BERGISCHE
UNIVERSITÄT
WUPPERTAL

Bergische Universität Wuppertal
Fakultät für Elektrotechnik, Informationstechnik und Medientechnik
Institut für Technologien und Management der digitalen Transformation

Hybrid Method for Continual Learning in Convolutional Neural Networks

Basile Tousside, M. Sc. (TUM)

Eine Dissertation, die zur Erlangung des akademischen Grades Doktor der Ingenieurwissenschaften (Dr.-Ing.) im Fachbereich Informatik vorgelegt wurde.

Einreichung: 07 November 2024
Verteidigung: 07 Februar 2025

Abstract

Humans excel at adapting to constantly changing environments, while convolutional neural networks (CNNs) face challenges like catastrophic forgetting in dynamic conditions. Continual learning (CL) aims to address this issue by enabling CNNs to sequentially acquire new knowledge, balancing stability, the retention of past knowledge, and plasticity, the adaptation to new tasks. This ability is essential for systems operating in dynamic environments. However, existing methods often rely on fixed-size models, which struggle to learn diverse behaviors, or growing-size models, which scale poorly as tasks increase.

To address these challenges, this thesis explores strategies for enabling continual learning in CNNs within dynamic, multi-task environments. The focus is on mitigating catastrophic forgetting while maintaining the network's adaptability. A top-down approach is used to define the theoretical foundations of CL, followed by real-world implementation and evaluation. In particular, three scientific contributions are presented:

The **first contribution** addresses the issue of catastrophic forgetting by introducing a novel method to identify crucial CNN filters for previous tasks and restrict their changes during new task training through a regularization term. While traditional approaches typically focus on individual network parameters, this thesis introduces a new filter-level metric to assess importance in task-specific learning. This metric evaluates the significance of each filter based on the average standard deviation of its post-activation values, providing a more granular and effective approach to mitigating forgetting.

The **second contribution** introduces a continual learning framework tailored for fixed-capacity neural networks. A novel sparsification algorithm is proposed to maximize the use of the limited network capacity. Instead of using a dense architecture, the framework leverages a sparse architecture, which enhances the network's plasticity by deactivating unnecessary filters during task training. This sparsity mechanism, embedded in the training process, ensures efficient filter utilization and allows the network to handle multiple tasks without significant performance degradation.

The **third contribution** extends the framework from the second contribution by removing the constraint of fixed capacity, enabling the network to learn an infinite number of tasks. A heuristic mechanism is introduced to detect when the network's capacity saturates, triggering an expansion of the network when necessary. This expansion is managed through sophisticated techniques that ensure the newly added neurons and filters do not disrupt the network's functionality. Additionally, mechanisms are developed to identify saturated layers and augment them with additional filters or neurons, ensuring efficient and targeted network expansion.

Kurzfassung

Der Mensch ist in der Lage, sich hervorragend an eine sich ständig verändernde Umgebung anzupassen, während Convolutional Neural Networks (CNNs) mit Herausforderungen wie dem katastrophalen Vergessen unter dynamischen Bedingungen konfrontiert sind. Das Ziel von Continual Learning (CL) ist es, dieses Problem zu lösen, indem CNNs die Fähigkeit erhalten, sequentiell neues Wissen zu erlernen. Dabei wird ein Gleichgewicht zwischen Stabilität, d.h. der Beibehaltung des zuvor erworbenen Wissens, und Plastizität, d.h. der Anpassung an neue Aufgaben, angestrebt. Diese Fähigkeit ist für Systeme, die in dynamischen Umgebungen betrieben werden, von entscheidender Bedeutung. Bestehende Methoden basieren jedoch häufig auf Modellen mit fester Größe, die nur schwer unterschiedliche Verhaltensweisen erlernen können, oder auf Modellen mit wachsender Größe, die nur unzureichend an wachsende Aufgaben angepasst werden können.

Die vorliegende Arbeit widmet sich der Untersuchung von Möglichkeiten zur Implementierung von Continual Learning in CNNs in dynamischen Multitasking-Umgebungen. Der Schwerpunkt liegt dabei auf der Vermeidung des katastrophalen Vergessens bei gleichzeitiger Erhaltung der Anpassungsfähigkeit des Netzes. Zu diesem Zweck wird zunächst ein Top-Down-Ansatz verwendet, um die theoretischen Grundlagen von CL zu definieren. In einem zweiten Schritt werden praktische Implementierungen und Evaluationen vorgestellt. Insbesondere werden drei wissenschaftliche Beiträge präsentiert:

Der **erste Beitrag** befasst sich mit dem Problem des katastrophalen Vergessens und stellt eine innovative Methode vor, um wichtige CNN-Filter für vorherige Aufgaben zu identifizieren und ihre Veränderung während des Trainings neuer Aufgaben durch einen Regularisierungsterm zu begrenzen. Traditionelle Ansätze konzentrieren sich in der Regel auf einzelne Netzparameter. Im Gegensatz dazu wird in dieser Arbeit eine neue Metrik auf Filterebene eingeführt, um die Relevanz des aufgabenspezifischen Lernens zu bewerten. Die Wichtigkeit jedes Filters wird anhand der durchschnittlichen Standardabweichung seiner Post-Aktivierungswerte, bewertet, wodurch ein detaillierterer und effizienterer Ansatz zur Verhinderung des Vergessens gewährleistet wird.

Im **zweiten Beitrag** wird ein Framework für kontinuierliches Lernen vorgestellt, das für neuronale Netze mit fester Kapazität geeignet ist. In diesem Beitrag wird ein neuartiger Sparse-Algorithmus vorgestellt, der darauf abzielt, die begrenzte Kapazität des Netzes optimal zu nutzen. Im Gegensatz zu einer dichten Architektur basiert das Framework auf einer sparsamen Architektur, wodurch die Plastizität des Netzes erhöht wird. Dies wird erreicht, indem nicht benötigte Filter während des Trainings der Aufgabe deaktiviert werden. Dieser in den Trainingsprozess integrierte Mechanismus gewährleistet eine effiziente

Nutzung der Filter und ermöglicht es dem Netz, mehrere Aufgaben ohne wesentliche Leistungseinschränkungen zu bewältigen.
Der **dritte Beitrag** erweitert den Rahmen des zweiten Beitrags, indem die Beschränkung auf eine feste Netzkapazität aufgehoben wird. Dadurch kann das neuronale Netz eine unbegrenzte Anzahl von Aufgaben erlernen. Es wird ein heuristischer Mechanismus eingeführt, der erkennt, wann die Kapazität des Netzes gesättigt ist. Bei Bedarf wird eine Erweiterung des Netzes angestoßen. Die Erweiterung des Netzes wird durch innovative Techniken gesteuert, die sicherstellen, dass die neu hinzugefügten Filter die Funktionalität des Netzes nicht beeinträchtigen. Darüber hinaus werden Mechanismen entwickelt, um gesättigte Schichten zu identifizieren und diese mit zusätzlichen Filtern zu ergänzen. Dadurch kann eine effiziente und zielgerichtete Erweiterung des Netzes gewährleistet werden.

Forschungsarbeiten

Die in dieser Dissertation vorgestellten Forschungsergebnisse führten zu mehreren begutachteten Veröffentlichungen. Die wichtigsten Publikationen sind im Folgenden aufgeführt:

1. [1] **Basile Tousside***, Janis Mohr, Jörg Frochte. Group and Exclusive Sparse Regularization-based Continual Learning of CNNs. 35th Canadian Conference on Artificial Intelligence, Canadian AI 2022, Toronto, 30 May - 3 June 2022. **Best Paper Award**.

2. [2] **Basile Tousside***, Jörg Frochte, Tobias Meisen. CNNs Sparsification and Expansion for Continual Learning. 16th International Conference on Agents and Artificial Intelligence, ICAART 2024, Rome, Italie, 24-26 February 2024.

3. [3] **Basile Tousside***, Lukas Friedrichsen, Jörg Frochte. Towards Robust Continual Learning using an Enhanced Tree-CNN. 14th International Conference on Agents and Artificial Intelligence, ICAART 2022, Online Streaming, 3-5 February 2022.

4. [4] **Basile Tousside***, Jörg Frochte, Tobias Meisen. Dynamic Capacity Expansion in Continual Learning: The eSECL Approach. Lecture Notes in Artificial Intelligence 2025.

5. [5] Janis Mohr*, **Basile Tousside**, Marco Schmidt, Jörg Frochte. Multiple Additive Neural Networks: A Novel Approach to Continuous Learning in Regression and Classification. 15th International Joint Conference on Computational Intelligence, IJCCI 2023, Rome, Italy, 13-15 November 2023.

6. [6] Janis Mohr*, **Basile Tousside**, Marco Schmidt, Jörg Frochte. Explainability and Continuous Learning with Capsule Networks. 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2021, Online Streaming, 25-27 October 2021.

Acknowledgements

The research presented in this dissertation was carried out during my tenure as a member of the academic staff at the Chair of Applied Computer Science & Mathematics at Bochum University of Applied Sciences, in collaboration with the Institute for Technologies and Management of Digital Transformation at the University of Wuppertal. My research was partially sponsored by the State of North Rhine-Westphalia and the European Regional Development Fund FKZ under the reference ERFE-040021.
Reflecting on the journey that culminated in this PhD thesis, I have come to appreciate the substantial support and collaboration that made this achievement possible.
First and foremost, I would like to extend my deepest gratitude to my supervisors, Prof. Jörg Frochte (Bochum University of Applied Sciences) and Prof. Tobias Meisen (University of Wupertal), for their invaluable guidance, insightful feedback, and constant encouragement throughout this journey. I feel particularly privileged to have benefited from their mentorship. Their expertise and dedication have played a pivotal role in shaping my research and driving it to completion.
I would also like to thank my colleagues for their constructive discussions, collaboration, and the supportive work environment they fostered. In particular, Christof Kaufmann, Janis Mohr and Ana Martinez provided valuable perspectives that significantly shaped the direction and development of my work.
A special thanks to the State of North Rhine-Westphalia and the European Regional Development Fund for their financial support, which enabled me to pursue this research. Their trust in the project's potential was a significant source of motivation.
On a personal level, I am immensely grateful to my family and friends whose constant support and understanding have carried me through the challenges of this Ph.D. journey. Their patience, love, and encouragement provided the foundation I needed to keep going, especially during the most difficult moments.
Finally, I would like to acknowledge all those who contributed, directly or indirectly, to the successful completion of this thesis. Your support has been greatly appreciated, and I owe much of my success to the many people who helped and encouraged me along the way. Thank you all.

Contents

Notation

Abbreviations

Abbreviation	Description	Definition
IID	Independently and Identically Distributed	Page 1
CNN	Convolutional Neural Network	Page 2
DNN	Deep Neuronal Network	Page 2
CF	Catastrophic Forgetting	Page 2
CL	Continual Learning	Page 2
ANN	Artificial Neural Network	Page 3
GPDR	General Data Protection Regulation	Page 3
AI	Artificial Intelligence	Page 4
EWC	Elastic Weight Consolidation	Page 5
FIM	Fisher Information Matrix	Page 5
MLP	Multilayer Perceptrons	Page 11
GD	Gradient Descent	Page 14
MTL	Multi-task Learning	Page 24
MAS	Memory Aware Synapse	Page 29
PNN	Progressive Neural Network	Page 35
ACC	Average Classification Accuracy	Page 44
AMCA	Average Mean Classification Accuracy	Page 44
F	Average Forgetting	Page 44
FWT	Forward Transfer	Page 44
GESCL	Group and Exclusive Sparse Regularization-based Continual Learing	Page 47
PGD	Proximal Gradient Descent	Page 58
SECL	Sparsification and Expansion-based Continual Learning	Page 77

Mathematical notation

x	vector
$\bar{x}$	mean of vector x
$\hat{x}$	estimated/predicted value of x
$\widetilde{x}$	binary version of vector x
$\mathbf{x}$	matrix
$\mathbb{R}^n$	Euclidean space of dimensionality n
$x \cdot y$	dot product between x and y
$\otimes$	element-wise multiplication
$\odot$	multiplication with broadcasting
$*$	convolution operation
x_i	i-th component of vector x
$\sigma(\mathrm{x})$	standard deviation of array x
$\|\mathbf{x}\|_2$	euclidean norm of array $\mathbf{x}$
∇_x	gradient with respect to x
$\mathrm{prox}_{\alpha\ h}$	proximal gradient descent for a function h scaled by α
Θ	overall neural networks parameters
$f(\cdot;\Theta)$	neural network parameterized by Θ
$\mathcal{L}$	overall loss function of the neural network model
l_i	loss value for an individual training sample i

Chapter 1

Introduction

1.1 Motivation

Humans exhibit a remarkable ability to acquire and remember many different tasks that they encounter throughout their lives. In some cases, they even leverage previously acquired knowledge to speed up and improve the learning of new tasks. For instance, learning to ride a motorcycle is much easier if one has prior experience riding a bicycle. In contrast, neural networks, although biologically inspired, excel mainly in static environments that rely on the strong assumptions that (i) all data are available at the same time and (ii) the data follow an independent and identically distributed (IID) pattern. These assumptions are violated in many real-world applications that either deal with non-stationary data distributions or involve scenarios where the data are not all available at the same time.

For example, millions of images with new tags appear every day on social media platforms such as Meta, Instagram, Pinterest, Flickr, or even X (formerly Twitter). Similarly, hundreds of hours of video are either uploaded or live streamed every minute on online video sharing platforms such as YouTube, TikTok, or Twitch. In both cases, this continuously available new content contains novel topics and trends (i.e., domain shifts) that may be significantly different from previous ones, making it challenging for a previously trained neural network to adapt to this fresh, non-stationary, and non-IID data stream.

To efficiently analyze the effects of non-stationary data distribution, the data stream can be empirically divided into several parts, referred to as tasks, which may be temporally separated or represent a single task undergoing temporal changes. In Figure 1.1, for example, the first task involves distinguishing between dogs and cats, whereas the second task focuses on differentiating between flowers and fungi. This partitioning allows for the observation of learning and forgetting behavior as a new task is learned. A task often refers to a particular period of time where the data distribution may (but not necessarily) be stationary and the objective function (Equation 1.1) is constant:

$$\mathcal{L}^t(\Theta^t, D^t_{train}) = \frac{1}{m^t} \sum_{i=1}^{m^t} l_i \left(f \left(x^t_i, \Theta^t \right), y^t_i \right), \tag{1.1}$$

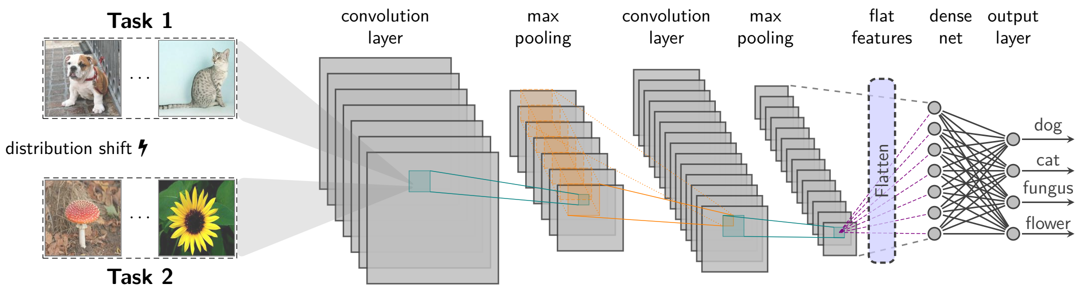


Figure 1.1: A convolutional neural network (CNN) is trained sequentially on *task 1* (distinguishing between dogs and cats) and *task 2* (differentiating between flowers and fungi), with the distribution of data changing as *task 2* introduces new classes. After learning *task 1*, if the CNN is naively trained on *task 2*, it overwrites its weights, thus catastrophically forgetting how to perform *task 1*. This catastrophic forgetting of *task 1* occurs because the features that are important for the first task (e.g., fur texture, ear shape) may not be relevant for the *task 2*, which may focus on features such as petal shape or color. This change in relevant features from one task to the next is an example of distribution shift.

where t is the task index, $D^t_{train} = \{(x^t_i, y^t_i)\}^{m^t}_{i=1}$ its training data, Θ is the model parameter, f is the neural network model, l_i is the loss function (typically cross-entropy for classification) of the i^{th} training sample and m^t is the number of samples for task t.

In this thesis, a task is thus represented by a pair of data $(x^t_i, y^t_i)^{m^t}_{i=1}$, where x represents the training samples and y their corresponding labels, as is generally the case in *supervised learning*, which is the focus of this thesis. Specifically, this thesis deals with *image classification*, a critical component in many real-world applications such as medical diagnostics, autonomous driving, or facial recognition. In Figure 1.1, for example, the first task is mathematically represented by matrices $\mathbf{X}^1 = (x^1_1, \dots, x^1_{m^1})$ of cats and dogs, while the second task is represented by matrices $\mathbf{X}^2 = (x^2_1, \dots, x^2_{m^2})$ of fungi and flowers.

In such scenarios, where (i) the data arrives sequentially in the form of tasks, (ii) there is a domain shift between the tasks, i.e. a change in data distribution from one task to another (as shown in Figure 1.1), (iii) the deep neural network (DNN) model cannot access all task data simultaneously due to constraints such as memory or privacy issues (discussed later in this section), and therefore focuses its learning exclusively on the data of the current task, a performance drop occurs on the old tasks, a phenomenon referred to as *catastrophic forgetting* (CF) [7, 8]. Overcoming catastrophic forgetting while limiting the model capacity, computational cost, and memory footprint is the focus of *continual learning* (CL), also referred to as lifelong, incremental or sequential learning [9, 10].

At the heart of catastrophic forgetting is the so-called *stability-plasticity* dilemma [11, 12], which describes the problem of adapting to non-stationary data while at the same time preventing forgetting and is a common issue for artificial neural networks. In fact, catastrophic forgetting occurs because as the model learns from the current task's data, it tends to overwrite or neglect what it has previously learned. This issue highlights the critical balance between retaining previously acquired knowledge and adapting to new information, especially when data and task characteristics evolve.

Hence, when an artificial learning model is presented with data in sequence, it needs (i) stability to remember how to solve earlier tasks without retraining on previous data while it also requires (ii) plasticity to acquire new knowledge i.e., to learn new tasks. If the model is trained without any immunity against the forgetting of past tasks, therefore focusing on the current task only, it will be very plastic but not stable, meaning that it can learn fast but also forgets quickly. On the other hand, if the network focuses mainly on being immune to forgetting it may lack plasticity i.e., sufficient capacity to learn future tasks.

Overall, continual learning consists of incrementally adapting a model to learn diverse tasks (defined according to the problem at hand) that are observed sequentially. Continual learning has two main objectives: maintaining long-term memory (remembering how to solve previous tasks) and continually navigating new experiences (quickly adapting to new tasks). This makes it a versatile tool with applications in a wide range of domains, such as classification [13, 14], image generation [15, 16], speech recognition [17, 18], reinforcement learning [19, 20], clinical application [21, 22], or language learning [23, 24].

Progress in continual learning could yield tremendous benefits for neural networks. In the remainder of this section, three scenarios are presented where continual learning is highly required.

Online Learning In an online learning setting, data arrives on the fly, requiring the model to adapt to new data. For example, consider an artificial neural network (ANN) deployed in a shopping store and trained to distinguish between pullovers and bags (Task 1). When training on new data, such as distinguishing sandals from shirts (Task 2) – representing new products to be available in the store – a standard neural network retrained exclusively on Task 2, under the assumption that Task 1 data are unavailable, will overwrite its existing weights. This leads to catastrophic forgetting, meaning it loses the ability to perform Task 1. Here, continal learning is desired so that the model performs well on both Task 1 and Task 2.

If there is no restriction on storing or regenerating data from past tasks, the model learner could use a so-called rehearsal strategy, which consists in storing the most representative data from Task 1 and using them jointly with data from Task 2 during the training phase for Task 2. Since the model is thus trained with data from both tasks, it will perform well on both Tasks 1 and 2 simultaneously. However, as will be seen in the next paragraph, data from older tasks may not be allowed to be stored over time.

