task, finding weights not specific to it. We train one copy of the model using each mini-task for $E_{inner}$ epochs. We refer to the copy $k$ trained for $e$ epochs as $KB_e^k$. For each mini-task we use a temporary classifier $C^k$ initialized with the parameters of $C^t$. We discard this classifier after the final iteration of the inner-loop.

Following (Finn et al., 2017), our method for episodic training consists of two nested loops, an inner and an outer loop. The inner loop is in charge of training the copies of our KB for the current mini-task while the outer loop is in charge of updating the KB weights following a gradient direction that leads to fast adaptation to new mini-tasks. During each inner loop, $KB^k$ and $C^k$ are trained end-to-end for $E_{inner}$ epochs.

To simulate the outer-loop and update the $KB$, we follow Equation 4.2. Specifically, for each $k$, we average the difference between the parameters of the $KB$ before $KB_0^k$ and after $KB_{E_{inner}}^k$ in the inner loop (see Equation 4.2). This is simply an average of the sum of the cumulative gradients for each model $KB^k$. The outer-loop is repeated $E_{outer}$ times.

$$KB = KB - \alpha \nabla KB \qquad \nabla KB = \frac{1}{E_{inner}} \sum_{k}^{K} \gamma_k (KB_{E_{inner}}^k - KB_0^k) \tag{4.2}$$

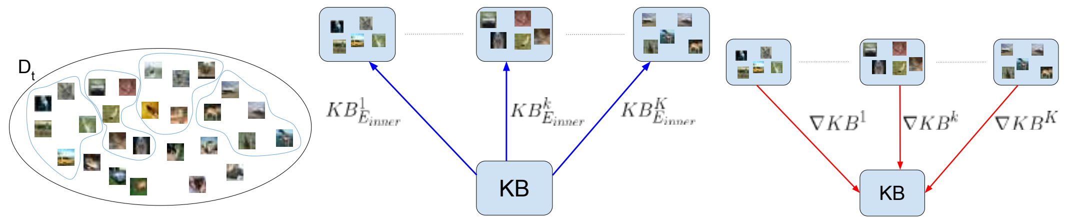
The subtraction in Equation 4.2 is weighted by $\gamma_k$. We compute $\gamma_k$ as shown in Equation 4.3, taking as a reference the accuracy of each model on a validation batch from the same task $t$.

$$\gamma_k = \frac{acc_k}{\sum_j^K acc_j} \tag{4.3}$$

By updating weights with a weighted average of solutions of all meta-tasks, we expect that the new values of the entire model should be (on average) an appropriate solution for different tasks.

## 4.2. Experiments

This section starts discussing benchmarks and baselines used in our experiments. Then, we detail the implementations behind the different methods. Finally, we present



(a) Generate mini-tasks. (b) Train copy of KB in each mini-task. (c) Update KB using gradients.

Figure 4.2. KB update using metalearning. a) Given a task $t$, we randomly generate a set of $K$ mini-tasks, where each mini-task consists of a subset of classes from the original task. b) For each mini-task, we train an independent copy of the current KB for a fixed amount of epochs, leading to $K$ models. c) Afterwards, we calculate gradients with respect to the loss function of each of these models using a hold-out set of training examples. Finally, we update the KB using a weighted average of these gradients.

our results and an ablation study presenting the main contribution of each component of our proposal.

### 4.2.1. Dataset

For our experiments, we use two benchmarks used in previous works (Ebrahimi et al., 2020; Zenke et al., 2017; Chaudhry, Rohrbach, et al., 2019). The first one is 20-Split CIFAR-100 (Krizhevsky & Hinton, 2009) that consists of splitting CIFAR-100 into 20 tasks, each one with 5 classes. The second one is 20-Split MiniImagenet (Vinyals, Blundell, Lillicrap, Wierstra, et al., 2016) that consists of dividing its 100 classes into 20 tasks.

### 4.2.2. Baselines

In our experiments, we compare MARK with recent state of the art methods: Hard Attention to the Tasks (HAT) (Serra et al., 2018), A-GEM (Chaudhry, Ranzato, et al., 2019), Adversarial Continual Learning (ACL) (Ebrahimi et al., 2020), GPM (Saha & Roy, 2021), Experience Replay and SupSup (Wortsman et al., 2020). We also include a Multitask baseline as an upper bound for performance, where all tasks are trained jointly.