Data Privacy The use of machine learning is increasingly subject to modern data protection and privacy laws which impose strict standards, including the need to delete collected data either under certain conditions or after a certain period of time. A common and prominent component of these data protection and privacy legislations is the “right to be forgotten”, also known as the “right to erasure”, which allows individuals to request the deletion or removal of their personal data from an organization’s records. Prominent examples include:

- The European Union’s General Data Protection Regulation (GDPR), implemented in May 2018, and the United Kingdom’s post-Brexit adaptation of GDPR in January

2021, both of which, in their Article 17, state: "*The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay.*"

- The California Consumer Privacy Act (CCPA), effective from January 2020, specifies in Section 1798.105 that: "*Consumers have the right to request that a business delete any personal information about the consumer which the business has collected from the consumer.*"
- Canada's Bill C-27, revised in June 2022, articulates in Section 55 that: "*If an organization receives a written request from an individual to dispose of their personal information that is under the organization's control, the organization must, as soon as feasible, dispose of the information, if (a) the information was collected, used or disclosed in contravention of this Act; (b) the individual has withdrawn their consent, in whole or in part, to the collection, use or disclosure of the information; or (c) the information is no longer necessary for the continued provision of a product or service requested by the individual.*"
- India's Digital Personal Data Protection Act (DPDP Act), passed in August 2023, asserts in Article 12 *a*) that: "*A Data Principal shall have the right to correction, completion, updating and erasure of her personal data for the processing of which she has previously given consent.*"

Under the constraints of such modern data privacy regulations, the need to build models incrementally – which is the aims of continual learning – becomes crucial. This is because (i) retraining from scratch becomes impractical since historical data may be either non-existent due to deletion or inaccessible due to privacy and regulatory constraints and (ii) reproducing or regenerating data is prohibited under privacy regulations.

A common example of such privacy constraints in artificial intelligence (AI) arises in healthcare facilities. Consider a scenario where a hospital's AI department has initially trained an ANN model on one set of patient data to forecast the likelihood of diabetes (Task 1). Later, the same model is to be reused or extended for another group of patient to predict a different medical condition, such as hypertension (Task 2), while maintaining its efficacy in diagnosing diabetes (Task 1). Reusing the same model for multiple tasks eliminates the cumbersome process of developing and maintaining separate models for each diagnosis, thereby optimizing resource allocation and operational efficiency.

However, if the patients involved in Task 1 exercise their right to be forgotten, resulting in the complete deletion of Task 1 data, the model must then learn Task 2 in the complete absence of Task 1 data, while still providing accurate predictions for both tasks. This is particularly challenging because the current data alone (task 2 or hypertension patient data) may not be sufficient to maintain performance on the previous task (task 1 or diabetes patient data). Overcoming this challenge requires a well-designed continual learning strategy.

Autonomous Systems In autonomous units like robots or cars, access to computer resources is highly limited. Even if an autonomous system has access to all the data at once, it may lack the computing power or memory capacity to train on it. Consequently, the ability to train incrementally is a highly desired behavior [25, 26].

As an illustration, consider a neural network embedded in an autonomous robot tasked with mapping and navigating a complex environment. The neural network has access to a large amount of data provided by the robot's sensors. However, the sheer volume of this dataset far exceeds the robot's onboard memory capacity such that simultaneously training on all data is infeasable.

To tackle this challenge, the robot could adopt a continual learning strategy. It begins by loading a manageable portion of the sensor data into its limited memory to train an initial version of its mapping and navigation algorithms. As the robot moves through the environment, collecting new sensor data, it overwrites its memory with this fresh information and updates its AI models accordingly. These refinements not only improve the existing map but also enhance the robot's obstacle-avoidance capabilities, leading to increased overall navigational efficiency and accuracy.

Let us summarize the three scenarios presented above. In the first scenario, the data are not simultaneously available because they arrive sequentially over time. While the model could be retrained from scratch as each task arrives, this approach quickly becomes cumbersome and resource intensive. In the second scenario, data from previous tasks is not existent (has been completely deleted) due to privacy and regulatory concerns, making retraining from scratch infeasible. In the third scenario, all data are available simultaneously, but cannot fit into the memory of the autonomous system, making training on all data at once infeasible. These three scenarios illustrate the need to design and develop robust continual learning systems, which is the goal of this thesis, that can learn incrementally instead of stationarily, as is usually the case in traditional machine learning. The next section presents the objectives and contributions of this thesis.

1.2 Objectives and Contributions of the Work

The driving goal of this thesis is to ***present novel continual learning paradigms that allow a neural network to learn multiple sequentially arriving tasks from a changing distribution while respecting memory, privacy, and computational cost constraints.***

The major contributions of the thesis are as follows:

1. As an initial contribution, this thesis presents an enhanced version of the Elastic Weight Consolidation (EWC) algorithm to tackle the issue of catastrophic forgetting in neural networks. This is achieved by identifying the network parameters that were crucial for learning previous tasks and restricting their change during the training of a new task using a regularization term. Traditional EWC relies on approximations of the Fisher information matrix (FIM), which requires to manage and store large matrices. To address this limitation, the thesis introduces a novel metric

for evaluating the importance of each parameter in task-specific learning. This metric is based on the average standard deviation of the post-activation values across all training samples of the CNN's filters. Evidence presented in this thesis confirms this metric as a reliable measure of a parameter's task-specific importance.

2. As a secondary contribution, this thesis introduces a comprehensive continual learning framework specifically tailored for fixed-capacity neural networks. Given that the network capacity is fixed/limited, care must be taken to use it sparingly in order to learn as many tasks as possible in a continual learning scenario. Therefore, a sparsification algorithm is proposed to make the most efficient use of the available capacity when training on each new task.

   Instead of the common practice of using a dense architecture, the proposed sparse network architecture enhances the network's plasticity with respect to future tasks by zeroing out as many weights as possible during training for each task. Furthermore, unlike dense architectures, a sparse architecture offers the flexibility to rewire connections, enabling previously deactivated filters to be reactivated for acquiring new knowledge.

   The basic idea of this sparsification strategy is to exploit the inherent overparameterization of neural networks. Taking into account that neural networks are often overparameterized, so that there is more capacity/parameters than strictly necessary to learn a given task, the sparsity mechanism allows to reinforce weight sparsity during the training process. This is achieved by a specialized regularization technique. Specifically, the objective function for training each task is automatically adjusted to promote weight sparsity, resulting in an effective continual learning framework capable of accommodating a reasonable sequence of tasks without compromising performance.

3. As a third contribution,the framework developed in the second contribution is extended to be able to learn an infinite number of tasks.

   In fact, as mentioned above, the second contribution focuses on neural networks with fixed capacity. Consequently, despite the developed sparsification strategy, the network capacity will reach saturation after learning a certain number of tasks t, where t depends on the initial capacity of the network.

   The third contribution addresses this shortcoming by removing the fixed-capacity network constraint. Specifically, a heuristic mechanism is designed to detect when the network capacity is exhausted and thus decide when to expand the network. However, expanding a network in a continual learning setting poses the problem of how to deal with added parameters, as they may affect the well-functioning of the overall network. To overcome this, sophisticated mechanisms are developed to deal with additional neurons/filters in an optimal and sustainable manner.

   Finally, the contribution includes the development of heuristic mechanisms to identify layers that have reached their capacity and to augment them with additional filters or neurons as needed, ensuring that the network expansion is both targeted and efficient.

1.3 Thesis Structure

The thesis is organized as follows. Chapter 2 serves as a foundational starting point, laying out the background and theoretical foundations of continual learning and catastrophic forgetting that are essential to understanding the rest of the thesis. This foundational chapter further delves into the state of the art and historical insights on continual learning, thereby contextualizing the contributions of the thesis within a broader academic context.

The core of the thesis begins with chapter 3, which delves into the first main contribution, presenting the methods for forgetting prevention based on filters post-activation. Furthermore, the chapter deals with the second main contribution, which focuses on the sparsification strategy. It details the algorithm developed to optimize the use of a fixed-capacity neural network through selective weight zeroing and connection rewiring.

Chapter 4 presents the third major contribution: a framework for network expansion coupled with underlying heuristics. This chapter details how a network can be dynamically adapted to accommodate an infinite number of tasks, while mitigating the challenges of capacity constraints and catastrophic forgetting.

Finally, the thesis concludes in chapter 5, which summarizes the main findings, discusses the implications of the research results, and outlines directions for future work. It discusses the strengths and limitations of the thesis and suggests ways to broaden the scope and application of the research.

Part of this thesis has been published in international peer-reviewed conferences [1, 3, 2, 4, 5, 6].

Other work outside the scope of this thesis has been published on the topic of explainable AI [6], and machine learning in general [27, 28, 29].

Chapter 2

Theoretical Foundations and Related Work

The previous chapter introduced a major challenge faced by traditional deep neural networks in supervised learning, namely their lack of adaptability to evolving data streams in real-world settings. This chapter takes a closer look at continual learning, a method that has proven to be a robust solution to this challenge, allowing learning over long task sequences such as those often encountered in real-world scenarios.

The chapter begins with the notation and formalism used in the thesis, followed by an overview of the basic principles of supervised learning in neural networks. It then explores the problem of catastrophic forgetting, examining the particular circumstances that trigger it and exploring various strategies to mitigate this problem via the continual learning paradigm. A review of related work and techniques for addressing continual learning is then presented. To set the stage for the rest of the thesis, the chapter concludes by outlining the basic terms, concepts, metrics, and datasets that will be used throughout the thesis.

2.1 Notation and Formalism

This section introduces the formalism and notation used throughout the thesis.

In this thesis, T represents a set of classification tasks in the area of supervised learning. The tasks are denoted as $\{\mathcal{T}^1, \ldots, \mathcal{T}^T\}$. Each specific task $\mathcal{T}^t$, includes a unique set of class labels and comes with an associated training set $D^t_{\text{train}} = \{(x^t_i, y^t_i)\}^{m^t}_{i=1}$, where x^t_i is an individual input sample, y^t_i is its corresponding class label, and m^t is the total number of training samples for that task. Additionally, the validation and test data sets for each task are denoted as D^t_{valid} and D^t_{test}, respectively. In this setup, each of the T tasks represents a distinct classification problem, i.e. the classes unique to a given task $\mathcal{T}^t$ do not overlap with those of any other task.

The thesis considers the realistic scenario where the training dataset D^t_{train} and the validation dataset D^t_{valid} for task t are accessible only during its training phase and become unavailable thereafter. This allows to comply with modern data protection and legislation requirements. Furthermore, the test dataset D^t_{test} is reserved for performance evaluation after the model has been trained on subsequent tasks, denoted by $\mathcal{T}^{t''}$, where $t'' > t$.

In terms of neural network architecture, this thesis primarily focuses on convolutional neural networks (CNNs) due to their proven effectiveness in tasks involving image data, which is the primary type of data used in this research. This focus is motivated not only by the unique ability of CNNs to capture spatial hierarchies and patterns but also by the fact that weights within a filter are inherently group-bound. Current research has not sufficiently explored this aspect. This thesis aims to address this gap by examining the significance of treating weights within the same filter uniformly, rather than as distinct units as done in previous work. This approach contributes to the broader understanding and optimal use of CNNs in continual learning systems, and will be further elaborated in Chapter 3.

Within the CNN, layers are indexed from 1 to L and each contains F_l filters f. $\Theta_{l,f}$ denotes the parameters of filters f in layer l, while $a_{l,f}(x_i)$ represents the activation output of a filter f in layer l for a given input sample x_i. The output of the filter post-activation is a two-dimensional feature map with the dimensions $H_o \times W_o$. For any given task t, the set of network parameters is denoted as $\Theta^t = \{\Theta^t_l\}^L_{l=1}$, with $\Theta^t_l = \{\Theta^t_{l,f}\}^{F_l}_{f=1}$.

2.2 Fundamentals of Supervised Learning in Neural Networks

Supervised learning is a subset of machine learning that focuses on training models with labeled data. Labeled data means that each example (x_i) in the training set is paired with the correct output (y_i). During the training phase, the model learns the relationship between the features of the input data and their corresponding labels. This learned relationship enables the model to make accurate predictions for new, unseen data during inference. Neural networks, known for their ability to model complex and non-linear relationships in data, have become increasingly popular in supervised learning. Their impact is particularly notable in image processing tasks, where the relationship between inputs and outputs is often complex. This section explores the core principles, algorithms, and mathematical foundations of supervised learning, specifically in the context of classification using neural networks, which is the primary focus of this thesis.

2.2.1 Basic Concepts

The foundation of supervised learning is a labeled data set of input-output pairs, $\mathcal{D} = \{x_i, y_i\}^m_{i=1}$, where each input x_i is associated with a label y_i. The supervised model $\mathbf{f}_\Theta : x \to y$ learns to predict the labels based on the input features, where both input and label are sampled identically and independently from a fixed joint probability distribution $P(X, Y)$. For example, in handwritten digit recognition, x_i is the input vector, while y_i is the drawn digit.

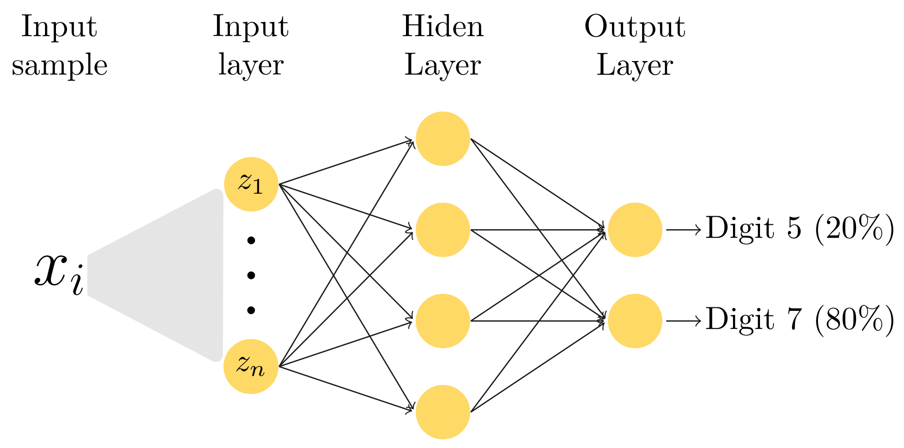


Figure 2.1: A basic multilayer perceptron consisting of three layers: The input layer: This layer receives the input signals (data) and passes them on to the next layer. Each neuron in the input layer represents one of the n features $(z_1, \ldots, z_n)$ of the input sample x_i. The hidden layer: This is where most of the computation within the neural network takes place. Each neuron in this layer processes the input it receives from the previous layer and passes its output to the next layer. Output Layer: Produces the output for the network. The design of this layer depends on the task at hand (e.g., one neuron for binary classification, multiple neurons for multi-class classification). Here, it has two neurons: one for classifying the digit 5 and the other for classifying the digit 7. The network outputs the probabilities for each class, with 20% for the digit 5 and 80% for the digit 7.

2.2.1.1 Multilayer Perceptrons

When dealing with structured data, where the input features are independent and lack spatial or temporal relationships, or in some cases of simpler unstructured data, such as images in handwritten digit recognition, multilayer perceptrons (MLPs) have proven to be a good choice for approximating the function $\mathbf{f}_\Theta(\cdot)$. As the name implies, a multi-layer perceptron is a type of feedforward Artificial Neural Network (ANN) that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer.

Figure 2.1 shows a multi-layer perceptron with three layers. For a given input image x_i, such as a the digit seven, the MLP is mathematically formulated as a function mapping this input to its corresponding output y_i, expressed as:

$$\mathbf{f}(x_i) = \mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(x_i))) \ , \tag{2.1}$$

where, $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{f}_3$ correspond to the input, hidden, and output layers of the network, respectively. Each layer $\mathbf{f}_l$ in the network is defined by a unique weight matrix Θ_l (weights connecting layer $l-1$ to layer l) and a bias vector b_l. These parameters are used to compute the output of the layer according to:

$$\mathbf{f}_l = \sigma\left(\Theta_l \ x_{l-1} + \mathbf{b}_l\right), \tag{2.2}$$

where x_{l-1} is the vector of outputs from the previous layer (or the input vector for the first hidden layer), and σ is the activation function (e.g., ReLU, sigmoid, etc.).

The use of the $\sigma(\cdot)$ activation function addresses the concern that a series of linear transformations, when applied sequentially, is essentially a single linear transformation. Therefore, unless nonlinear elements are integrated between layers, as $\sigma(\cdot)$ does, a multi-layer network would effectively behave like a single-layer network. This limitation would prevent the network from solving highly complex problems.

Other advanced types of neural networks have been developed to approximate the function $\mathbf{f}(\cdot)$ in a supervised learning scenario, each tailored to the specific nature and complexity of the task at hand. Among these, Convolutional Neural Networks (CNNs) stand out, which are mainly used in image processing thanks to their specialized structure suitable for handling visual data. This thesis mainly uses CNNs.

2.2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of neural network that are particularly effective at processing data with a grid-like topology, such as images. They have revolutionized the field of computer vision [30, 31] with their ability to automatically and adaptively learn spatial hierarchies of features from visual inputs.

The core building blocks of CNNs are their convolutional layers. These layers use filters, also called kernels, to perform convolutional operations on the input data, as shown in the upper part of Figure 2.2. Each filter slides over the input data, computing dot products between the entries of the filter and the input at each position. As these filters move over the input, they create feature maps that represent the presence of specific features or patterns in the inputs. In addition, once a CNN learns a pattern in one area of an image (e.g., “eye” in the top left corner), it gains the ability to recognize that pattern in different areas (e.g., top right corner, center), even across different images. In contrast, a fully connected neural network, after learning to recognize a pattern in a specific location, is limited to recognizing that pattern only in that specific location.

Unlike the layers of a MLP, where each neuron is connected to every neuron in the previous layer, the filters in the first convolutional layer of a CNN connect only to pixels within their specific receptive fields in the input image. Then, each filter in the second convolutional layer connects only to a small, rectangular region of the output from the first layer, focusing on localized features. This architectural design allows the network to focus on small, low-level features in the early hidden layers and gradually assemble these into larger, more complex, high-level features in subsequent hidden layers. This hierarchical approach to feature detection and integration reflects the typical structure observed in real-world images, which is a key factor in the effectiveness of CNNs for image recognition tasks.

Convolutional layers in CNNs are typically followed by pooling layers, such as max-pooling layers, to reduce spatial dimensions and computational complexity. Specifically, they progressively reduce the spatial size of the representation, thereby reducing the number of parameters and computations in the network. Pooling also plays a crucial role in making the feature detection process invariant to changes in scale and orientation.

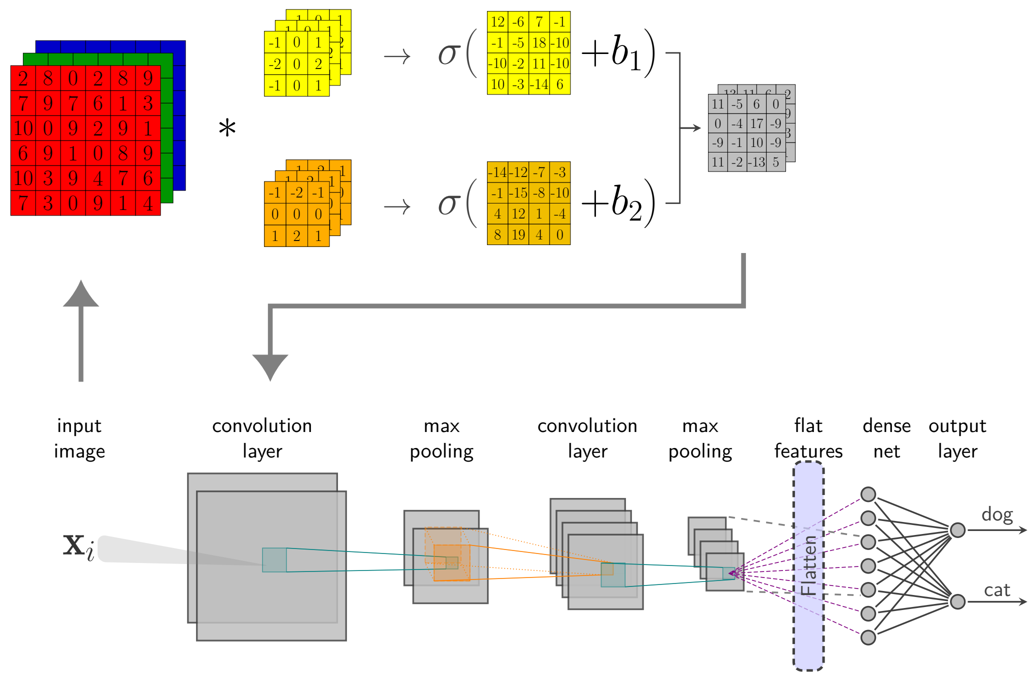


Figure 2.2: Illustration of a typical CNN architecture, including convolutional layers, max-pooling layers, and fully connected layers. The output layer predicts the class of the input sample x_i. The first convolutional layer (upper part of the image) consists of several filters applied to an input image to extract different features. Each filter in the convolutional layer detects different patterns or features in the image, such as edges or specific shapes. These extracted features are progressively refined by subsequent layers to facilitate accurate classification. In this example, the first convolutional layer applies two filters of size $3 \times 3 \times 3$, shown in yellow and orange, respectively, to the input image.

After each convolution operation in a CNN, a nonlinear layer (σ), similar to those found in multilayer perceptrons, is applied to introduce nonlinear properties into the network. This is illustrated in the upper part of Figure 2.2.

Finally, a CNN includes fully connected layers, similar to those found in a multilayer perceptron, typically located at the end of the network. Before reaching these layers, the feature maps generated by the convolutional and pooling layers are flattened into a vector. These fully connected layers integrate the learned features from the convolutional and pooling layers to classify the input image into different categories. Figure 2.2 illustrates a CNN with all of the above mentioned steps.

Equation (2.3) shows how the convolution operation applies a filter to an input image or feature map to produce a pixel of an output feature map. The summation is performed over the spatial dimensions of the filter and over the depth of the input feature maps. Specifically, a pixel in row i, column j of feature map k in a given convolution layer l is computed as the outputs of the pixels in the previous layer $l-1$, located in rows $i \times s_h$ to $i \times s_h + f_h - 1$ and columns $j \times s_w$ to $j \times s_w + f_w - 1$, over all feature maps in layer $l-1$:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \times \Theta_{u,v,k',k}$$
$$\text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \tag{2.3}$$

In this equation, $z_{i,j,k}$ is the value at position (i, j) in the k-th output feature map, b_k is the bias term for the k-th output feature map, f_h, f_w are the height and width of the filter, respectively, while f'_n is the number of feature maps in the previous layer. Furthermore, $\Theta_{u,v,k,k'}$ is the connection weight between each pixel in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'. Finally, s_h, s_w are the horizontal and vertical stride of the filter across the feature maps in layer $l - 1$.

2.2.2 Training Artificial Neural Networks

Artificial neural networks are trained using two main techniques: reverse-mode autodifferentiation (abbreviated as reverse-mode autodiff) and gradient descent (GD). Reverse-mode autodiff efficiently computes the gradients of the network's error with respect to each model parameter in only two passes through the network: one forward and one backward. Essentially, it determines how each connection weight and bias should be adjusted to minimize the network error, which is typically defined as the difference between the predicted output $\hat{y}$ and the expected output y. The loss function is a differentiable function, such as the squared Euclidean distance for regression:

$$L(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2.4}$$

or the cross-entropy loss (also known as log loss) for (multi-class) classification:

$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^{m} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}) \tag{2.5}$$

where, $y_{i,c}$ is a binary indicator of whether class c is the correct classification for sample i and $\hat{y}_{i,c}$ is the predicted probability that sample i is of class c.

By repeatedly computing the gradients automatically and applying gradient descent steps for each parameter Θ_j as shown in Equation (2.6) and (2.7) respectively, the neural network progressively reduces its error until it converges to a minimum:

$$\nabla_{\Theta_j} L(\hat{y}, y) = \frac{\partial L\left(\mathbf{f}\left(x; \Theta_j\right), y\right)}{\partial \Theta_j} \tag{2.6}$$

$$\Theta_j \leftarrow \Theta_j - \eta \nabla_{\Theta_j} \tag{2.7}$$

This combination of reverse-mode autodiff and gradient descent is called backpropagation. The goal of training is to find the model parameters that minimize the loss over the entire training set.

A key strength of artificial neural networks is their ability to efficiently back-propagate the gradient through the network. This propagation uses the chain rule to transfer the gradient from one layer to the next. For the network shown in Figure 2.1, and its corresponding mathematical representation $\mathbf{f}(x_i) = \mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(x_i)))$ as given in Equation (2.1), the chain rule for the gradient of the loss with respect to the parameters of each layer is:

$$\frac{\partial L}{\partial \Theta_1} = \frac{\partial L}{\partial \mathbf{f}_3} \cdot \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2} \cdot \frac{\partial \mathbf{f}_2}{\partial \mathbf{f}_1} \cdot \frac{\partial \mathbf{f}_1}{\partial \Theta_1} \tag{2.8}$$

$$\frac{\partial L}{\partial \Theta_2} = \frac{\partial L}{\partial \mathbf{f}_3} \cdot \frac{\partial \mathbf{f}_3}{\partial \mathbf{f}_2} \cdot \frac{\partial \mathbf{f}_2}{\partial \Theta_2} \tag{2.9}$$

$$\frac{\partial L}{\partial \Theta_3} = \frac{\partial L}{\partial \mathbf{f}_3} \cdot \frac{\partial \mathbf{f}_3}{\partial \Theta_3} \tag{2.10}$$

where $\Theta_1, \Theta_2, \Theta_3$ denote the parameters of the first, second, and third layers, respectively. Each of these gradients is used during backpropagation to update the corresponding parameters in the network by moving them in the direction that reduces the loss.

Backpropagation is widely used in machine learning and remains the most common technique for training neural networks. The method for the update step during training can vary depending on the amount of data used. When the entire dataset is used for each update, the process is called batch gradient descent. Conversely, mini-batch gradient descent uses only subsets of the data for each update. Finally, stochastic gradient descent updates the model using only one data point at a time.

2.2.3 Typical Optimization Algorithms

The gradient descent update rule outlined in Equation (2.7) illustrates the basic form of optimization. In recent years, the development of more advanced and efficient optimization techniques has played a crucial role in significantly speeding up modern supervised learning algorithms. Some of the most notable of these methods are Momentum, Nesterov Accelerated Gradient, AdaGrad, RMSProp, and Adam. The following section will discuss the specific technique used in this thesis.

2.2.3.1 Adaptive Gradient (AdaGrad)

AdaGrad modifies the standard gradient descent algorithm by adaptively adjusting the learning rate (η) for each parameter. This is particularly useful when dealing with unbalanced data or large-scale learning problems, as will be the case in chapters 3 and 4.

The AdaGrad update rule can be divided into two main steps:

$$s \leftarrow s + \nabla_\Theta L(\Theta) \otimes \nabla_\Theta L\Theta) \tag{2.11}$$

$$\Theta \leftarrow \Theta - \eta \frac{\nabla_\Theta J(\Theta)}{\sqrt{s + \epsilon}} \tag{2.12}$$

The first step (Equation (2.11)) involves updating a runing total of the squares of the gradients. In this equation, s, is a vector of the sum of squared gradients and $\otimes$ denotes the element-wise multiplication. The second step (Equation (2.12)) is almost identical to gradient descent, but with one major difference: The gradient vector $\nabla_\Theta J(\Theta)$ is scaled down by a factor of $\sqrt{s} + \epsilon$ (with ϵ being a small constant added to improve numerical stability).

Although AdaGrad is not directly utilized in this thesis, it provides a foundational understanding for RMSprop and Adam, both of which are extensively used.

2.2.3.2 Root Mean Square Propagation (RMSprop)

A common issue with AdaGrad is that the learning rate (η) can become infinitesimally small. It may become so negligible that the algorithm stops altogether before reaching the global optimum. RMSprop addresses this problem by accumulating only the gradients from recent iterations, rather than from the entire training period. This approach is often described as using a moving average of the squared gradients. RMSprop achieves this by implementing an exponential decay in the initial step, which ensures a more balanced and effective adjustment of the learning rate over time.

$$v \leftarrow \rho v + (1 - \rho) \nabla_\Theta L(\Theta) \otimes \nabla_\Theta J(\Theta) \tag{2.13}$$

$$\Theta \leftarrow \Theta - \eta \frac{\nabla_\Theta L(\Theta)}{\sqrt{s + \epsilon}} \ , \tag{2.14}$$

where v is the moving average of the squared gradients and ρ is the decay rate, which determines how quickly the moving average forgets the previous squared gradients.

RMSprop is particularly useful for non-convex optimization problems, such as those encountered in Chapters 3 and 4, as it helps to maintain a manageable learning rate by adjusting the step size based on recent gradient magnitudes.

2.2.3.3 Momentum

Momentum optimization addresses a key limitation of regular gradient descent, which moves in small steps on gentle slopes and larger steps on steep slopes without considering the history of the gradients. When the local gradient is small, regular gradient descent moves very slowly, potentially causing delays in reaching the minimum.

In contrast, momentum optimization pays considerable attention to previous gradients. At each iteration, it adjusts the momentum vector m by subtracting the local gradient (scaled by the learning rate η). The weights are then updated by adding this momentum vector. This approach essentially builds up velocity, helping the algorithm to navigate through flat areas more swiftly and dampening oscillations, leading to faster and often more stable convergence to the minimum. he momentum update rule is given as:

$$v \leftarrow \beta\ v - \eta \nabla_\Theta L(\Theta) \tag{2.15}$$

$$\Theta \leftarrow \Theta + v\ , \tag{2.16}$$

where β is the momentum coefficient (often set between 0.9 and 0.99). While momentum is not used directly in this thesis, it provides the foundation for understanding Adam, which is used extensively throughout the thesis.

2.2.3.4 Adaptive Moment Estimation (Adam)

Adam combines ideas from both Momentum and RMSprop. It maintains two moving averages for each parameter: one for the gradients (similar to Momentum) and one for the squared gradients (similar to RMSprop). It then uses these moving averages to adapt the learning rate for each parameter over the course of training. This is given as:

$$1.\quad m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\Theta J(\Theta) \tag{2.17}$$

$$2.\quad s \leftarrow \beta_2 s + (1-\beta_2)\left(\nabla_\Theta J(\Theta)\right)^2 \tag{2.18}$$

$$3.\quad \hat{m} \leftarrow \frac{m}{1-\beta_1^t} \tag{2.19}$$

$$4.\quad \hat{s} \leftarrow \frac{s}{1-\beta_2^t} \tag{2.20}$$

$$5.\quad \Theta \leftarrow \Theta + \eta \frac{\hat{m}}{\sqrt{\hat{s}}+\epsilon}\ , \tag{2.21}$$

where β_1 called momentum decay, corresponds to β in momentum optimization and is typically set to 0.9, while β_2 called scaling decay corresponds to ρ in RMSprop and is typically set to 0.999. Both β_1 and β_2 are set to these values in this thesis.

2.3 Introduction to Catastrophic Forgetting in Supervised Learning

The ability to continuously accumulate knowledge is a hallmark of general intelligence, yet artificial neural networks struggle to remember old concepts after acquiring new ones. As described in Chapter 1, this shortcoming is commonly known as catastrophic forgetting [7, 8]. Considering a standard neural network architecture, which typically consists of multiple layers of linear projections followed by element-wise nonlinearities (as discussed in the previous section), there are several scenarios in which catastrophic forgetting can occur. The rest of this section delves into these scenarios.

2.3.1 Task Incremental Learning

Consider a neural network learner $\mathbf{f}_\Theta$ that is initially trained on a supervised learning classification task – referred to as `Task 1` – which involves differentiating between images of cats and dogs, as shown in the first column of the task incremental row in Figure 2.3.

During the training phase, the neural network iteratively adjusts its parameter set, Θ, which includes weights and biases associated with the linear transformations within the network (as discussed in Section 2.2.3). This process continues until the network reaches an optimal set of learned parameters, Θ_1, that accurately solves `Task 1`.

Subsequently, as shown in the second and third columns of the task incremental row in Figure 2.3, the neural network f_{Θ_1} faces new classification challenges. In the first case, it must distinguish between various types of organisms, such as plants and fungi (`Task 2`). In another instance, it may need to classify different types of vehicles, including bicycles and trucks (`Task 3`). Training the network exclusively on the data from these new tasks would cause it to update its parameters, prioritizing the requirements of `Task 2` or `Task 3`. As a consequence, the parameters fine-tuned for `Task 1` would be overwritten, resulting in the loss of previously acquired knowledge.

More specifically, when the neural network is trained on `Task 2` or `Task 3`, its parameters evolve from the initial set Θ_1 to a new, task-specific set Θ_2 or Θ_3. This new parameter configuration is optimized for the current task, but may not be well suited for `Task 1`. As a result, if the network is later evaluated on `Task 1` after adapting to `Task 2` or `Task 3`, a significant drop in performance on `Task 1` is to be expected. This is the problem of catastrophic forgetting in so-called task incremental learning.

2.3.2 Class Incremental Learning

In class incremental learning scenarios, the focus is not on separate tasks, but rather on expanding categories within a single thematic domain, such as identifying different types of carnivores, as shown in the class incremental row of Figure 2.3. First, the neural network $\mathbf{f}_\Theta$ is fine-tuned to distinguish between cats and dogs, here referred to as `Classes 1`, and achieves proficiency in this area.

Next, new classifications are introduced within the same carnivore class. For example, it may be tasked with distinguishing between bears and wolves, denoted `Classes 2`, or between foxes and mustelines, denoted `Classes 3`. If training focuses exclusively on the datasets for these new classes – `Classes 2 or 3` – , the network will likely experience significant misclassification issues when it comes to distinguishing between `Classes 1`. This is a classic example of the catastrophic forgetting problem in class incremental learning.

2.3.3 Domain Incremental Learning

Another manifestation of catastrophic forgetting can be observed when new data arrive for the same task and classes, but with different input domains. These variations could be due to differences in data collection conditions, sources, or stylistic elements. To illustrate this, consider the domain incremental row of Figure 2.3, where a neural network is first

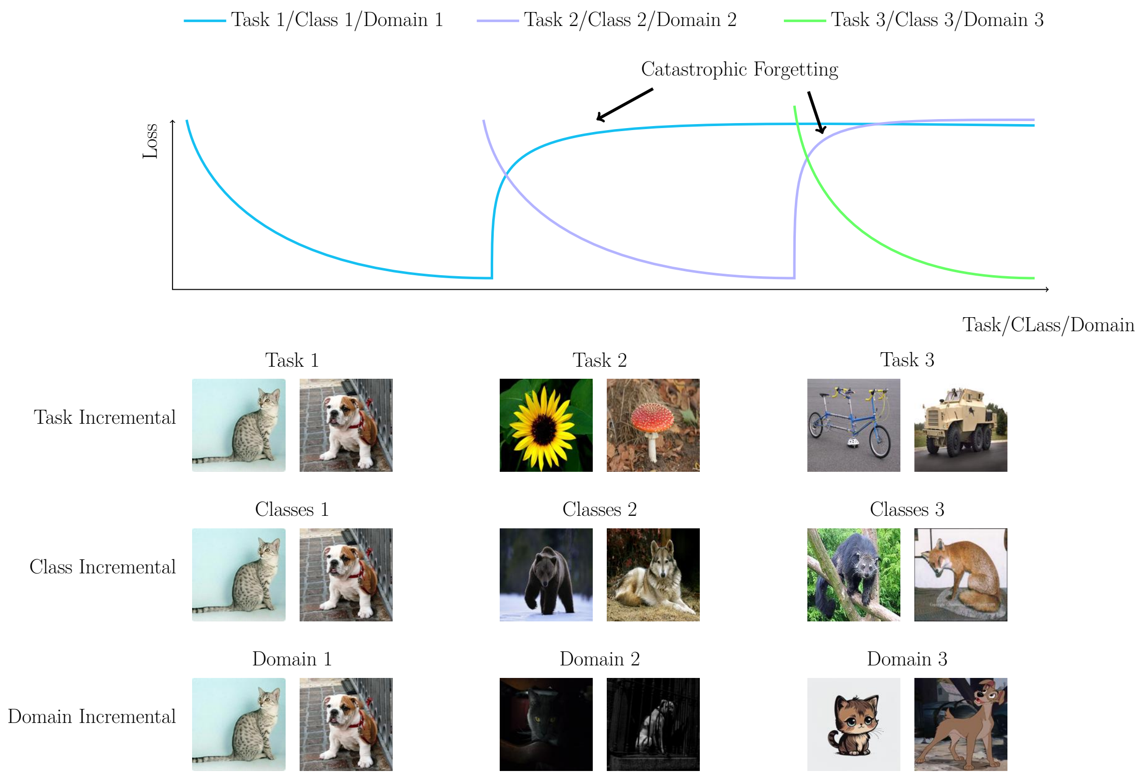


Figure 2.3: Overview of task-, class-, and domain-incremental learning. The first row shows curves of task/class/domain incremental learning versus loss, highlighting the phenomenon of catastrophic forgetting. After each new task/class/domain is learned in a naive manner (i.e., with no immunity to catastrophic forgetting), there is a significant degradation in the performance of the neural network on previous knowledge, as measured by increased loss. The Task Incremental row displays images from different tasks (animals, organisms, vehicles) arriving in sequence. The Class Incremental row shows the Model exposed to different categories within the same group of animals (dogs vs cats, bears vs wolves, and foxes vs mustelines), here carnivores. The Domain Incremental row illustrates images representing the same classes of animals (cats and dogs) but with different input domains, such as night and cartoon versions.

trained to recognize cats and dogs from daylight images, referred to as `Domain 1`. The model is then tasked with identifying these animals in images taken at night, referred to as `Domain 2`, which come with different lighting conditions and possibly altered animal appearances. Finally, the model is challenged to recognize cartoon representations of the same animals, categorized as `Domain 3`.

If the training of the neural network focuses only on the new data introduced with each successive domain, the model will tend to adapt mainly to these more recent domains. This would compromise its ability to accurately recognize images from earlier domains, leading to the problem of catastrophic forgetting.

In general, while artificial neural networks excel in supervised learning within static environments, they often struggle to adapt their behavior when when faced with new tasks, classes, or domains over time. This limitation presents a significant challenge in real-world scenarios where continual learning and adaptability are essential. A natural solution to this well-known catastrophic forgetting problem is to restart the training process with all available data each time new classes, tasks, or domains are introduced. However, this practice quickly becomes unsustainable in today's very dynamic world, as it would require storing and manipulating extremely large amounts of data. On the other hand, data privacy laws and regulations (e.g., GDPR) often make this strategy infeasible, as they require old data to be fully deleted after a certain period of time. As a result, there is a strong need to develop systems that continuously adapt and learn over time, commonly referred to as continual learning systems. The central goal of these CL systems is to address the challenge of catastrophic forgetting while limiting model capacity, computational cost, and storage requirements.

2.4 Introduction to Continual Learning for Supervised Learning

This section presents a formal and conceptual definition of continual learning (CL), exploring its historical context and driving principles. Core criteria and key terminology related to continual learning are then elaborated to provide a solid vocabulary foundation for the remainder of the thesis. The section concludes by outlining the boundaries between continual learning systems and other areas of machine learning.

2.4.1 Definition and Challenge of CL

Continual Learning (CL) is designed to train models sequentially on evolving data streams with varying distributions, balancing the retention of prior knowledge with adaptability to new information. This ability to sequentially learn multiple tasks, accumulate experience, and avoid forgetting is central to human-level intelligence, but poses a significant challenge for artificial neural networks. Often referred to interchangeably as lifelong learning [32, 33, 9], incremental learning [34, 35, 36], or sequential learning [37, 7, 38], continual learning aims at the dynamic accumulation of knowledge from data governed by non-stationary distributions.

Two key challenges emerge in CL when assimilating new data: (i) mitigating catastrophic forgetting, or the loss of previously acquired knowledge (presented in Section 2.3), and (ii) optimizing network plasticity for effective assimilation of future information. Addressing these issues primarily involves solving the stability-plasticity dilemma, a challenge encountered in both biological and artificial neural networks [39]. In the context of sequential data, an artificial learning model requires both stability, to sustain performance on earlier tasks without retraining on old data, and plasticity, to seamlessly integrate new knowledge and adapt to incoming data. If the model is trained without any measure against the forgetting of previous knowledge, therefore focusing on the current task only, it will be very plastic but not stable, meaning that it can learn fast but also forgets quickly. On the other hand, if the network focuses mainly on being immune to forgetting, it may lack plasticity i.e., enough capacity to learn future tasks, especially in the case of a fixed-capacity network, which is the focus of Chapter 3.