Similar to the previous Chapter, to quantify the performance of our method, we use two metrics: average accuracy (Acc) and *backward transfer* (BWT). These are given by:

$$Acc = \frac{1}{T}\sum_{i=1}^{T} Acc_{T,i} \qquad BWT = \frac{1}{T-1}\sum_{i=1}^{T-1} Acc_{T,i} - Acc_{i,i} \tag{4.4}$$

Acc measures average performance over the T tasks after the sequential learning. BWT measures how much performance is lost on previous tasks after sequential learning. As a measure of efficiency, we also consider the amount of memory used by each method, considering number of parameters and extra temporal information needed by each method.

### 4.2.3. Implementation details

We run all of our experiments using 3 different seeds. In terms of hyperparameters, we use SGD with a learning rate of $0.01$ and a batch size of 128. Each task is trained for 50 epochs. To update the KB, we use 10 meta-tasks ($K$), trained for $E_{inner}$ =40 epochs, each with a learning rate of $0.001$. We repeat this training stage 15 times ($E_{outer}$).

We impose no restriction on which models we can use on each part of MARK. To better compare methods, we run all experiments with the same architecture. Specifically, we use a similar architecture as proposed in ACL (Ebrahimi et al., 2020). Main component can be described as:

- **Knowledge Base ($KB$):** In this case, we use 3 shared blocks, each composed of 1 convolutional layer with 64, 128, and 256 components respectively, accompanied by an activation function (ReLU) and Max-Pooling. After these blocks, we add a fully connected layer that creates a representation to be input into the task classifier.

- **Mask-Generating functions ($M^t$):** As explained in Section 4.1.2.1, we use a fully connected layer for each function accompanied by an activation function (ReLU). Input dimensions depend on the dataset, output is $(64 + 128 + 256) = 448$.
- **Classifier ($C^t$):** Is a fully connected layer.

The feature embedding $F^t$, used to represent input instances, can be created by any model, pretrained or otherwise. We test the following approaches:

- **MARK-Task:** we train $F^t$ for each task adding a classifier on top of it that is trained using $D^t$. After training $F^t$, this classifier is discarded. As for the architecture, we use one similar to the private model in (Ebrahimi et al., 2020), consisting of 2 convolutional layers with an activation function, batch normalization, and Max-Pooling, followed by a fully connected layer with ReLU. The number of filters in each layer depends on the dataset used. For CIFAR-100 we use 32 filters per block, while for MiniImageNet we use 8 filters per block.
- **MARK-Random:** $F^t$ consists of a set of random weights that we do not train. All tasks share the same $F^t$. The architecture used is the same as MARK-Task.
- **MARK-Resnet:** all tasks share a Resnet-18 pre-trained on Imagenet as a feature extractor.

### 4.2.4. Results

We first analyze the general performance of our method with respect to the alternative approaches. We highlight the version of MARK where we pre-train $F^t$ using data from each task (MARK-Task). While Table 4.1 shows that MARK-Resnet leads to better results, it uses additional information due to its pre-training using ImageNet. It is important to note that embeddings $F_i^t$ are not directly used to classify the corresponding instances. Instead, they are applied to generate the masks used to query the KB. Aside from its impact on accuracy, we observe no meaningful impact of the input representation on BWT.

Table 4.1. Results using 20-split CIFAR-100 and 20-split MiniImagenet datasets. Standard deviation for 3 runs is listed in parentheses.

| | CIFAR100 | | | Mini-Imagenet | | |
|---|---|---|---|---|---|---|
| Method | Acc (std.)% | BWT% | Mem% | Acc (std.)% | BWT% | Mem% |
| HAT | 76.96 (± 1.2) | **0.01** | 146% | 59.45 (±0.1) | -0.04 | 198% |
| A-GEM | 61.88 (± 0.2) | -16.97 | 205% | 52.43 (±3.1) | -15.23 | 165% |
| ACL | 78.08 (± 1.3) | 0 | 134% | 62.07 (±0.5) | 0 | 195% |
| GPM | 76.67 (± 3.1) | -0.42 | 66% | 60.41 (±0.6) | 0 | 70% |
| Exp. Replay | 65.90 (± 1.2) | -16.9 | 130% | - | - | - |
| SupSup | 77.58 (± 1.3) | 0 | 26/111%[1] | - | - | - |
| Multitask | 84.08 (± 1.2) | 0 | 159% | 74.44 (±1.6) | 0 | 99% |
| MARK-Random | 60.91 (± 0.4) | -1.73 | 41% | 33.02 (±0.7) | **1.29** | 43% |
| MARK-Task | **78.31** (± 0.3) | -0.27 | 100% | **69.43** (±1.6) | -0.39 | 100% |
| MARK-Resnet | 86.29 (± 0.1) | -3.05 | 345% | 93.55 (±0.2) | -2.52 | 126% |