To address the stability-plasticity dilemma in continual learning, researchers have primarily focused on three methodological approaches: (i) weight regularization, which discourages drastic changes in network parameters by regularizing the loss function; (ii) model expansion, which involves introducing task-specific representations to mitigate catastrophic forgetting during the continual learning process; and (iii) memory replay, which employs various criteria to select a representative subset of data to be revisited during the training of subsequent knowledge. Weight regularization and model expansion will be discussed in detail in Sections 2.5.2 and 2.6.2. Memory replay is nowadays used less frequently due to its substantial storage requirements and potential privacy violations, and is not within the scope of this thesis.

In this thesis, the term "task" is used broadly to encapsulate various forms of incremental learning, including class, domain, and task incremental learning. Formally, if $\mathbf{f}_\Theta$ denotes the supervised neural network learner, parameterized by Θ, and assume that each task is associated with a dataset $D^t = \{(X_i^t, Y_i^t)\}_{i=1}^{m^t}$, where m^t denotes the number of samples in each task-specific dataset. The primary goal of continual learning, aside from maintaining plasticity, is to minimize the loss for each newly encountered task t, while simultaneously ensuring that the loss related to each previously learned task t' either remains stable or improves:

$$\begin{aligned} &\Theta^t = \arg\min_{\Theta^t} \frac{1}{m^t} \sum_{i=1}^{m^t} \ell\left(\mathbf{f}\left(x_i^t; \Theta^t\right), y_i^t\right) \\ &s.t \;\; \mathcal{L}^{t'}(\Theta^t) \leq \mathcal{L}^{t'}(\Theta^{t'}) + \xi^t, \quad \forall\, t' < t \\ &\xi^t \geq 0 \;, \end{aligned} \tag{2.22}$$

where, $\mathcal{L}^{t'}(\Theta^t)$ represent the loss for a previous task t' evaluated with the parameters Θ^t after learning task t, $\mathcal{L}^{t'}(\Theta^{t'})$ the loss of t' evaluated with the parameters $\Theta^{t'}$ immediately after learning task t', and ξ^t is a variable that allows for a slight increase in the losses incurred in previous tasks.

2.4.2 History and Motivation of CL

The idea of continual learning has been an integral part of artificial intelligence since its inception. However, the systematic study of this concept did not gain significant traction until the end of the 20th century. Cognitive scientist Mark Ring catalyzed its popularity within the machine learning community in 1992 with his seminal paper "Learning Sequential Tasks by Incrementally Adding Higher Orders" [40]. In this paper, he proposed the Temporal Transition Hierarchies (TTH) algorithm, a temporal function approximator designed to incrementally extend an agent's history length as a means of mitigating catastrophic forgetting.

Despite this fundamental contribution, the field of continual learning received only sporadic interest in the machine learning community between the late 1990s and the first decade of the 21st century. It's only in the last few years that significant research efforts have been revived in this area. Five primary factors can be identified to explain this historical lack of sustained attention:

- `Computational limitations`: Prior to the 1990s, limited, expensive, and less accessible computational resources prevented researchers from experimenting with complex learning problems such as continual learning, which requires a single model to perform multiple tasks sequentially.
- `Limited Memory and Storage Capacity`: In contrast to computational limitations, a related challenge was the memory and storage capacity of early computer systems, which was much lower than today's standards. Some categories of continual learning models require substantial memory to store information from previous tasks, which was a hurdle in the past.
- `Scarcity of large datasets`: Before the advent of the Internet and the big data boom of the 21st century, driven by the emergence of rich datasets such as ImageNet (2009) or COCO (2014), the availability of large and diverse datasets was limited. The effectiveness of continual learning algorithms depends on rich datasets to facilitate adaptation and generalization across multiple tasks; the lack of such data posed significant challenges to the practical implementation and testing of continual learning methods.
- `Algorithmic Constraints`: In earlier decades, the field of machine learning relied primarily on simpler, more basic algorithms, such as linear regression, decision trees, and basic forms of clustering, which were limited in their ability to handle complex patterns in data. Techniques that are fundamental to modern continual learning, such as various forms of regularization (crucial in regularization-based continual learning approach to prevent the model from overfitting the new task), knowledge transfer (essential in continual learning as it allows the use of knowledge from previous tasks to improve performance or accelerate learning on new tasks), etc., were either not developed or not optimized.
- `Focus on Theoretical Research Over Practical Application`: Historically, the field of deep learning has focused primarily on theoretical research, with less emphasis on practical application. This theoretical focus obscured the real-world challenges

and requirements that underscore the need for continual learning. As deep learning began to find applications in real-world scenarios, new requirements emerged that underscored the importance of models capable of continuously learning from evolving data streams. As a result, continual learning gained prominence as these real-world applications highlighted its importance and drove its development, especially in recent times.

2.4.3 CL Desiderata

Over the past few years, several guiding principles have emerged that shape ongoing research in the area of continual learning. These principles describe the essential properties that are expected from continual learning algorithms. Some frequently emphasized desiderata (see [9, 41]), which are consistent with the present work, are as follows:

1. `Data Exclusivity/Privacy:` After training on a particular task, the model should not revisit the training data for that task. Such data is considered inaccessible, often for privacy reasons. This aligns with modern regulations, such as GDPR, which emphasize the protection of personal data and limit the retention or reuse of sensitive information.

2. `Learn without forgetting:` When deployed or evaluated, the continual learning model should maintain optimal performance across all tasks, both new and old. This means the model must have the capability to incrementally acquire new skills without erasing or degrading its prior knowledge.

3. `Forward transfer of knowledge:` Knowledge gained from previous tasks should enhance the efficiency and speed of learning new tasks. This transfer not only accelerates the learning process but also helps the model generalize better to unfamiliar challenges, leveraging prior experience to improve overall performance.

4. `Problem Agnostic:` A continual learning algorithm should be versatile and not limited to a specific problem domain such as classification. It should be adaptable to various tasks, such as regression, reinforcement learning, or other types of problem-solving scenarios, ensuring broad applicability.

5. `Time-efficient learning:` The model should be able to adapt and learn under tight time constraints, especially if it is intended for real-time or online applications.

2.4.4 Delimitation to Other ML Fields

This section explores key machine learning paradigms that are closely related to continual learning, such as multi-task learning, transfer learning, and federated learning, in order to clarify their conceptual boundaries and contextualize continual learning within the broader machine learning landscape. This discussion not only sheds light on the unique characteristics and benefits of continual learning, but also highlights its differences from these related fields.

2.4.4.1 Multi Task Learning

Multi-task learning (MTL), also known as joint learning, seeks to enhance model performance by concurrently training on multiple related tasks. In contrast to continual learning, MTL allows simultaneous exposure to all tasks during training, which violates the first CL desiderata (training with the current task data only) mentioned in Section 2.4.3. This allows the MTL model to recognize both similarities and differences between tasks, thereby optimizing its overall effectiveness. However, a limitation of MTL is its general inability to adapt to new tasks sequentially without retraining or significant modification. As a result, multi-task learning lacks the inherent ability to solve the challenge of sequential task adaptation, a core feature of continual learning without retraining.

2.4.4.2 Transfer Learning

Transfer learning is a branch of ML that deals with transferring knowledge from one task to another, and is a widely used technique in machine learning today, especially when dealing with DNNs. A key property that enables transfer learning in DNNs is the hierarchical organization of features across layers. For example, lower hidden layers typically encode basic structures such as line segments, while intermediate layers aggregate these into more complex shapes such as squares or circles. The higher hidden layers, along with the output layer, synthesize these intermediate features to represent even more sophisticated, high-level constructs such as faces. This hierarchical organization of features is essential for efficient knowledge transfer between tasks.

As an example, consider a scenario where a deep learning model has been trained to identify cats in images (Task 1) and is later tasked with identifying dogs (Task 2). In transfer learning, training for Task 2 can be accelerated by reusing the lower layers of the model trained on Task 1. This approach avoids the need to relearn the basic structures common to both cats and dogs, and instead focuses on refining the higher-level features that distinguish the two species.

Both transfer learning and continual learning use a pre-trained model when tackling a new task. However, they differ in their post-training requirements. In transfer learning, the top layers of the network can be completely reconfigured to specialize on Task 2, since maintaining proficiency on Task 1 is not a requirement. This core feature of transfer learning, however, violates the second desideratum of continual learning (learning without forgetting) mentioned in Section 2.4.3. In contrast, continual learning requires the model to maintain its performance on Task 1 even after training on Task 2, which is a more complicated challenge.

2.4.4.3 Federated Learning

Federated Learning is a distributed approach to machine learning where training is performed across multiple decentralized devices or servers, each holding local data samples. This setup is similar to continual learning in terms of adapting to learning from decentralized data sources, although the latter focuses more on sequential task acquisition.

Both techniques require models to be highly adaptive and able to generalize from different data distributions. In federated learning, this adaptability is crucial to effectively learn from diverse, localized data sets, similar to how continual learning must adapt to new tasks while retaining prior knowledge. In addition, both face challenges in handling non-IID data, as federated learning deals with potentially disparate data across devices, similar to the diverse data distributions that continual learning encounters with each new task.

Despite their differences, with federated learning emphasizing privacy and distributed computing, and continual learning focusing on sequential learning without forgetting, they share a core similarity in their quest for robust, adaptive models capable of performing efficiently in dynamic, real-world scenarios with decentralized or varying data sources.

2.4.4.4 Online Learning

Online learning is primarily concerned with the real-time adaptation of a model's parameters in response to incoming data, with the goal of accommodating shifts in the data distribution as they occur. In this framework, the model undergoes incremental training, where each new data point or batch is used to immediately adjust the model's parameters. This is in contrast to continual learning, where the focus is not only on adapting to new data, but also on retaining skills from previous tasks. In essence, online learning is designed to respond to evolving data in real time and does not align with the first and second continual learning desiderata mentioned in section 2.4.3. In contrast, continual learning aims to balance the acquisition of new knowledge with the retention of previously learned knowledge. Therefore, continual learning can be seen as a combination of online learning and avoiding catastrophic forgetting without retraining on previous tasks.

In summary, the present section showed that continual learning, as a distinct paradigm within machine learning, addresses unique challenges and scenarios that set it apart from concepts such as multi-task learning, transfer learning, federated learning, and online learning. While each of these paradigms shares a common approach to learning from data, continual learning is distinguished by its focus on sequential task learning and the retention of previously acquired knowledge. This comparison of related but distinct learning strategies not only clarifies the conceptual boundaries of continual learning, but also underscores its importance in scenarios where adaptability over time and preservation of past learning are crucial.

As outlined in Section 2.4.1, there are three primary methods that address continual learning: regularization-based, expansion-based, and replay-based approaches. The following sections will provide a theoretical foundation and a comprehensive overview of the first two groups, as they are the focus of this thesis.

2.5 Foundational Principles and Overview of Regularization Based CL Approaches

This section provides a detailed exploration of the basic principles and methods associated with regularization-based approaches in continual learning. It begins with a theoretical overview focusing on the fundamental concepts and frameworks that form the basis of regularization strategies in continual learning systems. This is followed by a detailed examination of various methods and techniques that have been developed and implemented in the field.

2.5.1 Theoretical Foundations

Let us first introduce some formalism of regularization methods in continual learning. It is assumed a stream of data composed of $T \geq 2$ distinct tasks, denoted as $\{\mathcal{T}^1, \cdots, \mathcal{T}^T\}$, each arriving sequentially. For each task t, the model has access solely to data from $\mathcal{T}^t$ and the previously learned parameter set $\Theta^{1:t-1}$, which is the cumulative result of continual learning from task $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$. Additionally, let $\mathbf{f}_\Theta$ represent a function parameterized by Θ that implements the neural network model. At each task t, the model learns an optimal set of parameters $\Theta^{*,t}$ that optimizes the task loss $\mathcal{L}^t_{\text{task}}$ (typically a cross-entropy loss for classification). In a continual learning context, it is imperative that $\Theta^{*,t}$ not only optimizes the current task $\mathcal{T}^t$ but is also optimal for all preceding tasks $\mathcal{T}^{1:t-1}$.

In machine learning in general, regularization is a technique used to prevent overfitting by adding a penalty term to the objective function. In the specific context of continual learning, regularization is used to prevent models from overfitting to new tasks, which can lead to the loss of previously acquired skills. The theory behind these methods is rooted in the concept of stability-plasticity dilemma, which as discussed in Chapter 1, refers to the trade-off between a model's plasticity (its ability to learn new information) and its stability (its ability to retain previously learned information).

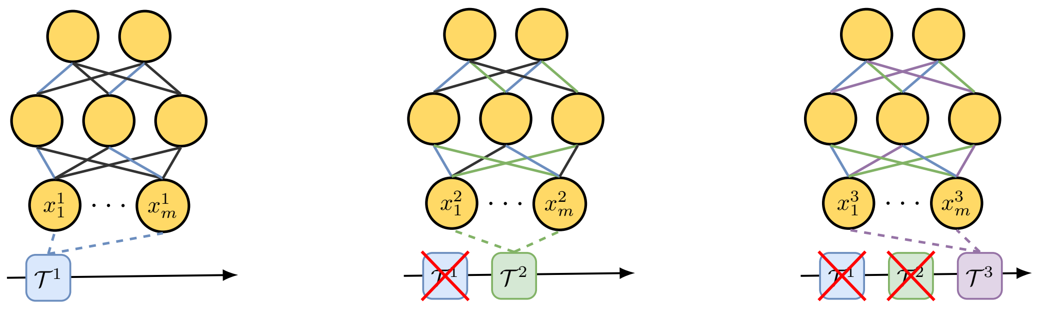
The objective is to find a balance where the model remains flexible enough to absorb new knowledge, while maintaining enough stability to prevent the loosing of previously acquired information. Mathematically, this balance is achieved by modifying the loss function such that its includes two terms: one for the current learning task $\mathcal{L}_{\text{Task}}$ (typically a cross-entropy loss) and another that acts as a regularizer, denoted $\mathcal{L}_{\text{reg}}$ and formulated as:

$$\mathcal{L}_{\text{Reg}}(\Theta^{*,t-1}, \Theta^t) = \frac{1}{2}\sum_i \Omega_i^{1:t-1}\left(\Theta_i^t - \Theta_i^{*,t-1}\right)^2, \tag{2.23}$$

where $\Omega_i^{1:t-1}$ is a vector containing the importance of each parameter for learning $\mathcal{T}^1$ up to $\mathcal{T}^{t-1}$ and the factor $1/2$ simplifies the derivative calculations during optimization.

The regularization loss $\mathcal{L}_{\text{Reg}}(\Theta^{*,t-1}, \Theta^t)$ is usually scaled by a hyperparameter $\lambda \in [0, 1]$, which controls the strength of the regularization. This regularizer imposes a constraint on the optimization process, penalizing changes in parameters that are deemed important for past learning tasks (with the notion of importance determined accordingly). Thus, it effectively embeds a memory of past tasks into the model, helping to prevent catastrophic forgetting.

Figure 2.4 shows the important weights for each task, represented by solid lines in the same color as their corresponding tasks. The regularization term binds the dynamics of each network weight Θ_i^t at task t to the corresponding old network weight $\Theta_i^{1:t-1}$ through the importance parameter $\Omega_i^{1:t-1} \geq 0$.


x_1^1 ⋯ x_m^1
$\mathcal{T}^1$
x_1^2 ⋯ x_m^2
$\mathcal{T}^1$ $\mathcal{T}^2$
x_1^3 ⋯ x_m^3
$\mathcal{T}^1$ $\mathcal{T}^2$ $\mathcal{T}^3$

(a) Beginning with Task 1 (b) Arrival of Task 2 (c) Arrival of Task 3

Figure 2.4: Illustration of regularization-based continual learning strategies. The solid blue connections indicate weights that were essential for learning the first task. In contrast, the solid green and purple connections indicate weights that are essential for the second and third tasks, respectively. While some strategies involve freezing the connections tied to a particular task, more recent approaches tend to constrain these connections to ensure minimal change during the learning of subsequent tasks. The red "X" marks over Task 1 and Task 2 in (b) and (c) indicate the absence of these tasks' data upon the arrival of Task 2 and Task 3, respectively.

In other words, when a new task is introduced, the model's parameters are adjusted not only to minimize the error on this new task but also to ensure that the adjustments do not significantly disrupt the knowledge retained from previous tasks. To achieved this, regularization approach leverage overparameterization of neural network, meaning that neural network often contain more parameter than needed to learn a given task as demonstrated by Meyes et al. in [42]. Therefore, after or during learning a given task, it can be computed which parameters or weights was important in learning that task. This is represented by the importance vector $\Omega_i^{1:t-1}$, which indicate how crucial each parameter is to the model performance on past tasks.

The learning process at each task $\mathcal{T}^t$ then becomes a constrained optimization problem, where the model searches for parameter values that strike an optimal balance between the new and old tasks:

$$\Theta^{*,1:t} = \arg\min_{\Theta^*} \left[\mathcal{L}_{\text{CL}}^t(\Theta^t) = \mathcal{L}_{\text{Task}}^t(\Theta^t) + \frac{\lambda}{2} \mathcal{L}_{\text{reg}}^t \left(\Theta^{*,t-1}, \Theta^t \right) \right] \tag{2.24}$$

where $\mathcal{L}_{\text{CL}}^t$ is the continual learning loss, $\mathcal{L}_{\text{Task}}^t$ is the task-specific loss, $\mathcal{L}_{\text{reg}}^t$ is the regularized loss as defined in Equation (2.23), and $\lambda \in [0, 1]$ is a hyperparameter that scales the regularization loss. This scaling controls the trade-off between effectively fitting the current task (reducing the task-specific loss $\mathcal{L}_{\text{Task}}^t$) and preserving prior knowledge (reducing the regularization loss $\mathcal{L}_{\text{reg}}^t$).

As mentioned previously, the term $\Omega_i^{t-1} \in [0, 1]$ quantifies the importance of the weight $\Theta_i^{1:t-1}$ in learning previous tasks $\mathcal{T}^{1:t-1}$. A lower value of Ω_i^{t-1} indicates that the weight was relatively unimportant for previous tasks, while a higher value indicates its critical importance for past learning, implying that $\Theta_i^{1:t-1}$ should undergo minimal modification. Another key aspect of applying regularization methods in continual learning is the fine-tuning of the regularization strength hyperparameters (λ), which can significantly impact the performance of the model. This process involves experimentation and validation to find the right settings that provide the best trade-off between learning new information and retaining old knowledge.

Despite the effectiveness of regularization methods in dealing with catastrophic forgetting, they have certain limitations. For example, the process of determining the importance of parameters is not always straightforward and can be computationally intensive, as will be seen in the next section. Furthermore, as the number of tasks increases, the constraints imposed by the regularizer can become too restrictive, limiting the model's ability to learn new tasks. The research conducted in this thesis seeks to develop more sophisticated regularization techniques that can efficiently scale with the number of tasks and maintain a balance between stability and plasticity over long sequences of tasks. The next section provides an overview of the current state of the art in regularization-based continual learning approaches.

2.5.2 Overview of Regularization Methods in CL

Regularization methods have provided a new approach to address the catastrophic forgetting problem, especially when the network capacity is fixed and cannot be extended, and data from previous tasks are non-existent or inaccessible (e.g., for privacy reasons), preventing rehearsal. In this section, several state-of-the-art regularization-based CL methods are presented. Notably, some of these methods, such as EWC (Elastic Weight Consolidation), form the foundation for part of the work done in this thesis. These state-of-the-art methods are also used as baselines in Chapter 3 to compare with the newly developed methods.

2.5.2.1 Elastic Weight Consolidation (EWC)

A pioneering work in regularization-based continual learning method was introduced in 2017 by Kirkpatrick et al. in their seminal paper "Overcoming catastrophic forgetting in neural networks" [43]. It has since become a cornerstone in the field. They presented an algorithm named Elastic Weight Consolidation (EWC), which augments the task-specific loss with a regularization term as follows:

$$\mathcal{L}_{\text{CL}}^t(\Theta) = \mathcal{L}_{\text{Task}}^t\left(f_\Theta(x), y\right) + \frac{\lambda}{2}\sum_i \Omega_i^{1:t-1}\left(\Theta_i^{*,1:t-1} - \Theta_i^t\right)^2 \tag{2.25}$$

The regularized loss $\mathcal{L}_{\text{Reg}}(\Theta^{*,t-1}, \Theta^t)$ in Equation (2.25) can then be identified as:

$$\mathcal{L}_{\text{Reg}}(\Theta^{*,t-1}, \Theta^t) = \frac{1}{2}\sum_i \Omega_i^{1:t-1}\left(\Theta_i^{*,1:t-1} - \Theta_i^t\right)^2 \tag{2.26}$$

EWC calculates the importance of each parameter (Ω_i) through an approximation of the Fisher Information Matrix (FIM) as follows:

$$\Omega_i^t = \mathbb{E}_{(x,y)\sim D^t}\left[\left(\frac{\partial \log p(y|x,\Theta^t)}{\partial \Theta_i^t}\right)^2\right] \tag{2.27}$$

where $p(y|x,\Theta)$ is the model's predicted probability of the true label y given the input x and the model parameters Θ.