As shown in Table 4.1, for both datasets, MARK-Task outperforms all competing methods in terms of average accuracy, while showing no sign of CF (BWT is close to zero). This is especially remarkable for Mini-Imagenet, as competing methods use complex AlexNet-based architectures during training versus our simple convolutional model. It is also noteworthy that MARK-Task accomplishes this while using almost half less memory than its closest competitor, ACL. This is because MARK-Task reuses the same stored KB for all tasks, while for each task it needs to store a small mask-generating function and classifier. Other methods either need to store extra parameters for adversarial training (Ebrahimi et al., 2020) or they require access to past gradients (Chaudhry, Ranzato, et al., 2019)(Saha & Roy, 2021).

### 4.2.5. Ablation study

#### 4.2.5.1. Evolution of Weight Updates

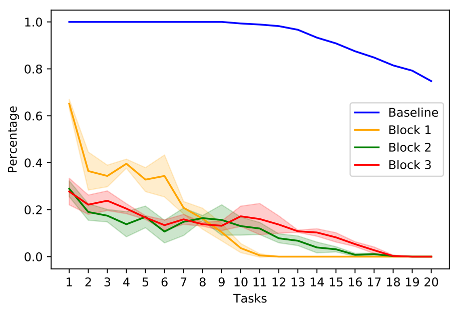
We hypothesize that the positive results achieved by MARK are due to the knowledge stored in its KB being highly reusable. To test this, we analyze the updated weight during

[1]Memory usage of SupSup depends on whether the model is generated from a seed or stored fully.

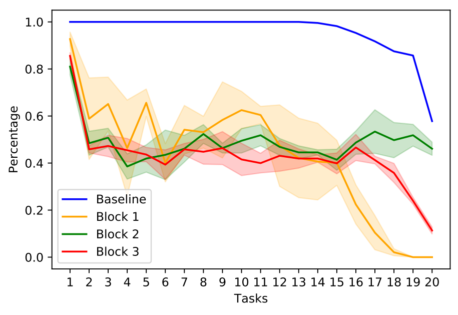
training of all tasks. As a baseline, we consider a version of MARK that does not include metalearning and mask-filtering steps. In this experiment, weights are considered updated if their average deviation is over a certain threshold after training a task.

Figure 4.3 shows the percentage of weights that change after training each task. We observe two distinct phenomena: 1) MARK changes a rather small number of weights compared to the baseline, and 2) As more tasks are trained, the amount of weight updates dwindles.

We attribute both phenomena to MARK's ability to reuse previous knowledge to tackle new tasks. Thus updates should be needed mostly when learning new knowledge. The dwindling amount of updates can be linked to MARK's ability to develop its KB incrementally, thus new tasks are far more likely to be tackled with knowledge already in the KB. We believe that task similarity plays a crucial part in this behaviour, with each new task learned facilitating MARK's ability to find similarities for future tasks. However, to elucidate how important is task similarity to achieve these results is beyond the scope of this study.



(a) 20-Split CIFAR-100

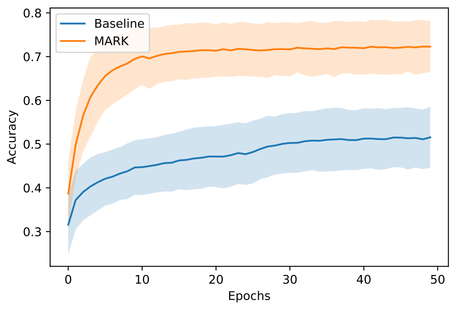


(b) 20-Split MiniImageNet

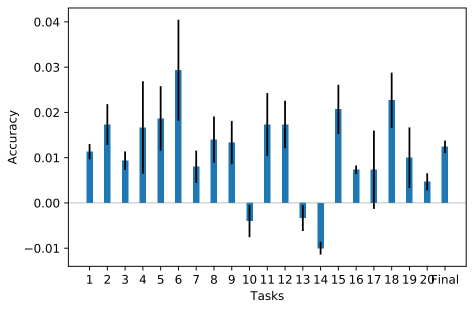
Figure 4.3. Percentage of updated weights in different blocks of the KB during sequential training. As a baseline, we consider a version of MARK that do not include metalearning and mask-filtering. Using MARK fewer updates are required, suggesting that new tasks add less knowledge to the KB due to incremental learning.