The diagonal elements of the FIM measure the curvature of the likelihood function for each parameter, which can be approximated using a Hessian matrix near the optimum. In other words, a larger change in the gradient indicates greater relevance of the corresponding parameter to previous tasks. The FIM is computed after the training of each task, which means that it cannot fully capture the learning trajectory of each network weight.

2.5.2.2 Memory Aware Synapse (MAS)

Memory Aware Synapse (MAS) [44] addresses the EWC shortcoming by computing the importance of each parameter based on the sensitivity of the output function to changes in that parameter over the entire training trajectory.

To do so, memory aware synapses proposes to accumulate the changes of each parameter across the update history. The importance of a parameter (Ω_i) is assessed based on the magnitude of the updates on that parameter, as indicated by the corresponding change in the output. Mathematically, this is computed as:

$$\Omega_i = \int \left|\frac{\partial L_{\text{MAS}}\left(y, f(x;\Theta)\right)}{\partial \Theta_i}\right| dt, \tag{2.28}$$

where the integral $\int$ is over the training trajectory, representing the accumulation of these gradients over time.

In essence, this means that, if a slight adjustment in a specific network parameter results in a substantial change in the output, that parameter should be prioritized for retaining past knowledge.

2.5.2.3 PackNet

PackNet [45] introduces a new dimension to the continual learning challenge by using network pruning – an idea borrowed from neural network compression – to increase the plasticity of the network, allowing it to accommodate a much larger number of tasks. Specifically, in addition to maintaining the performance of previously learned tasks, PackNet uses network pruning to free up redundant parameters that can then be reused to learn new tasks. This is achieved by iteratively pruning and retraining the network after each task. Retraining, which involves fewer epochs than the initial training, is necessary because pruning a network leads to performance degradation due to the sudden change in network connectivity. This effect is particularly pronounced when the pruning ratio is high.

After learning a task t, a fixed percentage of eligible weights from each layer is removed in a post-training step. To determine eligibility, the weights in a layer are sorted by their absolute magnitude, and the lowest 50% or 75% are selected for removal. This post-training pruning technique contrasts with the method presented in Chapter 3 of this thesis, where pruning is directly embedded in the loss function, eliminating the cumbersome separate pruning and retraining process.

Another significant limitation of PackNet is that during inference, the network parameters are masked to match the state learned during training for each specific task. This requires maintaining and applying task-specific masks, which adds complexity to the inference process. Furthermore, the PackNet pruning method is unable to perform simultaneous inference on all tasks, as the responses of a filter change depending on its level of sparsity and are no longer separable after passing through a nonlinearity such as the ReLU. Performing filter-level pruning, in which an entire filter is switched on or off (as done in the method presented in Chapter 3 and 4), instead of a single parameter, allows for simultaneous inference. This approach simplifies the inference process and enables the network to handle multiple tasks concurrently.

2.5.2.4 Hard Attention to the Task (HAT)

Hard Attention to the Task (HAT) adopts a similar strategy to PackNet by addressing the issue of forgetting through selective focus on task-specific parameters. HAT employs a hard attention mechanism that involves creating task-specific masks, which are learned via stochastic gradient descent. These masks determine which neurons or weights are active for a particular task, effectively "freezing" the parts of the network essential for previous tasks. This approach ensures that learning a new task does not interfere with or overwrite the critical weights associated with older tasks, thereby preserving prior knowledge.

The core of the HAT mechanism involves learning near-binary attention vectors through gated task embeddings. These vectors generate masks that constrain weight updates during training, ensuring that the critical weights for previous tasks remain unaffected. The attention vectors are learned concurrently with each task, and the cumulative attention from previous tasks conditions the learning of new tasks. The attention mask for a layer l in task t is given by:

$$a_l^t = \sigma(s \cdot e_l^t) \tag{2.29}$$

where σ is the sigmoid function, s is a positive scaling parameter, and e_l^t is the task embedding for layer l. The cumulative attention vector after learning task t is then computed as:

$$a_l^{\leq t} = \max(a_l^t, a_l^{\leq t-1}) \tag{2.30}$$

During the training of task $t+1$, the gradient $g_{l,ij}$ at layer l is modified by:

$$g'_{l,ij} = \left[1 - \min(a^{\leq t}_{l,i}, a^{\leq t}_{l-1,j})\right] g_{l,ij} \tag{2.31}$$

This ensures that weights crucial for previous tasks are not significantly altered during the learning of new tasks.

HAT differs from approaches like PackNet by eliminating the need for pre-assigned pruning ratios and allowing intermediate mask values between 0 and 1. However, HAT also has several limitations. First, the need to store attention masks and task-specific embeddings increases the storage requirements, which can be a significant drawback in environments with limited memory or storage capacity. Second, as acknowledged by the authors, HAT, while competitive, still demands considerable training time, especially when dealing with large datasets and numerous tasks. The additional steps required for learning and applying attention masks contribute to this extended training duration.

2.5.2.5 Learning Without Forgetting (LwF)

LwF [46], which involves a convolutional neural network (CNN), takes a different approach compared to the previously mentioned methods by dividing the network parameters into shared parameters (θ_s), old task-specific parameters (θ_o), and new task-specific parameters (θ_n). This division enables the network to effectively manage and update its knowledge base without requiring access to old training data.

The LwF process begins by recording the responses from the original network for the new task images. New task nodes are then added to the output layer with randomly initialized weights. The training proceeds in two steps: first, θ_s and θ_o are frozen while θ_n is trained to convergence; second, all weights comprising the network's shared parameters (θ_s), old task-specific parameters (θ_o), and new task-specific parameters (θ_n), are jointly optimized until convergence.

The loss function used for new tasks is the multinomial logistic loss:

$$\mathcal{L}_{\text{new}}(y_n, \hat{y}_n) = -y_n \cdot \log \hat{y}_n \tag{2.32}$$

For old tasks, the Knowledge Distillation loss is employed:

$$\mathcal{L}_{\text{old}}(y_o, \hat{y}_o) = -H(y'_o, \hat{y}'_o) = -\sum_{i=1}^{l} y'_o(i) \log \hat{y}'_o(i) \tag{2.33}$$

where y'_o and $\hat{y}'_o$ are softened versions of the recorded and current probabilities, respectively.

The total loss function combines the losses for the old and new tasks, balanced by a weight parameter λ_o. The resulting optimization term is as follows:

$$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \arg\min_{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n} \Big(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \Big) \ , \tag{2.34}$$

where $\mathcal{R}$ is a regularization term, typically weight decay.

A significant limitation of LwF is the need for careful balancing during the joint optimization phase. This process can be complex and often requires extensive tuning of hyperparameters, such as the balance weight (λ_o) between old and new task losses. Achieving the right balance is crucial to prevent the model from either forgetting old tasks or failing to adequately learn new tasks.

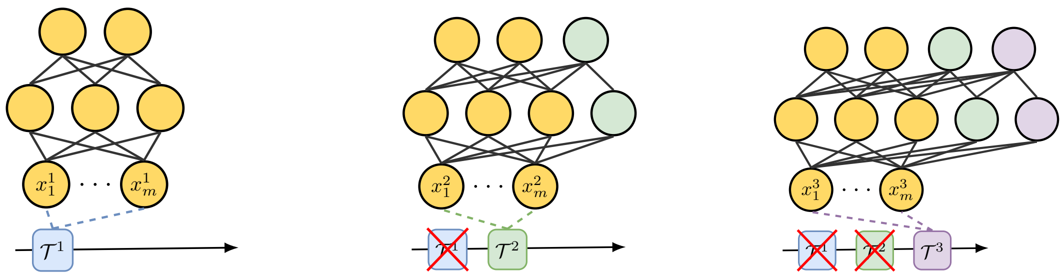
2.5.2.6 Continual Learning via Neural Pruning (CLNP)

The core idea of Continual Learning via Neural Pruning (CLNP) [47] is to leverage the overparameterization of neural networks through activation-based post-training neural pruning. This approach involves categorizing network weights into three types: active, free, and interference weights. Active weights connect active neurons, free weights connect inactive neurons, and interference weights connect active neurons to inactive neurons. By setting interference weights to zero, the free weights can be adjusted for new tasks without impacting the performance on previously learned tasks. This selective adjustment allows the network to learn new tasks while preserving knowledge from prior tasks, thus addressing the challenge of catastrophic forgetting.

To address the trade-off between model sparsity and performance, the authors introduce the concept of graceful forgetting, which allows a small, controlled amount of forgetting to regain network capacity. This approach helps prevent uncontrolled performance loss during future task training. In essence, graceful forgetting allows for a deliberate, limited reduction in accuracy to free up capacity for learning new tasks. However, determining the optimal amount of acceptable performance degradation is challenging. Excessive forgetting can significantly harm performance on previously learned tasks, while insufficient forgetting may fail to free up enough capacity, thereby limiting the model's ability to effectively learn new tasks.

2.6 Foundational Principles and Methods of Architecture Based CL Approaches

This section provides a detailed exploration of the basic principles and methods associated with architecture-based approaches in continual learning. It begins with a theoretical overview focusing on the fundamental concepts and frameworks that form the basis of architecture-based strategies in continual learning. This is followed by a detailed examination of various methods and techniques that have been developed and implemented in the field.



(a) Begining with Task 1 (b) Arival of Task 2 (c) Arival of Task 3

Figure 2.5: Illustration of architecture-based continual learning strategies. When a second task is introduced, the network is augmented with additional green neurons. Some methods freeze all yellow neurons responsible for learning the previous task, while others integrate a selection of both yellow and green neurons to form a module dedicated to mastering the second task. This process is repeated at the arrival of the third task, characterized by the expansion of the network by the addition of purple neurons. The red "X" marks over Task 1 and Task 2 in (b) and (c) indicate the absence of these tasks' data upon the arrival of Task 2 and Task 3, respectively.

2.6.1 Theoretical Foundations

Let us first introduce some formalism of architecture-based continual learning methods. An artificial neural network is presented with T task, where each task $\mathcal{T}^t$ has a corresponding function f_{Θ^t} approximating the neural network.

In architecture-based continual learning methods, the model's architecture is dynamically modified to incorporate new knowledge without disrupting previously learned information. The focus is on adjusting the neural network's structure to prevent catastrophic forgetting. This concept is inspired by the way biological brains compartmentalize learning for distinct tasks. The goal is to design neural networks that can expand or reconfigure themselves in response to new information. The underlying theory suggests that by allocating dedicated parameters or modules for different tasks, a model can retain old knowledge while effectively acquiring new information.

This modular approach involves creating distinct modules within the network, each responsible for specific tasks or subsets of tasks. Each task is addressed by combining a selection of neural modules, which can either be recycled from previous tasks or trained anew for the current task. These approaches enable the network to utilize shared components for general knowledge while preserving task-specific information. This strategy not only enhances the network's ability to learn continuously but also ensures that past knowledge is maintained alongside newly acquired skills.

Mathematically, modular networks can be thought of as a collection of subnetworks, or modules. Each module, denoted as $M_i \;\; i \in (0, \cdots, n)$, is specialized to handle a particular task or set of tasks. The entire network can then be represented as a function that either selects or combines the outputs of these individual modules:

$$f_{\Theta} = \mathcal{S}(\{M_1(x, \Theta_1), M_2(x, \Theta_2), ..., M_n(x, \Theta_n)\}), \tag{2.35}$$

where $\mathcal{S}$ is the selection or combination function that determines how the outputs of the modules are used to produce the final output. Θ_i represents the parameters specific to module M_i, and x is the input to the network.

There are various strategy how S can be implemented, including:

Hard Selection: In the simplest case, $\mathcal{S}$ could be a hard selection function that activates only one module based on the task identifier, effectively routing the input to the module trained for the current task.

$$\mathcal{S}(\{M_i\}, t) = M^t(x, \Theta^t) \tag{2.36}$$

where t is the task identifier, and M^t is the module associated with task t. This approach ensures that only the module specialized for the current task is utilized, thereby preserving the integrity of task-specific knowledge.

Soft Selection (Gating): $\mathcal{S}$ could also be a soft selection mechanism, such as a gating network that assigns a weight to each module's output based on the input. The final output is a weighted sum of all module outputs.

$$\mathcal{S}(\{M_i\}) = \sum_i g_i(x, \gamma) M_i(x, \Theta_i) \tag{2.37}$$

where g_i are gating functions parameterized by γ that produce weights for each module's output. This approach allows the network to dynamically combine the outputs of multiple modules, enabling more flexible and nuanced decision-making based on the input.

Hierarchical Combination: In hierarchical modular networks, $\mathcal{S}$ could represent a hierarchical structure where outputs of lower-level modules are combined by higher-level modules. This process repeats until the final output is produced:

$$\mathcal{S}\left(\{M_1^{(i)}\}, \{M_2^{(j)}\}, ..., \{M_n^{(k)}\}\right) = M_n^{(k)}\left(... \left(M_2^{(j)}\left(M_1^{(i)}(\mathbf{x}, \Theta_1^{(i)}), \Theta_2^{(j)}\right), ...\right), \Theta_n^{(k)}\right) \tag{2.38}$$

where $M_1^{(i)}$, $M_2^{(j)}$, ..., $M_n^{(k)}$ represent modules at different levels of the hierarchy. In this setup, the output of each module at a lower level feeds into the next module at a higher level, creating a nested structure. This hierarchical approach enables the network to build complex representations and make decisions by progressively combining simpler, lower-level features into more abstract, higher-level concepts.

The design of $\mathcal{S}$ is crucial as it dictates how the network adapts to new tasks and how it utilizes the specialization of different modules. The choice of strategy for $\mathcal{S}$ impacts the network's ability to generalize across tasks, as well as its efficiency and scalability in a continual learning context. In summary, while architectural methods show great promise in effectively addressing catastrophic forgetting, they also present challenges, particularly in terms of scalability and efficiency. As the number of tasks increases, the required model capacity grows correspondingly, which may become impractical in resource-constrained environments. Moreover, these methods often necessitate sophisticated mechanisms to integrate the outputs of various architectural components effectively.

2.6.2 Overview of Architecture-based Methods

Architectural methods offer a distinct approach to addressing the stability/plasticity dilemma in continual learning by expanding the network architecture to accommodate new tasks. This section presents several state-of-the-art architecture-based methods for continual learning, which will serve as baselines for comparison with the method developed in Chapter 4.

2.6.2.1 Progressive Neural Networks (PNN)

Pioneering contributions to expansion-based continual learning were proposed by Progressive Neural Networks (PNN) [48], where a new network is introduced for each new task. Each of these subnetworks (also called columns in a PNN) is associated with a specific task and can be conceptualized as a sequence of layers, each of which computes hidden states based on the current and all previous tasks' learned representations. Such a layer is represented as:

$$h_i^{(t)} = g\left(W_i^{(t)} h_{i-1}^{(t)} + \sum_{j<t} U_i^{(t;j)} h_{i-1}^{(j)}\right), \tag{2.39}$$

where $h_i^{(k)}$ is the hidden state of layer i in the subnetwork dedicated to task k, g is an activation function, such as ReLU or sigmoid, $W_i^{(k)}$ is the weight matrix for layer i in task t's subnetwork and $U_i^{(k;j)}$ is the lateral connection weight matrix from task j's network column to task t's network at layer i.

Since a sub-network is added as each task arrives, the total network grows linearly with the number of tasks. This continuous growth can lead to significant scalability challenges as the expanding network requires more and more computing resources and storage. In addition, managing and optimizing such a large network becomes increasingly complex, potentially impacting overall performance and efficiency.

2.6.2.2 Progress and Compress (P&C)

The scalability concern of PNN was addressed in Progress and Compress (P&C) [49], who proposed to distill the new network back to the original one after each task. To do this, they optimize the following loss with respect to the Θ_k^{KB} parameters of the knowledge base, while keeping the parameters of the active column or subnetwork unchanged:

$$\mathbb{E}[\mathrm{KL}(\pi_t(\cdot|x) \parallel \pi_{\mathrm{KB}}(\cdot|x))] + \frac{1}{2}\|\Theta_t^{\mathrm{KB}} - \Theta_{t-1}^{1/\gamma} F_{t-1}\|^2, \tag{2.40}$$

where $\pi_k(\cdot|x)$ and $\pi^{\mathrm{KB}}(\cdot|x)$ are the prediction of the active column (after learning on task t) and knowledge base respectively, $\mathbb{E}$ denotes expectation over either the dataset or the states of the environment under the active column, F_{k-1}^* is the diagonal Fisher of the online EWC Gaussian approximation resulting from previous tasks, and γ is a hyperparameter.

One of the main limitations of P&C is its fixed expansion locations. In fact, The P&C framework relies on a fixed knowledge base and active column structure. The knowledge base serves as a static repository of learned information, and the active column is the only part that gets expanded when new tasks are learned. This rigidity might limit the flexibility in choosing optimal expansion locations within the network, potentially missing opportunities for more granular or localized expansions in different layers or parts of the network.

2.6.2.3 Compack, Picking and Growing (CPG)

Another approach to addressing the scalability issue has been proposed in CPG [50], a method that integrates model compression, critical weight selection, and progressive network expansion. During the compacting phase, the model undergoes gradual pruning to eliminate redundant weights, which compacts the model and releases some weights that can be used for new tasks. In the picking phase, critical weights from previously learned tasks are selected via a learnable binary mask. This mask helps identify which weights are essential for retaining previous knowledge while allowing the integration of new information. By preserving only the crucial weights, CPG avoids the inertia effect caused by retaining all old-task weights.

Before training on each task t, an accuracy goal is set for that task. If the released weights are insufficient to meet this accuracy goal, the model architecture is expanded. However, setting an accuracy goal for each task poses a significant limitation for CPG in determining when to expand the network, as it requires prior knowledge of the task's difficulty. Additionally, such a fixed criterion may not be adaptive enough to respond to varying task complexities or dynamic changes in data distributions. Consequently, the network might either expand too frequently, leading to unnecessary complexity, or fail to expand sufficiently, hindering performance on new tasks.

2.6.2.4 DYnamic TOken eXpansion (DyTox)

Another notable architectural approach, termed DyTox, presented in [51], involves a transformer architecture. It addresses the challenge of catastrophic forgetting by leveraging a task-specific token expansion mechanism. The architecture shares self-attention layers across all tasks and employs task-specific tokens to achieve task-specialized embeddings. This design ensures that the core model can learn generalized features applicable to a wide range of tasks, while the task-specific tokens allow for fine-tuning to the nuances of each individual task without interfering with the learned representations of previous tasks. The task-specific tokens are dynamically added to the model for each new task, allowing the network to develop specialized embeddings for these tasks. By maintaining a shared set of parameters for common features and adding specific tokens for new tasks, DyTox effectively balances the need for model plasticity and stability, ensuring high performance across all tasks without suffering from catastrophic forgetting.

DyTox like PNN expands the network with the introduction of each new task. However, this approach might not be optimal for all scenarios. The model lacks a sophisticated heuristic to determine the precise necessity and timing for expansion based on task complexity or data distribution changes. This could result in either premature expansion, leading to unnecessary complexity, or delayed expansion, hindering performance on new tasks. Furthermore, regarding where to expand the networks, DyTox primarily focuses on expanding through task-specific tokens. While this method is efficient, it might not fully exploit the potential of expanding other parts of the network.

2.6.2.5 FOSTER

Feature Boosting and Compression for Class-Incremental Learning (FOSTER) [52] draws inspiration from gradient boosting algorithms and incorporates two primary stages: feature boosting and feature compression. In the feature boosting stage, FOSTER retains the old model with its parameters frozen and adds a new module to fit the residuals between the target and the output of the original model. This approach enables the model to adapt to new tasks without forgetting previously learned information. The compression stage is crucial for long-term incremental learning, addressing the exponential growth of parameters and feature dimensions that can result from continuously adding new modules. FOSTER employs a distillation strategy to effectively transfer knowledge from the expanded model to a single, more compact model, ensuring minimal performance degradation. This approach allows FOSTER to maintain a manageable model size while preserving accuracy across both old and new tasks.