#### 4.2.5.2. Task Learning Speed

We hypothesize that if the KB is storing reusable knowledge, then learning speed for new tasks should increase as the KB incrementally contains more knowledge. We analyze the accuracy curves for each task when using MARK versus a model that is sequentially trained without including any mechanism to avoid CF. Given that MARK trains its masks and classifiers two times per task, to be fair, we train the baseline for twice as many epochs as MARK. Consequently, we report results for the baseline every two epochs. Figure 4.4a shows the comparison in speed between the two models. We observe that indeed, on average, MARK achieves higher accuracy and stabilizes quicker than the baseline. This provides further evidence about the ability of MARK to encode reusable knowledge.



(a) Training Speed



(b) Accuracy Gain

Figure 4.4. (a) Average test accuracy in the CIFAR-100 dataset achieved by MARK and a baseline model that is sequentially trained without including any mechanism to avoid CF. MARK achieves both greater accuracy and faster stabilization. (b) Accuracy in a task when we re-training the task after the model complete its sequential learning using all available tasks. We observe an increase in performance for most tasks, suggesting successful knowledge accumulation in the KB.

#### 4.2.5.3. Importance of Metalearning and Mask Functions

We study the impact of the metalearning strategy used to update the KB and the mask-generating functions used to query the KB. To do this, we compare MARK against three ablations:

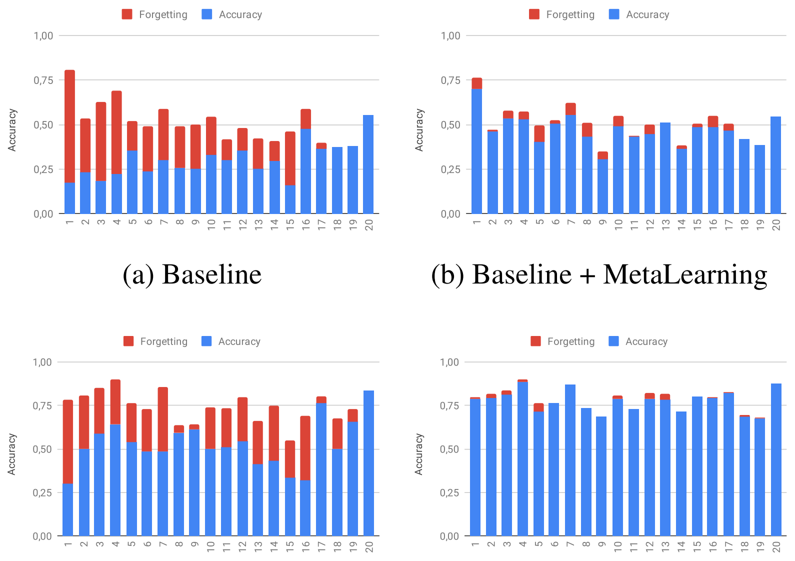
- **Baseline:** Simple sequential learning with no metalearning or mask-generating functions. We use the same architecture as the KB.
- **Baseline + ML:** We improve the baseline by adding metalearning, i.e., KB update.
- **Baseline + Mask:** We improve the baseline by adding task-specific mask functions.

Figure 4.5 shows that the baseline suffers from both significant forgetting and reduced performance. In contrast, when we include metalearning (*Baseline + ML*) forgetting is reduced. Similarly, when we only add mask-functions (*Baseline + Mask*), there is a boost in performance but forgetting also increases. By using metalearning and mask-functions, our full MARK-Task model achieves high accuracy with almost no forgetting.

Two important observations can be extracted from these experiments: 1) The forgetfulness of *Baseline + ML* is more significant than that obtained in MARK ($-3\%$ v/s $-0.25\%$), showing that learning to use prior knowledge through masks can also help reduce forgetfulness. 2) The maximum accuracy averaged over tasks (without forgetting) is higher in MARK than in *Baseline + Mask* ($78\%$ v/s $74\%$), showing that training with metalearning helps knowledge transfer between tasks. This behaviour might seem counterintuitive as we metalearn only using the current task. However, an important part of MARK is forcing the reuse of previously useful features which biases solutions to start from a point which is useful for previous tasks. It is also worthwhile to note that it is known (Nichol et al., 2018) that a REPTILE-style update (as is used in MARK) optimizes for synergistic gradient updates between minitasks. This might also help learn features that are more reusable, which might explain part of the success of MARK.

#### 4.2.5.4. Incremental Construction of KB

We expect that, as MARK faces new tasks, its KB should be enriched with new knowledge. To test this hypothesis, we analyze the effect of training again each task after the model completes its sequential learning using all available tasks. As expected, Figure 4.4b shows that indeed, for most tasks, there is an absolute increase of 1.24% in performance when they are revisited after completing the sequential learning cycle. However, this increase might have to do with relearning what was forgotten rather than learning new knowledge. Thus, we also compared the difference between the maximum accuracy achieved during training for a task versus its final accuracy after retraining. We observed an absolute 0.97% average increase in accuracy as well, which shows that indeed new knowledge was used by earlier tasks.



(a) Baseline (b) Baseline + MetaLearning

(c) Baseline + Masks (d) Full model: MARK-Task

Figure 4.5. CF and accuracy for different versions of MARK-Task tested on 20-Split CIFAR100. (a) Without using metalearning and mask-functions, performance is low and CF is high. (b) Adding only metalearning, performance is still low, but there is almost no forgetting. (c) Adding only mask-functions, performance increases but forgetting is still high. (d) MARK-Task, our full model, achieves high performance with almost no forgetting.

# 5. CONCLUSIONS AND FUTURE WORK

This thesis introduces two novel approaches for continual learning scenarios based on combining different learning strategies. Previous works had focused on minimizing the modification of relevant weights, either by introducing suitable regularization or using structural changes to select relevant information. By reducing the modification of those weights, these works narrow the hypothesis space to learn new tasks, while this avoid weights interference, it only tackle the symptoms of the catastrophic forgetting problem. In this thesis, we focus on learning better representations via new learning strategies. Results show that these strategies encourage learning weights that can transfer their knowledge between tasks, reducing forgetting and improving classification accuracy.

In our first approach, we introduce GoCaT, which exploits two complementary strategies. The first strategy avoids catastrophic overwriting 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 metalearning approach to encourage learning weights that are expected to be helpful to solve future tasks.

Our experiments demonstrate that group-sparse coding (GoSpaR) and binary masks effectively allocate the network resources to avoid catastrophic forgetting. Furthermore, by adding normalization functions (CaFil) and a metalearning strategy (ET), our model correctly combines previous and current knowledge, outperforming alternative approaches by a large margin.

In our second approach we present MARK, which is a novel method for continual learning scenarios based on the construction and query of a KB. This KB is trained via metalearning to accumulate relevant knowledge from different tasks incrementally.

Our experiments indicate that using metalearning to build the KB is crucial to mitigate CF, while using mask-functions to query the KB is crucial to achieve high performance.

Specifically, MARK achieves state-of-the-art results in both 20-Split CIFAR-100 and 20-Split MiniImageNet, while suffering almost no BWT.

## 5.1. Limitations

The above proposals are not exempt from issues and limitations. In the case of GoCaT, we freeze a portion of the model weights in each task, losing some plasticity after training a task. In extreme cases, the model loses the plasticity entirely after training many tasks, which lower the accuracy obtained by the model, as shown in Figure 3.2 when comparing a fully flexible model (SGD) versus GoCaT.

Considering that the group-sparse regularization and the Calibration Functions are only applicable in Convolutional layers, the architecture that be used is limited in GoCaT. This issue limits the scenarios where this method can be applied.

Another limitation in GoCaT is that both learning strategies modify the same set of weights (shared weights), which can increase the risk of interference between both optimization functions. This problem happen because both strategies have different objectives, which helps find a good starting point but can also lead to interference when both objectives are very different. This interference can mainly occur when tasks are very different from each other.

Although several limitations were tackled in our second proposal, as in GoCaT, MARK relies heavily on the assumption that there is a common structure to be learned between tasks. Thus, if there is no common pattern between tasks, a new task will have to modify the KB drastically to obtain good results. Therefore, MARK is sensitive to the quality and structure of the first task.

A relevant limitation present in both methods is that both rely on task-specific modules, which create two problems: (1) the model increase the number of parameters with the number of tasks, and (2) make the methods dependant on the task-id in train and test time.

This limitation hinders the practical use of our proposal since it is not always feasible to have this identifier.

## 5.2. Future work

Given the limitations presented by our proposals and considering new approaches presented in the CL community, many edges remain to be researched further. Next, we will describe the ones that we consider most interesting.

- **Learning Better Representation**: The central hypothesis of this thesis is that by learning representations with reusable information we can mitigate the problem of Catastrophe Forgetting. Results in both proposed methods shown that by applying metalearning we can mitigate the CF. However, other learning strategies that achieve agnostic-to-task representations have also been explored: self-supervised (Jaiswal, Babu, Zadeh, Banerjee, & Makedon, 2021), which is training models with non-manually created labels, or Disentangled Representation (Tenenbaum & Freeman, 1997), which is an unsupervised learning technique. Can we incorporate some of these ideas into new CL scenarios?
  Using different learning strategies, we can find more agnostic-to-task representations. However, we need to find a balance between non-useful agnostic representations and representations that are to specific for a task. For example, a model can be trained with a self-supervised learning strategy and obtained weights that are only useful for an specific self-supervised task. How can we achieve this balance between representations useful across tasks and specific enough to solve the CL scenario?
- **Memory**: One of the important limitations of our two proposed methods, is the need of a Task-ID in train and test time. Both methods have specific functions per task, which are necessary to learn to use the previously learned weights. How can we extend these methods to eliminate the need for the task-ID?