However, FOSTER expands the network with each new task based on a predetermined approach of adding new modules to fit residuals. This method may not be sufficiently adaptive to respond dynamically to varying complexities or changes in data distribution, potentially leading to either premature or delayed expansion.

2.6.2.6 Hierarchical Combination

Another form of architecture-based continual learning approach is hierarchical combinations, typically implemented in the form of a tree structure, which allows for specialized learning at different levels of abstraction [53]. An illustrative example is a network that begins by learning to distinguish between broad biological categories, such as vertebrates and invertebrates. Once this foundational knowledge is established, the network can branch out within the vertebrate category to further specialize, for instance, distinguishing between birds and fish. In this hierarchical model, the "child" network, which focuses on more specialized tasks, can effectively build on and leverage the foundational knowledge already acquired by the "parent" network. This minimizes the need for additional network capacity in the child network, as it extends the existing knowledge base rather than starting from scratch.

The tree-based expansion approach has received limited attention in the existing literature, primarily due to its narrow applicability. However, it is a promising tool when dealing with hierarchically structured datasets, where knowledge can be efficiently transferred and specialized across different levels of the hierarchy.

2.7 Interplay of Regularization- and Architectured-based CL Methods

A major drawback of the regularization-based approach is that the network is considered fixed and may reach saturation at some point, making network expansion inevitable. This is particularly true when (i) there is substantial variability in tasks complexity depending on the initial capacity of the network, or (ii) the number of learning tasks is infinite. Although the latter is a theoretical scenario, it underscores the inherent constraints of regularization methods.

`Variability in task complexity:` As discussed earlier, the lower layers of neural networks capture basic features that are progressively combined in higher layers to form more complex representations. For instance, in a regularization-based network initially trained to identify cats in images, the lower layers would detect features like edges, textures, and basic shapes such as whiskers and ears. When the network is later tasked with identifying dogs, it can leverage these same lower-level features, as both tasks share similar visual elements. However, if the network's new task is to recognize cars in images, it may require a different set of low-level features, such as straight lines, curves, and textures specific to car designs. This shift could demand increased model capacity, such as additional filters in a CNN, to accommodate the unique requirements of the new task, necessitating a combination of the regularization approach with an expansion-based approach.

`Large number of task:` As discussed in Section 2.5, regularization-based CL works by identifying parameters essential for learning previous tasks and constraining them to limit significant changes during the learning of new tasks. However, as the number of tasks grows, so does the need for additional network capacity. If the number of tasks is infinite, the network will eventually reach a point where its entire capacity is dedicated to past tasks, leaving no room for learning new ones. At this stage, combining regularization-based methods with architectural expansion strategies may become necessary to accommodate further learning without compromising previously acquired knowledge.

In both cases –variability in task complexity and a large number of tasks– there is a need to identify the saturation point. In other words, a heuristic is required to determine when to expand the network. Few works have directly addressed this issue. Among the most notable are DEN [54] and CPG [50], which proposed heuristic algorithms for automatically adding and pruning weights. These methods are similar to the approach presented in Chapter 4, but with key differences, particularly in the heuristics regarding when and where (at which layer) the network should be expanded.

For instance, DEN [54] selectively retrains, expands, and splits or duplicates neurons based on task requirements, effectively mitigating semantic drift and achieving significant performance improvements over existing methods with fewer parameters. On the other hand, CPG [50] iteratively combines deep model compression via weight pruning (Compacting), critical weight selection (Picking), and progressive network expansion (Growing). Both methods, however, rely on a threshold for the loss function and training accuracy, respectively, beyond which the network is expanded. These strategies require an a priori estimation of task difficulty, which limits their practicality in real-world applications.

2.8 Overview of Neural Network Model Compression

Neural network model compression [55, 56] involves techniques designed to reduce the size and complexity of neural networks while maintaining or even enhancing their performance. Initially, these techniques were primarily applied to deploy models on devices with limited computational resources, such as mobile phones and embedded systems. However, model compression has also become increasingly relevant in other areas of machine learning, including continual learning, where the goal is to minimize network size and complexity to facilitate efficient and effective task learning over time.

In the methods developed in Chapters 3 and 4, model compression is leveraged to sparsify the neural network, thereby improving its plasticity and adaptability. These approaches are further elaborated in the respective chapters.

The remainder of this section explores state-of-the-art techniques for model compression in the context of continual learning.

2.8.1 Weight Pruning

Weight pruning consists of removing (pruning) less significant weights (typically those close to zero) from a neural network [57, 58]. This is typically done by minimizing the loss function of the form:

$$\min_{W} \mathcal{L}(W) + \lambda \|W\|_0, \tag{2.41}$$

where W represents the weights of the neural network, $\|W\|_0$ is the ℓ_0 norm, which counts the number of non-zero weights, and λ is a regularization parameter controlling the level of sparsity.

After pruning, models are typically retrained or fine-tuned to recover any performance degradation. In continual learning, this process must be repeated for each new task, leading to substantial computational overhead. Additionally, pruning must be carefully managed to avoid excessive loss of critical information, particularly when applied across multiple tasks.

2.8.2 Knowledge Distillation

In knowledge distillation, a smaller student network S is trained to replicate the output of a larger, more complex teacher network T [59, 60]. This approach compresses neural networks by transferring knowledge from the teacher model to the student model, thereby achieving a more efficient model with reduced complexity. The training objective typically takes the form:

$$\mathcal{L}_{KD} = \alpha \mathcal{L}_{\text{CE}}(S(x), y) + \beta \mathcal{L}_{\text{KL}}(S(x), T(x)), \tag{2.42}$$

where $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss between the student network's predictions $S(x)$ and the true labels y, and $\mathcal{L}_{\text{KL}}$ is the Kullback-Leibler divergence between the student network's output $S(x)$ and the teacher network's output $T(x)$. The parameters α and β control the balance between these two objectives.

While knowledge distillation offers an effective way to compress models, it often requires access to the original training data used for the teacher model. In continual learning, particularly in privacy-sensitive applications, this may not be feasible due to data privacy regulations and constraints, making distillation more challenging. Additionally, finding the optimal balance between the student model's simplicity and performance remains a key challenge.

2.8.3 Quantization

Quantization consists of reducing the precision of the weights and activations in a neural network, thereby decreasing the model's size and computational requirements [61, 62]. For example, converting 32-bit floating-point weights W to 8-bit integers $\bar{W}$:

$$\bar{W} = round\left(\frac{W - W_{\min}}{W_{\max} - W_{\min}} \times (2^k - 1)\right), \tag{2.43}$$

where $W_{\min}$ and $W_{\max}$ represent the minimum and maximum weight values, k denotes the number of bits used for quantization (e.g., 8 for 8-bit quantization), and *round* refers to the rounding operation.

Although quantization is highly effective for compressing models, it has several challenges in continual learning scenarios. Reducing weight precision can lead to a loss in model accuracy, which is particularly problematic when high performance across multiple tasks must be maintained. Additionally, many continual learning algorithms depend on precise weight updates, and quantization can interfere with the effectiveness of techniques such as Elastic Weight Consolidation (EWC) [43] or gradient-based methods, thereby reducing their ability to retain knowledge from previous tasks. Therefore, careful consideration is needed when applying quantization in the CL contexts.

2.9 Datasets and Metrics

This section outlines the datasets, state-of-the-art baseline methods, and performance metrics employed to evaluate the methods developed and implemented in this thesis.

2.9.1 Datasets

Table 2.1 summarizes the datasets frequently referenced in the continual learning literature and used in the experiments conducted in this thesis. These datasets were selected for their diversity, complexity, and simulation of real-world scenarios. Illustrative examples of these datasets are shown in Figure 2.6.

Table 2.1: Summary of the datasets used in the thesis

Dataset	**Description**	**# Image**	**Dimen-sions**
MiniImageNet [63]	A subset of the ImageNet dataset, challenging for CL due to a large number of classes and limited data instances per class.	100,000	84×84
Cifar-10 [64]	Contains color images divided into distinct object classes, with a simple structure.	60,000	32×32
Cifar-100 [64]	Similar to Cifar-10 but with more categories.	60,000	32×32
SVHN [65]	Features images of house numbers from Google Street View, centered on single digits.	600,000	32×32
Facescrub [66]	Tailored for face recognition, includes celebrity face images from online sources.	107,000	32×32
Traffic Sign [67]	German traffic sign recognition benchmark dataset contains images have varying resolutions and conditions, including different lighting and weather scenarios.	$50,000$	
Omniglot [65]	Handwritten characters from 50 different alphabets, both real and fictional.	32,460	28×28
ImageNet32 [68]	A downsampled (32×32) version of the entire ImageNet [69].	$161,065$	32×32
Fashion Mnist [70]	Dataset of Zalando's article images similar to MNIST in size and format but consisting of 10 more challenging classes, such as t-shirts, trousers, and shoes.	$70,000$	28×28
Mnist [71]	Consists of grayscale images of handwritten digits, widely used for benchmarking image processing systems.	$70,000$	28×28
Permuted MNIST [72]	A permuted version of the MNIST dataset where the pixels of each image are randomly shuffled. This represents a domain incremental continual learning scenario, where the shuffled images create tasks from the same classes but with varied input domains.	$70,000$	28×28
NotMnist [71]	Contains images of letters from A to J taken from different fonts.	$500,000$	28×28

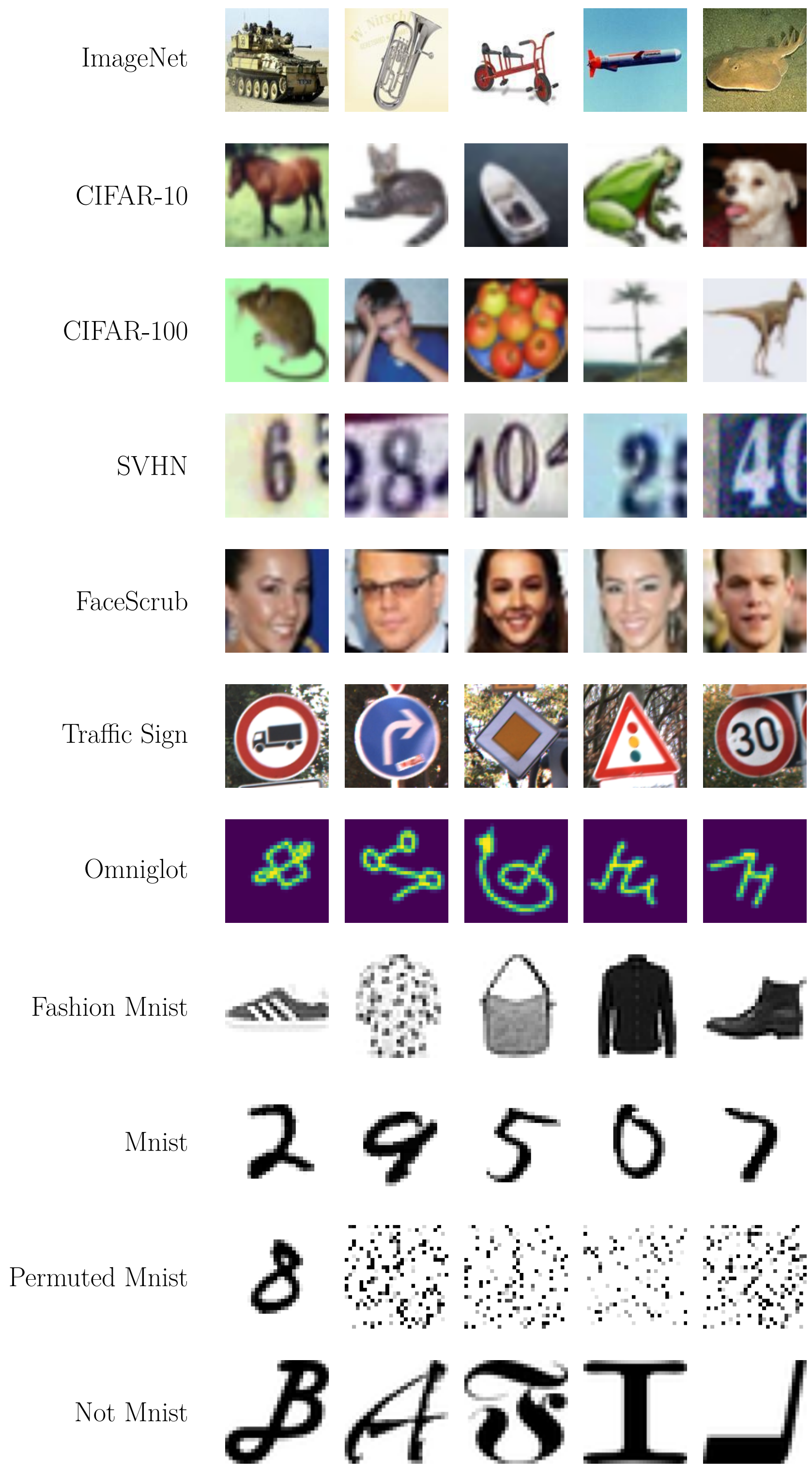


Figure 2.6: Sample images from the datasets used in this thesis.

2.9.2 Baselines

Table 2.2 summarizes the baselines used in the experiments conducted in this thesis. These baselines were chosen for their strong performance as state-of-the-art methods in the field. The selection process involved a comprehensive review of recent and influential papers in continual learning, focusing on approaches similar to those developed in this thesis, which are then used as benchmarks.

Table 2.2: Summary of the baselines used in the thesis.

Method	**Description**
`Elastic Weight Consolidation (EWC)` [43]	EWC pioneers the use of a prior-focused strategy in regularized-based continual learning, where a model's updates are guided not only by the new data or task at hand, but also by a weighted consideration of what the model has previously learned.
`Learning Without Forgetting (LWF)` [46]	LWF uses knowledge distillation, a data-driven regularization technique, to prevent forgetting.
`Memory Aware Synapse (MAS)` [44]	MAS introduces a mechanism for weighting the importance of synaptic connections (i.e., the parameters of a neural network) based on their contribution to the previously learned tasks.
`Hard Attention to the Task (HAT)` [73]	HAT introduces a hard attention mechanism that selectively activates different parts of the neural network for different tasks. This mechanism is based on the idea that if a subset of network parameters is responsible for a specific task, then protecting these parameters during subsequent learning should prevent forgetting.
`PackNet` [45]	PackNet iteratively prunes and retrains the network architecture to pack multiple tasks into a single network.
`CL via neural pruning (CLNP)` [47]	CLNP is based on the sparsification of neural models. It trains subsequent tasks using the inactive filters of the sparsified network.
`Progress and Compress (P&C)` [74]	Combine expansion and experience replay. It creates a new network (ResNet) per task. All ResNets' embedings are concatenated and passed to a unique classifier.
`Feature boosting and Compression (FOSTER)` [75]	FOSTER dynamically expands new modules to fit the residuals of the old model and then distills them into a single model.
`Dynamic Token Expansion (DyTox)` [51]	DyTox expands task tokens for each new task, which requires much less memory than saving the entire backbone.
`Compact Pickinig and Growing (CPG)` [50]	CPG uses model compression and progressive network expansion to provide scalable incremental learning that prevents forgetting and maintains model compactness.

2.9.3 Evaluation Metrics

To conclude this chapter, this section outlines the metrics used to evaluate the continual learning methods developed in this thesis. These metrics serve specific purposes aimed at understanding the algorithm's ability to learn sequentially over time without forgetting previously acquired knowledge. Each metric provides insight into different aspects of the algorithm's performance.

1. `Average Accuracy (ACC):` The average classification accuracy for task t is calculated as the mean test accuracy over tasks ranging from 1 to t [76]. Let $a^{t,t'}$ denote the accuracy –measured as the fraction of correctly classified images– on the test set of task t', where $t' < t$, after the model has undergone sequential training on tasks $\mathcal{T}^1$ through $\mathcal{T}^t$. The average accuracy $A^t \in [0,1]$ for task t can then be mathematically defined in terms of $a^{t,t'}$ as follows:

$$A^t = \frac{1}{t}\sum_{t'=1}^{t} a^{t,t'} \tag{2.44}$$

2. `Forgetting (F):` Forgetting for task t' at task t, where $t' < t$, is quantified as the decline in performance between the peak accuracy achieved for t' and it accuracy measured after the model has been successively trained up to task t [34]. This can be formulated as:

$$f^{t,t'} = \max_{i\in\{1,\cdots,t-1\}} a^{i,t'} - a^{t,t'}\ , \quad \forall\ t' < t. \tag{2.45}$$

   The average forgetting after learning task t (defined as the decrease in performance on each task between peak accuracy and accuracy after learning all t tasks) can then be expressed as:

$$F^t = \frac{1}{t-1}\sum_{t'=1}^{t-1} f^{t,t'}. \tag{2.46}$$

3. `Forward Transfer (FWT):` Relative to the notion of backward transfer, forward transfer is defined as the difference in accuracy between the model just before training on a given task and the accuracy produced by a randomly initialized network [76, 34].

   In other words, forward transfer quantifies the impact on the performance of subsequent tasks $t'' > t$ when the model is in the process of learning the task t:

$$FWT^{t,t''} = a^{t-1,t''} - \bar{b}^{t''}, \tag{2.47}$$

   where $\bar{b}^{t''}$ is the accuracy on task t'' of a random baseline. The average forward transfer is then defined as:

$$FWT^t = \frac{1}{t-1}\sum_{t''=2}^{t-1} FWT^{t''-1,t''} \tag{2.48}$$

Although it's arguable that the ability to classify unseen classes is valuable, the relevance of such a metric is highly context dependent. In class incremental and task incremental learning, forward transfer can be beneficial when tasks share similar visual features across classes, such as identifying different species of birds in successive tasks. However, when tasks involve entirely distinct classes, like moving from identifying fruits to recognizing types of vehicles, forward transfer becomes more challenging. On the other hand, forward transfer gains greater importance in domain-incremental learning scenarios, where the same class appears but under varying input conditions.

2.10 Chapter Summary

In this chapter, background knowledge on catastrophic forgetting and continual learning in neural networks was introduced. Different forms of catastrophic forgetting –task incremental learning, class incremental learning, and domain incremental learning– were discussed in Section 2.3. Section 2.4 covered the definition, challenges, history, and motivation of continual learning. Section 2.4.3 outlined the desired properties of the continual learning methods, both in general and those developed in this thesis.

It was introduced that continual learning research is categorized into three main groups of methods: regularization-based, architecture-based, and memory-based. Memory-based methods are not discussed in this thesis due to their large storage requirements and potential privacy violations.

Section 2.5 provided an overview, fundamental principles, and a review of related work on regularization-based methods. Key methods discussed included Elastic Weight Consolidation (EWC) [43], which computes weight importance via the Fisher information matrix. This matrix is computed after each task but fails to capture the learning trajectory of each network weight. Memory Aware Synapse (MAS) [44] addresses this issue by computing parameter importance based on output sensitivity over the entire training trajectory. Other methods discussed included Synaptic Intelligence (SI) [77], which computes weight importance during training, considering contributions to changes in the loss function. Less Forgetting Learning [78] retains old task knowledge by minimizing the gap between activations of previous and current models. This review identified that these methods, along with existing regularization-based fixed-capacity neural network approaches in general, do not sufficiently address the plasticity challenge in continual learning.

Section 2.6 provided an overview, fundamental principles, and a review of related work on expansion-based methods. The requirements for a sustainable expansion-based method –when, where, and how to expand the network– were introduced. The review included key methods such as Progressive Neural Networks (PNN) [48], which introduce a new subnetwork for each new task but do not address the "when" aspect, leading to scalability challenges. Other methods discussed included Progress and Compress (P&C) [49], which addresses the "when" requirement by introducing a threshold on the loss function and training accuracy, beyond which the network is expanded. However, this strategy requires a priori estimation of task difficulty, limiting its practicality. PathNet [79] employs agents to identify and reuse parts of the network for new tasks, addressing the “where” aspect but not fully addressing “when” and “how”. Hierarchical combinations [53], implemented as tree structures, address the question of “where” to expand the network by leveraging foundational knowledge for specialized tasks. However, they do not address the “when” and “how” in a scalable manner. This review highlights a notable gap in current research: none of the existing expansion-based approaches fully address all three critical requirements: “when”, “where”, and “how” to expand the network.