One way to avoid this issue, is to use a function shared between all tasks, but this function will suffer from forgetting if not trained carefully. To mitigate this interference, some methods save a subset of past tasks as memory and added to the current dataset to remember, which reduce forgetting. Nevertheless, these solutions have two major problems: (1) We need access to past elements, which is not always feasible, either for data privacy or other reasons, and (2) We need to dedicate disk space to store this information.

The number of elements stored in this memory can vary depending on the different needs. However, in general terms, the more items we save, the more we can remember, but it also increases the space required. One question that arises from this problem is: Can we better select the elements that we store in this memory? Can we find a subset of the previous tasks that better represents the whole dataset with a minimal number of elements?

- **Generative Replay**: An alternative of saving elements from previous tasks is to generate them, which avoid issues like privacy and space. We can generate new elements by using Generative Models (Shin et al., 2017), VAE (van de Ven, Siegelmann, & Tolias, 2020) to generate elements or latent representations (Hayes et al., 2020). However, is not trivial to train generative models, either because of the high dimensionality of the elements or the interference between task in the generative model.

  An alternative to generative models, are trained representations that compress the information of the whole dataset, either by using Dataset Distillation (T. Wang, Zhu, Torralba, & Efros, 2018) or Dataset Condensation (Zhao, Mopuri, & Bilen, 2020) techniques. These techniques seek to generate a small set of elements that can entirely represent the complete dataset. The problem is that these techniques usually do not work very well with large elements, such as high-dimensional images. Nevertheless, it could be an exciting edge to explore.
- **Efficiency**: One of our motivation to work on Continual Learning is helping in efficiency during training new tasks. This efficiency can happen because we

need slight modifications in the weights to train new data, mainly because we are reusing knowledge that already exists.

We have already explained why it is good to have better representations. However, we can also analyze the sparsity of the weights and in the gradient of the model. Is it always necessary to train the entire model every time a new task arrives? How much do we need to change the weights during training a new task? How do we select which weights can be modified to learn a task and minimize interference?

A few techniques have been proposed in this thesis to tackle how to select relevant knowledge, based on feature-wise transformations. Nevertheless, other techniques may be useful, like those based on Hyper-Networks (von Oswald et al., 2020), which seek to generate parts of the model from a given representations.

- **Shift Detection**: Working with dissimilar tasks is not a limitation exclusive to MARK or GoCaT. If a very different task arrives, more change to the weights are necessary to learn it correctly, which can cause more interference. However, it is not clear how much a task can change before these methods fail, nor is it clear how to measure this difference between tasks.

  Related questions are: When does a distribution drift occur? When does a new class arrive at our continuous flow of data? When is it necessary to learn new patterns or classes? Let imagining a model in real-world environments. Identifying when an element/class that the model does not know appears in our flow, is essential to avoid incorrect classifications. We should be able to train the model to add a new class when needed, identifying drastic change in the distribution, instead of making misclassifications because of unknown classes.

Overall, in this thesis we present two proposals to face the problem of continual learning. We move away from the motivation of traditional solutions as we encourage the transfer of knowledge by training weights that can be useful across tasks. We believe this is a step towards alleviating the catastrophic forgetting, by thinking more about the representations that the model is learning and less about modifying past weights. By doing this, we avoid interference, improve accuracy and increase positive transfer between tasks.

## REFERENCES

Ahuja, K., Shanmugam, K., Varshney, K., & Dhurandhar, A. (2020). Invariant risk minimization games. In *International conference on machine learning* (pp. 145–155).

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., & Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *Proceedings of the european conference on computer vision (eccv).*

Aljundi, R., Rohrbach, M., & Tuytelaars, T. (2019). Selfless sequential learning. In *International conference on learning representations.*

Alvarez, J. M., & Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in neural information processing systems.*

Arjovsky, M., Bottou, L., Gulrajani, I., & Lopez-Paz, D. (2019). Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Baddeley, A. D. (1997). *Human memory: Theory and practice*. Psychology Press.

Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K. O., Clune, J., & Cheney, N. (2020). Learning to continually learn. *European Conference on Artificial Intelligence*.