The chapter concluded with Section 2.9, which discussed the datasets, and metrics used to evaluate the methods developed in this thesis, as well as the compared baselines.

Chapter 3

A New Regularization-based CL Method for Static Neural Networks

The present chapter provides a novel regularization-based approach, termed "*Group and Exclusive Sparse Regularization-based Continual Learinig (GESCL)*", aimed at improving continual learning systems in fixed-capacity neural networks. Using extensive and diverse datasets, the proposed approach is benchmarked against state-of-the-art continual learning methods.

Parts of this chapter have been published in [1] and [2].

3.1 Problem Statement

As introduced in Chapter 2, regularisation-based continual learning approaches protect acquired knowledge from catastrophic forgetting by detecting parameters that are important for past tasks and preventing them from changing too much when learning new tasks. To recapitulate, catastrophic forgetting refers to the abrupt erasure of knowledge related to previous tasks (e.g., knowledge from task 1, $\mathcal{T}^1$) while the network assimilates information relevant to the current task (e.g., a new different task $\mathcal{T}^2$). As discussed in Chapter 2, this typically occurs due to changes in the data distribution between $\mathcal{T}^1$ and $\mathcal{T}^2$, leading to a significant discrepancy in the optimal weights required for each task (the set of weights to be learned for $\mathcal{T}^2$ is radically different from those learned for $\mathcal{T}^1$).

Next, a detailed formal mathematical examination of the problem of catastrophic forgetting as it manifests in the sequential learning of tasks with different data distributions will be undertaken. Consider the problem of learning a function from a data stream which, as defined in Section 2.1, consists of a series of tasks $\{\mathcal{T}^1, \cdots, \mathcal{T}^T\}$, where each task consists of a collection of sample pairs $\mathcal{D}^t = \{(X_i^t, y_i^t)\}_{i=1}^{m^t}$, where m^t is the number of samples in D^t and $\mathcal{D}^t \cap \mathcal{D}^{t'} = \emptyset \ \ \forall \ \ t, t' \in \{0, \cdots, T\}$.

Given $f(\cdot;\Theta^t)$ as the neural network learner at learning task t, the goal of continual learning is to maximize the model's performance on the current task $\mathcal{T}^t$, while concurrently minimizing the forgetting on prior tasks $\mathcal{T}^1, \cdots, \mathcal{T}^{t-1}$, as assessed on their respective test sets $D^{t'}$ with $t' \in \{1, \cdots, t\}$. In this context, the learning objective for a single task $\mathcal{T}^t$ is expressed as:

$$\begin{aligned}\Theta^{*,t} &= \arg\min_{\Theta^t} \mathcal{L}^t(\Theta^t, D^t_{train}) \\ &= \arg\min_{\Theta^t} \frac{1}{m^t} \sum_{i=1}^{m^t} l_i\left(f\left(x_i^t, \Theta^t\right), y_i^t\right),\end{aligned} \tag{3.1}$$

where l_i denotes the loss function (typically cross-entropy) for the i^{th} training sample. Within the continual learning framework, the ultimate goal after sequential training on all T tasks is to minimize the cumulative loss across all tasks. Formally, this is expressed by the following objective:

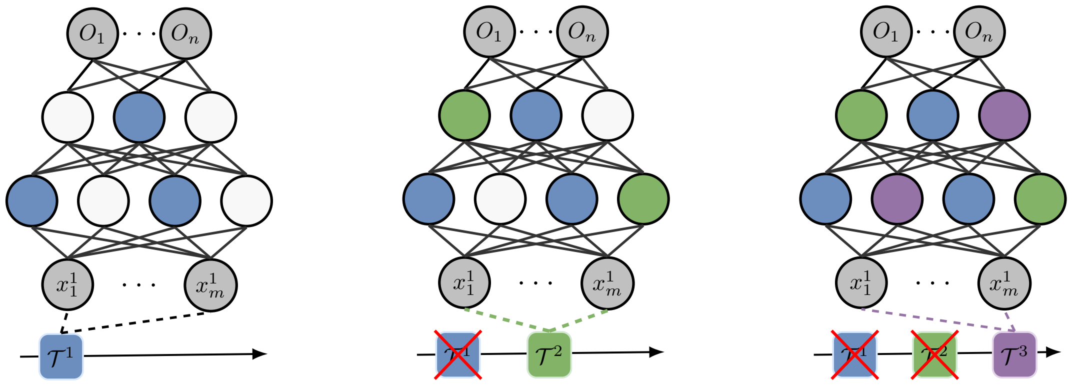
$$\begin{aligned}\Theta^{*,1:T} &= \arg\min_{\Theta^{1:T}} \sum_{t=1}^{T} \mathcal{L}^t(\Theta^t, D^t_{train}) \\ &= \arg\min_{\Theta^{1:T}} \sum_{t=1}^{T} \frac{1}{m^t} \sum_{i=1}^{m^t} l_i\left(f\left(x_i^t, \Theta^t\right), y_i^t\right).\end{aligned} \tag{3.2}$$

However, given the challenging setting of this thesis, where D^t_{train} becomes inaccessible after learning task t, the objective function in Equation (3.2) cannot be directly minimized. Therefore, when training on $\mathcal{T}^t$, the challenge is to (i) stabilize previous knowledge $\sum_{t'=1}^{t-1} \mathcal{L}^{t'}(\Theta^{t'}, D^{t'}_{train})$ without explicitly measuring it (since $D^{t'}_{train}$ is not available), while at the same time (ii) accurately estimating Θ^t by optimizing Equation (3.1). Addressing these two aspects ensures that the model retains its effectiveness on earlier tasks when the current task t is solved.

The next section describes how the GESCL method developed in this chapter addresses this challenge. Another major weakness of regularization-based continual learning approaches is that they focus on the stability of the network, thus largely neglecting its plasticity for learning new tasks. The approach presented in this chapter overcomes this shortcoming by introducing a sophisticated plasticity regularizer that ensures a sustainable use of network capacity during training on each task. This is presented in detail in Section 3.2.5.

3.2 Method: Group and Exclusive Sparse Regularization-based CL (GESCL)

This section describes a novel approach to the problem of continual learning in fixed-capacity neural networks. The cornerstone of this method is the newly derived filter/neuron importance measure, along with two regularization terms designed to maintain the stability and plasticity of the network throughout training on each new task.



(a) Beginning with Task 1 (b) Arrival of Task 2 (c) Arrival of Task 3

Figure 3.1: Overview of the proposed Group and Exclusive Sparse Regularization-based Continual Learning (GESCL) approach. A fixed capacity neural network with m and n input and output neurons is given. After learning the first task $\mathcal{T}^1$, GESCL identifies neurons (highlighted in blue) that were important in learning this task. A plasticity constraint embedded in the loss function ensures that only a small subset of the total free neurons (shown in white) are selected for learning each task, starting from $\mathcal{T}^1$. This ensures that there is enough remaining capacity to learn new tasks - here $\mathcal{T}^2$ (green neurons) and $\mathcal{T}^3$ (purple neurons). As each task is learned, a stability regularizer embedded in the loss function uses the identified critical neurons from past tasks to prevent catastrophic forgetting on those tasks. This synergy between the stability and sparsity regularizers allows GESCL to efficiently manage resource allocation across multiple tasks during the continual learning experiments. In the case of a convolutional neural network, a neuron as described here corresponds to a 2D filter. The red "X" marks over Task 1 and Task 2 in (b) and (c) indicate the absence of these tasks' data upon the arrival of Task 2 and Task 3, respectively.

3.2.1 Preliminaries

At the start of this thesis in the late 2019s, the most established continual learning system for mitigating catastrophic forgetting in fixed-capacity neural networks was a technique known as "elastic weight consolidation" (EWC) [43], introduced by James Kirkpatrick and colleagues. EWC [43] not only pioneered regularization-based approaches but also remains the foundation for many contemporary regularization-based continual learning systems.

As mentioned in Chapter 2, EWC builds on the concept of regularization, a common mechanism in machine learning to prevent overfitting, repurposing it to prevent forgetting past knowledge by utilizing the inherent overparameterization of neural networks (neural networks often have more parameters than needed to learn a given task [42]). Parameters important for past tasks are constrained to change minimally, while superfluous parameters can be used to learn the current task. This principle forms the essence of EWC [43] and serves as the basis for most state-of-the-art regularization-based continual learning approaches to overcoming catastrophic forgetting, including the method described in this chapter.

3.2.2 From Weights to Filter/Neuron Importance

EWC, along with other reference regularization-based methods such as SI [77], MAS [44], VCL [80], or PIGWM [81], assigns importance to each individual weight within a neural network. Specifically, for convolutional networks, this process involves assigning importance to individual weights within a convolutional filter.

This approach, however, neglects the inherent structure of the spatial hierarchies and patterns in the data. Weights within a convolutional filter are not isolated; instead, they work together. Changing a single weight (unimportant for the previous task) when learning a new task affects not only the function of the filter, but also the effectiveness of subsequent layers. Thus, evaluating weights on a individual basis does not effectively represent these nuanced interdependencies. In this sense, the research presented in this thesis proposes that in convolutional neural networks, weights within the same filter should be uniformly significant, which is translated into the notion of filter importance.

A similar conclusion was reached for densely connected networks. Specifically, by treating the weights associated with a neuron as a collective entity rather than as discrete units, as previous studies have done, a more accurate representation of the learned task is achieved. This has led to the concept of neuron importance. The following section describes the techniques used in this thesis to estimate the importance of each filter or neuron in a neural network during the continual learning experiments.

3.2.3 Estimating Filter/Neuron Importance

To evaluate the importance of each filter for a given task, the cumulative post-activation values of that filter or neuron across all samples within the task's training set are averaged. This approach differs from existing methods like Elastic Weight Consolidation (EWC) [43], which uses the Fisher Information Matrix to estimate parameter importance; Synaptic Intelligence (SI) [77], which measures the contribution of each weight to the reduction of the loss function over time; or Memory Aware Synapses (MAS) [44], which assesses the sensitivity of the output function to changes in each parameter. By focusing specifically on the cumulative post-activation values of the entire filter rather than dealing with individual weights, this method provides a unique perspective on filter significance.

Unlike Fisher Information (as used in EWC, the foundation of regularization-based continual learning methods including GESCL), which requires access to gradients and labels to compute per-parameter importance scores, the proposed post-activation-based method is computationally more efficient and better aligned with the filter- or neuron-level consolidation approach adopted in GESCL. In particular, GESCL uses the standard deviation of activation maps (a simple, interpretable measure of activation variability defined in Equations (3.4) and (3.3)), as a proxy for importance. This avoids gradient computation entirely and yields a natural scalar relevance value per filter or neuron.

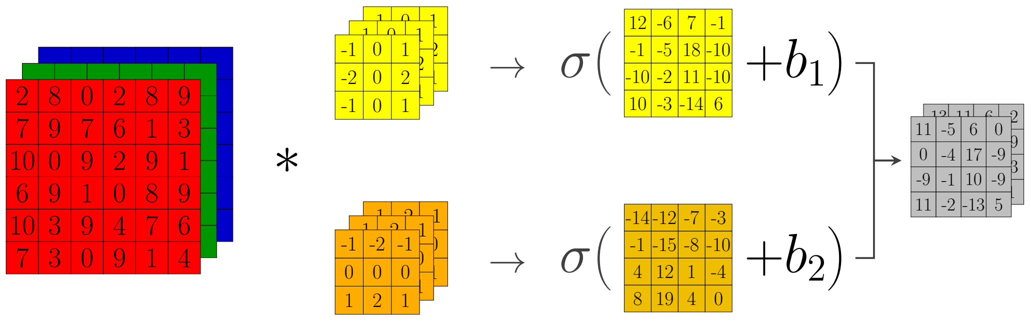


Figure 3.2: Visual representation of a convolutional layer for a single sample image in a convolutional neural network: starting with a three-channel input volume, two sets of filters (yellow and orange) are applied to the input, producing feature maps. Bias terms ($b_1$and b_2) are added to the maps, followed by an activation function (σ). The final output is a combined feature map, illustrating the key operations of filtering, bias addition, and activation.

3.2.3.1 Intuition Behind $\Omega^t_{l,f}$

The use of activation variability (quantified via standard deviation) as an indicator of filter/neuron importance is grounded in the idea that informative filters or neurons respond selectively to input patterns. A filter or neuron that produces nearly constant (i.e., low-variance) activations across diverse inputs is unlikely to capture task-relevant features. Conversely, filters/neurons that exhibit high variation in their activation values are more responsive to specific structures in the data and thus more likely to contribute meaningfully to task performance.

Standard deviation is used in GESCL as a principled and interpretable summary statistic for activation variability: it is scale-aware, sensitive to outliers, and differentiable, making it suitable for identifying discriminative filters or neurons in neural networks.
This intuition aligns with the behavior of ReLU activations, which induce sparsity by suppressing negative activations. A consistently inactive filter or neuron (i.e., producing near-zero output) will naturally have low standard deviation and thus low importance under the proposed filter- or neuron-level relevance measure.

3.2.3.2 Filter Importance $\Omega^t_{l,f}$ at Learning Task t ($\mathcal{T}^t$)

When a filter is applied to the input image (or the output of a previous layer), it convolves over the width and height of the input, computing a dot product between the filter weights and the input at each location. This process produces a two-dimensional matrix, where each element represents the response of the filter at a given location in the input. Each filter thus produces a two-dimensional activation map. An activation function (typically ReLU) is applied to this activation map to produce the final filter output activation $a_{l,f} \in \mathbb{R}^{H_o \times W_o}$. This process is illustrated in Figure 3.2, where the final activation map, shown in gray, has a size of $4 \times 4 \times 2$. As in other neural network architectures, the purpose of the activation function is to introduce nonlinearity into the model, allowing it to learn and represent more complex patterns.

In light of the above, the importance $\Omega^t_{l,f} \in [0,1]$ of a filter f in layer l at learning task t is quantified as the average standard deviation of the filter's post-activation value over all training samples of task t. More specifically, the standard deviation $\sigma^t_{l,f} \in \mathbb{R}^{H_o \times W_o}$ of each filter's output activation (activation map followed by ReLU) $a_{l,f} \in \mathbb{R}^{m^t \times H_o \times W_o}$ over all m^t training samples is computed as:

$$\sigma^t_{l,f} = \sqrt{\frac{1}{m^t} \sum_{s=1}^{m^t} (a_{l,f}(x^t_i) - \overline{a_{l,f}})^2}, \tag{3.3}$$

where $a_{l,f}(x^t_i) \in \mathbb{R}^{H_o \times W_o}$ is the filter output activation produced by applying the filter f in layer l for the training instance $x^t_i \in D^t_{\text{train}}$, and $\overline{a_{l,f}}$ is the average activation over all training samples.

The filter importance $\Omega^t_{l,f} \in [0,1]$ is then obtained by averaging the element of the 2D tensor $\sigma^t_{l,f}$ as:

$$\Omega^t_{l,f} = \frac{1}{H_o \times W_o} \sum_{q \in \sigma^t_{l,f}} q \tag{3.4}$$

with

$$0 \leq \Omega^t_{l,f} \leq 1, \quad \forall l \in \{1, \ldots, L\}, \ \forall f \in \{1, \ldots, F_l\}. \tag{3.5}$$

Note that for dense layers, Equation (3.3) is sufficient to compute the importance of each neuron based on its post-activation values.

3.2.3.3 Filter Importance $\Omega^{1:t}_{l,f}$ at Learning Task $\mathcal{T}^1$ up to $\mathcal{T}^t$

Once the importance $\Omega^t_{l,f}$ in learning task t only is computed, it is crucial to consider the importance of this filter in learning previous tasks $\mathcal{T}^{1:t-1}$ ($\mathcal{T}^1$ to $\mathcal{T}^{t-1}$) so that it remains stable in performing these tasks. For this purpose, after learning each task t, the importance $\Omega^{1:t}_{l,f}$ of the filter f in learning tasks $\mathcal{T}^1$ to $\mathcal{T}^t$ is updated as:

$$\Omega^{1:t}_{l,f} := \nu \ \Omega^{1:t-1}_{l,f} + \Omega^t_{l,f}, \tag{3.6}$$

where $\Omega^{1:t-1}_{l,f}$ is the importance of f at learning tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$ and ν is a hyperparameter balancing the filter importance before and after training on task t.

The hyperparameter ν is typically determined through a combination of cross-validation and empirical testing. This process involves experimenting with different values of ν and selecting the one that optimizes the performance of the model on a validation set, ensuring a balance between retaining past knowledge and integrating new information.

Finally, to represent the importance $\Omega^{1:t}$ of all filters in the CNN after learning tasks 1 to t, the importance values can be organized into a matrix of vectors (one for each layer), where each element in the vector corresponds to the importance of a filter in a specific layer. Formally, $\Omega^{1:t}$ can be represented as a collection of matrices for each layer l:

$$\Omega^{1:t} = \{\Omega_l^{1:t}\}_{l=1}^{L}, \tag{3.7}$$

where each $\Omega_l^{1:t}$ is a vector of dimensions F_l for layer l, representing the importance of each filter in that layer:

$$\Omega_l^{1:t} = \left[\Omega_{l,1}^{1:t}, \Omega_{l,2}^{1:t}, \ldots, \Omega_{l,F_l}^{1:t}\right]. \tag{3.8}$$

3.2.4 Mitigating Forgetting in Sequential Task Learning

To overcoming catastrophic forgetting, the focus is on preserving performance on previous tasks by stabilizing the cumulative loss across them $\sum_{t'=1}^{t-1} \mathcal{L}^{t'}(\Theta^{t'}, D_{train}^{t'})$ (despite the inaccessibility of $D_{train}^{t'}$ for direct measurement), while simultaneously estimating the solution $\Theta^{*,t}$ of the current task $\mathcal{T}^t$ through optimization.

To stabilize the performance of the network on previous tasks $\mathcal{T}^{1:t-1}$, the convolution kernel parameters of each filter f learned up to task $t-1$ (denoted $\Theta_{l,f}^{1:t-1}$) are constrained so that their deviation during learning on task t (i.e., learning $\Theta_{l,f}^{t}$) is penalized. Following the principle of traditional regularization, this is implemented by incorporating a stability regularizer $\mathcal{R}_S(\Theta)$ into the learning objective. Specifically, this regularizer enforces a minimal difference between $\Theta_{l,f}^{1:t-1}$ and $\Theta_{l,f}^{t}$, thereby preserving previously acquired knowledge.

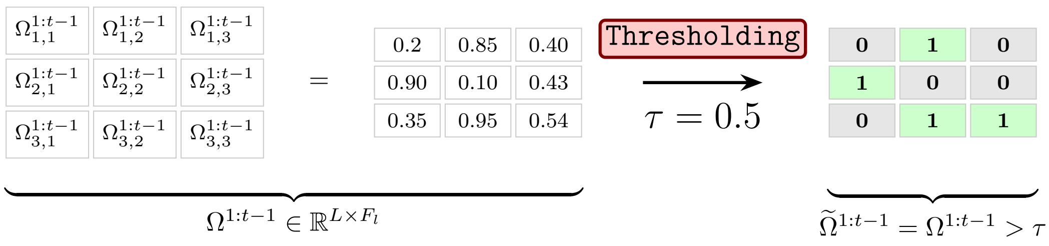
As a result, the optimization problem in Equation (3.1), when training on task t after learning tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$, can be rewritten as:

$$\begin{aligned}\Theta^{*,1:t} &= \arg\min_{\Theta^{1:t}} \; \mathcal{L}_{\text{CL}}^{t}(\Theta^{1:t}, \mathcal{D}_{train}^{t}) \\ &= \arg\min_{\Theta^{1:t}} \; \underbrace{\frac{1}{m^t}\sum_{i=1}^{m^t} \ell_i(f(x_i^t, \Theta^t), y_i^t)}_{\mathcal{L}_{\text{task}}^{t}(\Theta^t)} + \lambda_s \underbrace{\sum_{l=1}^{L}\sum_{f=1}^{F_l} \|\Theta_{l,f}^{t} - \Theta_{l,f}^{1:t-1}\|_2}_{\mathcal{L}_{\text{reg}}^{t}(\Theta^{1:t-1},\Theta^t)=\mathcal{R}_S^t(\Theta^{1:t-1},\Theta^t)} \quad ,\end{aligned} \tag{3.9}$$

where, $\mathcal{L}_{\text{CL}}^{t}$ is the continual learner loss during training at $\mathcal{T}^t$, $\mathcal{L}_{\text{task}}^{t}$ is the task specific loss as defined in Equation (3.1), $\mathcal{L}_{\text{reg}}^{t}$ is the regularization loss also called stability regularizer and λ_s sets the influence of the stability penalty.