Caccia, L., Belilovsky, E., Caccia, M., & Pineau, J. (2020). Online learned continual compression with adaptive quantization modules. In *International conference on machine learning.*

Caccia, M., Rodriguez, P., Ostapenko, O., Normandin, F., Lin, M., Page-Caccia, L., ... others (2020). Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*.

Camp, B., Mandivarapu, J. K., & Estrada, R. (2020). Continual learning with deep artificial neurons. *arXiv preprint arXiv:2011.07035*.

Cao, W., Ming, Z., Xu, Z., Zhang, J., & Wang, Q. (2019). Online sequential extreme learning machine with dynamic forgetting factor. *IEEE Access*, *7*, 179746–179757.

Chaudhry, A., Dokania, P. K., Ajanthan, T., & Torr, P. H. (2018). Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the european conference on computer vision.*

Chaudhry, A., Ranzato, M., Rohrbach, M., & Elhoseiny, M. (2019). Efficient lifelong learning with a-GEM. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=Hkf2_sC5FX`

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., & Ranzato, M. (2019). Continual learning with tiny episodic memories. *International Conference on Machine Learning*.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., … Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

De Lange, M., & Tuytelaars, T. (2020). Continual prototype evolution: Learning online from non-stationary data streams. *arXiv preprint arXiv:2009.00919*.

Dhar, P., Vikram Singh, R., Peng, K.-C., Wu, Z., & Chellappa, R. (2019). Learning without memorizing. In *The ieee conference on computer vision and pattern recognition.*

Donoho, D. L., & Elad, M. (2003). Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization. *Proceedings of the National Academy of Sciences*.

Ebrahimi, S., Elhoseiny, M., Darrell, T., & Rohrbach, M. (2019). Uncertainty-guided continual learning in bayesian neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition workshops.*

Ebrahimi, S., Meier, F., Calandra, R., Darrell, T., & Rohrbach, M. (2020). Adversarial continual learning. *European Conference on Computer Vision*.

Ebrahimi, S., Petryk, S., Gokul, A., Gan, W., Gonzalez, J. E., Rohrbach, M., & trevor darrell. (2021). Remembering for the right reasons: Explanations reduce catastrophic forgetting. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=tHgJoMfy6nI`

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., … Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks.

Hadsell, R., Rao, D., Rusu, A. A., & Pascanu, R. (2020). Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*.

Hayes, T. L., Kafle, K., Shrestha, R., Acharya, M., & Kanan, C. (2020). Remind your neural network to prevent catastrophic forgetting. In *European conference on computer vision.*

He, X., Sygnowski, J., Galashov, A., Rusu, A. A., Teh, Y. W., & Pascanu, R. (2019). *Task agnostic continual learning via meta learning.*

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., & Kira, Z. (2018). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In *Neurips continual learning workshop.* Retrieved from https://arxiv.org/abs/1810.12488

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the ieee conference on computer vision and pattern recognition.*

Hu, W., Lin, Z., Liu, B., Tao, C., Tao, Z., Ma, J., … Yan, R. (2019). Overcoming catastrophic forgetting via model adaptation. In *International conference on learning representations.*

Hurtado, J., Lobel, H., & Soto, A. (2021). Overcoming catastrophic forgetting using sparse coding and meta learning. *IEEE Access*.

Hurtado, J., Raymond-Saez, A., & Soto, A. (2021). Optimizing reusable knowledge for continual learning via metalearning. *arXiv preprint arXiv:2106.05390*.

Iscen, A., Zhang, J., Lazebnik, S., & Schmid, C. (2020). Memory-efficient incremental learning through feature adaptation. In *European conference on computer vision.*

Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., & Makedon, F. (2021). A survey on contrastive self-supervised learning. *Technologies*, *9*(1), 2.

Javed, K., & White, M. (2019). Meta-learning representations for continual learning. In *Advances in neural information processing systems.*

Karpicke, J. D. (2016 (accessed June, 2020)). A powerful way to improve learning and memory [Computer software manual].

Kemker, R., & Kanan, C. (2018). Fearnet: Brain-inspired model for incremental learning. In *International conference on learning representations.*

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., ... others (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*.

K J, J., & N Balasubramanian, V. (2020). Meta-consolidation for continual learning. In *Advances in neural information processing systems.*

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., & Zhang, B.-T. (2017). Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems.*

Lesort, T., Caccia, M., & Rish, I. (2021). Understanding continual learning settings with data distribution drift analysis. *arXiv preprint arXiv:2104.01678*.

Lesort, T., Caselles-Dupré, H., Garcia-Ortiz, M., Stoian, A., & Filliat, D. (2019). Generative models from the perspective of continual learning. In *Ijcnn.*

Li, Z., & Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lobel, H., Vidal, R., & Soto, A. (2015). Learning shared, discriminative, and compact representations for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lobel, H., Vidal, R., & Soto, A. (2020). Compactnets: Compact hierarchical compositional networks for visual recognition. *Computer Vision and Image Understanding*.

Lomonaco, V., Maltoni, D., & Pellegrini, L. (2020). Rehearsal-free continual learning over small non-iid batches. In *2020 ieee/cvf conference on computer vision and pattern recognition workshops (cvprw)* (pp. 989–998).

Lopez-Paz, D., & Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Advances in neural information processing systems.*

Mallya, A., Davis, D., & Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights.

Mallya, A., & Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the ieee conference on computer vision and pattern recognition.*

Masana, M., Tuytelaars, T., & van de Weijer, J. (2020). Ternary feature masks: continual learning without any forgetting. *arXiv preprint arXiv:2001.08714*.

Masse, N. Y., Grant, G. D., & Freedman, D. J. (2018). Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*.

McLeod, S. A. (2008). *Forgetting.* https:www.simplypsychology.orgforgetting.html. ([Online; accessed 30-Sept-2019])

Mendez, J. A., & Eaton, E. (2021). Lifelong learning of compositional structures. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=ADWd4TJO13G`

Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

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

Perez, E., Strub, F., de Vries, H., Dumoulin, V., & Courville, A. C. (2018). Film: Visual reasoning with a general conditioning layer. In *Association for the advance of artifical intelligence.*

Raghu, A., Raghu, M., Bengio, S., & Vinyals, O. (2020). Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International conference on* *learning representations.*

Rajasegaran, J., Khan, S., Hayat, M., Khan, F. S., & Shah, M. (2020). itaml: An incremental task-agnostic meta-learning approach. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition.*

Ravi, S., & Larochelle, H. (2017). Optimization as a model for few-shot learning. In *International conference on learning representations.*

Rebuffi, S.-A., Bilen, H., & Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in neural information processing systems.*

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2001–2010).

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., ... Tesauro, G. (2019). Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International conference on learning representations.*

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., ... Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671.*

Saha, G., & Roy, K. (2021). Gradient projection memory for continual learning. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=3AOj0RCNC2`

Scardapane, S., Comminiello, D., Hussain, A., & Uncini, A. (2017). Group sparse regularization for deep neural networks. *Neurocomputing*.

Serra, J., Suris, D., Miron, M., & Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning.*

Shin, H., Lee, J. K., Kim, J., & Kim, J. (2017). Continual learning with deep generative replay. In *Neurips.*

Shmelkov, K., Schmid, C., & Alahari, K. (2017). Incremental learning of object detectors without catastrophic forgetting. In *2017 ieee international conference on computer* *vision (iccv).*

Tenenbaum, J. B., & Freeman, W. T. (1997). Separating style and content. *Advances in neural information processing systems*, 662–668.

Thrun, S., & Pratt, L. (1998). *Learning to learn*. Springer Science & Business Media.

van de Ven, G. M., Siegelmann, H. T., & Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, *11*(1), 1–14.

Van de Ven, G. M., & Tolias, A. S. (2019). Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

Vinyals, O., Blundell, C., Lillicrap, T., kavukcuoglu, k., & Wierstra, D. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems.*

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems.*

von Oswald, J., Henning, C., Sacramento, J., & Grewe, B. F. (2020). Continual learning with hypernetworks. In *International conference on learning representations.*

Vuorio, R., Cho, D., Kim, D., & Kim, J. (2018). Meta continual learning. *arXiv preprint arXiv:1806.06928*.

Wang, T., Zhu, J.-Y., Torralba, A., & Efros, A. A. (2018). Dataset distillation. *arXiv preprint arXiv:1811.10959*.

Wang, Z., Mehta, S. V., Póczos, B., & Carbonell, J. (2020). Efficient meta lifelong-learning with limited memory. *arXiv preprint arXiv:2010.02500*.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, *3*(1), 1–40.

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. In *Conference on neural information processing systems.*

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., & Farhadi, A. (2020). Supermasks in superposition. *Advances in Neural Information Processing Systems*, *33*.

Zenke, F., Poole, B., & Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International conference on machine learning.*

Zhao, B., Mopuri, K. R., & Bilen, H. (2020). Dataset condensation with gradient matching. In *International conference on learning representations.*

Zhou, H., Alvarez, J. M., & Porikli, F. (2016). Less is more: Towards compact cnns. In *European conference on computer vision.*