The stability regularizer $\mathcal{R}_S^t(\Theta^t)$ in Equation (3.9) penalizes the deviation on all filters with the same hardness, treating them as equally important. However, such an assumption does not hold in modern CNNs architectures, which, as discussed above, tend to be overparameterized (see Meyes et al. [42]), resulting in certain filters playing a more critical role in task-specific learning than others.

To address this shortcoming, the filter importance metric $\Omega^{1:t-1}$ (containing the importance of each filter in learning tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$) described in Section 3.2.3 is used. It abandons the uniform importance approach by assigning each filter a weight proportional to its role in encapsulating knowledge from previous tasks.



$\Omega^{1:t-1}_{1,1}$ $\Omega^{1:t-1}_{1,2}$ $\Omega^{1:t-1}_{1,3}$
$\Omega^{1:t-1}_{2,1}$ $\Omega^{1:t-1}_{2,2}$ $\Omega^{1:t-1}_{2,3}$
$\Omega^{1:t-1}_{3,1}$ $\Omega^{1:t-1}_{3,2}$ $\Omega^{1:t-1}_{3,3}$
=
0.2 0.85 0.40
0.90 0.10 0.43
0.35 0.95 0.54
Thresholding
$\tau = 0.5$
0 1 0
1 0 0
0 1 1
$\Omega^{1:t-1} \in \mathbb{R}^{L\times F_l}$
$\widetilde{\Omega}^{1:t-1} = \Omega^{1:t-1} > \tau$

Figure 3.3: Binary mask matrix generation across CNN layers. Each row l corresponds to a layer, and each column f to a filter within that layer. The matrix $\Omega^{1:t-1}$ contains filter-wise importance values, which are thresholded at $\tau = 0.5$ to yield the binary mask $\widetilde{\Omega}^{1:t-1} \in \{0,1\}^{L\times F_l}$. Green entries (1) indicate filters $\Theta_{l,f} \in \Theta^{1:t-1}_{+}$ to retain (preserve), while gray entries (0) indicate filters $\Theta_{l,f} \in \Theta^{1:t-1}_{-}$ that can adapt to learn a new task.

This approach is similar to previous continual learning regularization strategies for mitigating catastrophic forgetting. However, it differs in scope: instead of penalizing changes in individual parameters, the method presented here focuses on the collective deviation of parameter groups, represented by filters within a CNN or by neurons (all their incoming connections) in a fully connected network.

The binarized version of $\Omega^{1:t-1}$ is a binary mask matrix denoted $\widetilde{\Omega}^{1:t-1} \in \{0,1\}^{L\times F_l}$. The binary mask matrix $\widetilde{\Omega}^{1:t-1} \in \{0,1\}^{L\times F_l}$ is constructed by thresholding each filter's importance score in $\Omega^{1:t-1}$ as follows:

$$\widetilde{\Omega}^{1:t-1}_{l,f} = \begin{cases} 1 & \text{if } \Omega^{1:t-1}_{l,f} \geq \tau \\ 0 & \text{otherwise} \end{cases}, \tag{3.10}$$

where τ is a fixed hyperparameter, selected empirically through cross-validation on a small held-out validation set. It reflects the minimum importance value above which a filter is considered critical to retain. This thresholding process is illustrated in Figure 3.3.

The thresholding operation converts the continuous-valued importance matrix $\Omega^{1:t-1}$ into a binary mask $\widetilde{\Omega}^{1:t-1} \in \{0,1\}^{L\times F_l}$, where each entry indicates whether the corresponding filter $\Theta_{l,f}$ was important (1) or not (0) for retaining knowledge from tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$.

Based on the latter, and denoting multiplication with broadcasting by $\odot$, the operation $\widetilde{\Omega}^{1:t-1} \odot \Theta^{1:t-1}$ can be performed, which divides $\Theta^{1:t-1}$ into 2 sets, denoted $\Theta^{1:t-1}_{+}$ and $\Theta^{1:t-1}_{-}$, respectively containing important and unimportant filters for learning tasks up to $\mathcal{T}^{t-1}$.

Integrating $\Omega^{1:t-1}_{l,f} \subseteq \Omega^{1:t-1}$ into the stability constraint, and constraining only the filters $\Theta_{l,f} \in \Theta^{1:t-1}_{+}$ that are important for the previous tasks, the stability regularizer in Equation (3.9) becomes:

$$\mathcal{R}_S^t(\Theta^{1:t-1}, \Theta^t) = \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta^{1:t-1}_{+}}}^{F_l} \Omega_{l,f}^{1:t-1} \, \|\Theta_{l,f}^t - \Theta_{l,f}^{1:t-1}\|_2 \tag{3.11}$$

Consequently, the final training objective for learning a task t without loosing performance on previously learned tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$, can be formulated as follows:

$$\mathcal{L}_{\mathrm{CL}}^t(\Theta^{1:t}, D_{train}^t) = \frac{1}{m^t} \sum_{s=1}^{m^t} \ell_s(f(x_s^t, \Theta^t), y_s^t) + \lambda_s \mathcal{R}_S^t(\Theta^{1:t-1}, \Theta^t) \tag{3.12}$$

Note that after training on task t, the importance vector $\Omega_{l,f}^{1:t}$ is updated as described in Equation (3.6). This update rule introduces a nuanced approach by incorporating a scaling factor ν (where $0 < \nu \leq 1$) applied to the previous importance scores $(\Omega_{l,f}^{1:t-1})$ before adding the importance scores from the current task $(\Omega_{l,f}^{t})$. This is in contrast to other continual learning regularization methods that use basic accumulation strategies, which generally result in simpler methods that aggregate the importance of parameters across sequentially learned tasks without taking into account the decreasing importance of earlier tasks (e.g., it is desired that the direct importance of initial tasks decrease when their knowledge is already integrated into newer tasks, or when newer tasks provide similar or overlapping information).

3.2.5 Addressing Plasticity in Sequential Task Learning

The loss function defined in Equation (3.12) encourages the network to be stable on previously learned tasks. However, its capacity is not yet optimally used to capture upcoming knowledge, especially in cases where the number of tasks is large. Experiments have shown that this loss function gives satisfactory results on small continual learning benchmarks such as SplitCIFAR10 [64], which consists of only 5 tasks. However, for benchmarks such as SplitMiniImageNet [82] or SplitOmniglot [65], which have 20 and 50 tasks respectively, the CNN quickly becomes inefficient at learning future tasks. Therefore, when training on task $\mathcal{T}^t$, the challenge is to make the network plastic enough to learn tasks $\mathcal{T}^{t+1}$ to $\mathcal{T}^T$ while keeping its capacity fixed.

The strategy proposed in this chapter to address this challenge is inspired by model compression techniques in neural networks [83]. This attractive area of research aims to generate a compact neural network by identifying and removing redundant parameters from an over-parameterized network. The methods proposed for this purpose typically start from a neural network that has been trained in a traditional way, without taking into account such future compression. The method developed in this chapter aims to account for compression during training on each task. Furthermore, instead of discarding unimportant or redundant parameters (here filters/neurons) as model compression techniques do, they are reinitialized to be used for learning future tasks.

The main idea developed here to account for compression during training is to empower the loss function with a regularizer, denoted *plasticity regularizer*, which sparsifies the CNN such that the kernel of redundant filters is zeroed-out. The most effective regularizer for sparsifying CNNs is the group sparsity regularizer [84], which deactivates the entire filter at once, thus achieving structured sparsity. Using group sparsity, the plasticity regularizer takes the form:

$$\mathcal{R}^t(\Theta^t) = \sum_{l=1}^{L} \sum_{f=1}^{F_l} \|\Theta^t_{l,f}\|_2. \tag{3.13}$$

Group sparsity regularizers sparsify a CNN by strongly promoting feature sharing between layers. This is highly desirable in lower layers of a convolutional neural network, where features need to be shared and grouped into a most representative geometry. On the other hand, in upper layers, which aim to discriminate between classes, feature discrimination will be a more appropriate perspective. A regularizer that promotes such feature discrimination has been proposed in [85] and can be defined as:

$$\mathcal{R}^t(\Theta^t) = \sum_{l=1}^{L} \sum_{f=1}^{F_l} \|\Theta^t_{l,f}\|_1. \tag{3.14}$$

In a CNN, this results in constraining the convolutional filters to be as different from each other as possible. The filters thus learn disjoint sets of features, which removes redundancies among them. To allow both feature sharing (at lower layers) and feature discrimination (at higher layers), the group sparsity regularizer in Equation (3.13) is combined with the exclusive sparsity regularizer in Equation (3.14).

As a key component, both sparsity regularizers are applied to the set of filters identified as unimportant for previous tasks, i.e., $\Theta_{l,f} \in \Theta_-^{1:t-1}$, while the filters that are important for learning these tasks ($\Theta_+^{1:t-1}$) are constrained by the stability regularizer not to change much.
The resulting plasticity regularizer can then be formulated as:

$$\mathcal{R}^t_P(\Theta^t) = \underbrace{\sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta_-^{1:t-1}}}^{F_l} \psi_l \|\Theta^t_{l,f}\|_2}_{\text{Group sparsity}} + \underbrace{\sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta_-^{1:t-1}}}^{F_l} \zeta \|\Theta^t_{l,f}\|_1}_{\text{Exclusive sparsity}}, \tag{3.15}$$

with $\zeta = \frac{(1-\psi_l)}{2}$, where $\psi_l = 1 - \frac{l}{L-1}$ adjusts the degree of sharing and discriminating features at each layer, giving more weight to group sparsity at lower layers, while exclusive sparsity dominates at higher layers.

3.2.6 Combining Stability and Plasticity to Enhance CL Efficiency

Section 3.2.4 discusses the methods formulated in this chapter for ensuring network stability, while Section 3.2.5 outlines a proposed strategy for achieving network plasticity.

Combining stability and plasticity enables to create a CL system that can efficiently learn from a continual stream of data by maintaining a balance between these two aspects. This balance ensures that the model remains adaptable enough to learn from new data (plasticity) while retaining essential knowledge from past data (stability). To achieve this, both stability and plasticity regularizers are incorporated into the training objective.

Therefore, during training on task t, the network ensures both the stability on previous tasks $\mathcal{T}^{t'}, t' \in \{1, \cdots, t-1\}$ and the plasticity for future tasks $\mathcal{T}^{t''}, t'' \in \{t+1, \cdots, T\}$ by combining Equations (3.9), (3.11) and (3.15) to form the final training loss function as follow:

$$\begin{aligned}\mathcal{L}^t_{\text{CL}}(\Theta^{1:t}, \mathcal{D}^t_{\text{train}}) &= \mathcal{L}^t_{\text{task}}(\Theta^t) + \lambda_s \mathcal{R}^t_S(\Theta^{1:t-1}, \Theta^t) + \lambda_p \mathcal{R}^t_P(\Theta^t) \\ &= \frac{1}{m^t}\sum_{i=1}^{m^t} l_i(f(x^t_i, \Theta^t), y^t_i) + \lambda_s \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta^{1:t-1}_+}}^{F_l} \Omega^{1:t-1}_{l,f} \, \|\Theta^t_{l,f} - \Theta^{1:t-1}_{l,f}\|_2 \\ &+ \lambda_p \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta^{1:t-1}_-}}^{F_l} \psi_l \|\Theta^t_{l,f}\|_2 + \lambda_p \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta^t_{l,f} \in \Theta^{1:t-1}_-}}^{F_l} \frac{(1-\psi_l)}{2} \, \|\Theta^t_{l,f}\|_1 \,,\end{aligned} \tag{3.16}$$

where λ_s and λ_p are the dimensionless strengths of the stability and plasticity regularizers, $\Omega_{l,f} \in [0,1]$ indicates the importance of filter f in layer l and $\psi_l = 1 - \frac{l}{L-1}$ modulates the balance between sharing and discriminating features in each layer, emphasizing group sparsity in the lower layers and prioritizing exclusive sparsity in the upper layers.

3.2.7 Solving the Resulting Optimization Problem

This section presents the approach to solving the optimization problem arising from the loss function derived in Equation (3.16). This loss function, combines a smooth, differentiable component $\mathcal{L}^t_{\text{task}}(\Theta^t)$ and a non-smooth component $\mathcal{R}^t_S(\Theta^{1:t-1}, \Theta^t) + \mathcal{R}^t_P(\Theta^t)$. Due to this complexity, traditional gradient descent methods are inadequate.

In fact, gradient descent works well for optimizing smooth functions since it relies on the gradient (derivative) to find the direction of the steepest descent. However, it struggle with non-smooth functions due to: (i) Non-Differentiability: the gradient is not defined at all points for non-smooth functions. For example, $g(x) = |x|$ is not differentiable at $x = 0$. (ii) Inefficiency: even where subgradients exist, using standard gradient descent can lead to inefficiencies and poor convergence properties.

To handle the non-smooth part of the objective function in Equation (3.16), a more sophisticated approach, known as proximal gradient descent is required.

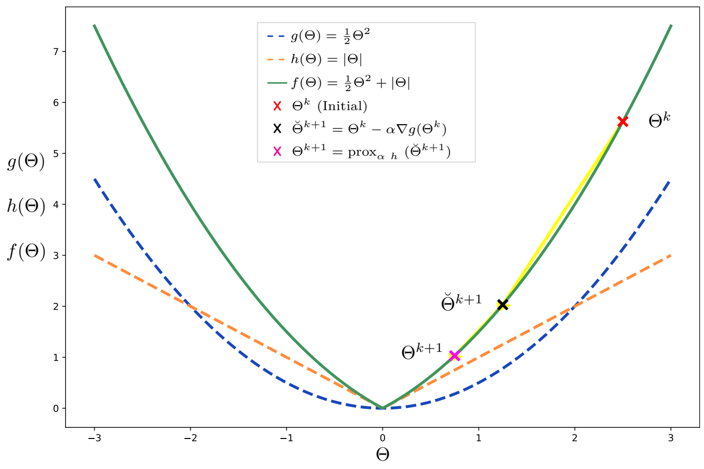


Figure 3.4: Visualization of proximal gradient descent optimization with a learning rate (α) of 0.5. The objective function $f(\Theta) = \frac{1}{2}\Theta^2 + |\Theta|$ consists of a smooth component $g(\Theta) = \frac{1}{2}\Theta^2$ (dashed blue line) and a non-smooth component $h(\Theta) = |\Theta|$ (dashed orange line). The red, black, and magenta "x" markers respectively denote the initial parameter $\Theta^{(k)}$, the intermediate value $\breve{\Theta}^{(k+1)} = \Theta^{(k)} - \alpha\nabla f(\Theta^{(k)})$ obtained after applying the gradient descent step on the smooth component $g(\Theta)$, and the updated value $\Theta^{(k+1)}$ obtained after applying the proximal operator to handle the non-smooth component $g(\Theta)$. The yellow arrows indicate the directions of the gradient descent and proximal steps.

3.2.7.1 Proximal Gradient Descent

To minimize the regularized learning objective function defined in Equation (3.16), a proximal gradient descent (PGD) approach is used. Proximal gradient descent combines the strengths of gradient descent for smooth functions with proximal operators to handle non-smooth regularization, making it suitable for a wide range of optimization problems in machine learning and signal processing. In others words, PGD is particularly effective for optimization problems where the objective function consists of both smooth and non-smooth components. It leverages the smooth part for efficient gradient-based updates, while the proximal operator manages the non-smooth part to ensure effective convergence.

Specifically, PGD is suitable for objectives of the form:

$$\min_{\Theta} g(\Theta) + h(\Theta), \tag{3.17}$$

where $g(\Theta)$ is a convex, differentiable function, and $h(\Theta)$ is potentially non-smooth [86].

The idea is to first take for each epoch $k \in \{0, \cdots, K-1\}$ a gradient step on $g(\Theta)$ followed by a "corrective" proximal gradient step to satisfy $h(\Theta)$:

$$\Theta^{k+1} := \mathrm{prox}_{\alpha\ h}\left(\Theta^k - \alpha\nabla g(\Theta^k)\right), \tag{3.18}$$

where Θ^k is the k^{th} proximal update step and $\mathrm{prox}_{\alpha\ h}$ is the proximal operator defined for a function h scaled by a scalar $\alpha > 0$ as:

$$\text{prox}_{\alpha\ h}(v) = \arg\min_{\Theta} \left(h(\Theta) + \frac{1}{2\alpha} \|\Theta - v\|_2^2 \right). \tag{3.19}$$

The proximal operator essentially "projects" the point found after the gradient step back onto the feasible region defined by the non-smooth term, ensuring that the update respects the structure imposed by $g(x)$.

This allows the proximal gradient descent algorithm to iteratively update the solution by taking into account both the smooth and non-smooth parts of the objective. This process is described in Figure 3.4. Let us rewrite the training objective in Equation (3.16) as:

$$\mathcal{L}_{\text{CL}}^t(\Theta^{1:t}, \mathcal{D}_{\text{train}}^t) = \mathcal{L}_{\text{task}}^t(\Theta^t) + \mathcal{L}_{\text{reg}}^t(\Theta^{1:t-1}, \Theta^t), \tag{3.20}$$

where $\mathcal{L}_{\text{task}}^t(\Theta^t)$ is the smooth, convex ordinary task-specific loss on D_{train}^t (first term in Equation (3.16)) and $\mathcal{L}_{\text{reg}}^t(\Theta^{1:t-1}, \Theta^t)$ is the non-smooth, regularization loss that combines the stability and plasticity regularizers. The proximal gradient iteration as defined in Equation (3.18) thus results in minimizing the task specific loss $\mathcal{L}_{\text{Task}}^t(\Theta^t)$ for only one epoch, with learning rate α, and from the resulting solution applying the proximal operator of the regularization loss $\mathcal{L}_{\text{reg}}^t(\Theta^{1:t-1}, \Theta^t)$.

Assuming training on the task index t, which is omitted in the following for the sake of readability, the gradient descent and proximal gradient steps in Equation (3.18) can then be rewritten for the final training objective in Equation (3.16) as:

$$\breve{\Theta}^{k+1} := \Theta^k - \alpha \nabla \mathcal{L}_{\text{task}}(\Theta^k) \tag{3.21}$$

$$\Theta^{k+1} := \text{prox}_{\alpha\ \mathcal{L}_{\text{reg}}}\ (\breve{\Theta}^{k+1}). \tag{3.22}$$

Since the convolution filters are non-overlapping, the proximal gradient update in Equation (3.22) can be applied independently for each filter in each layer. Before deriving the update rule for each filter, let introduce some lemmas that serve as a fundamental basis for deriving the update rule for each individual filter.

Lemma 3.1.
The proximal operator for a scaled L_2 norm function, denoted by $f(x) = a\|x\|_2$, is expressed as:

$$\text{Prox}_{\alpha f}(v) = \left(1 - \frac{\alpha a}{\|v\|_2}\right)_+ v \tag{3.23}$$

where $\text{Prox}_f(v)$ is the proximal operator applied to the vector v and $\|v\|_2$ is the L_2 norm of v. The term $\left(1 - \frac{\alpha a}{\|v\|_2}\right)_+ = \max\left(1 - \frac{\alpha a}{\|v\|_2}, 0\right)$ ensures that the coefficient remains non-negative by applying a threshold at zero, thereby potentially reducing the magnitude of v without changing its direction.

Proof:

For f defined as $f(z) = a\|x\|_2$ the proximal operator translates to:

$$\text{Prox}_{\alpha f}(v) = \arg\min_x \left\{ \frac{1}{2}\|x - v\|_2^2 + \alpha a\|x\|_2 \right\}$$

This is an unconstrained optimization problem that can be solved by setting the gradient with respect to x to zero:

$$\nabla_x \left(\frac{1}{2}\|x - v\|_2^2 + \alpha a\|x\|_2 \right) = 0$$

The gradient (with respect to x) of the first term, $\frac{1}{2}\|x - v\|_2^2$, is $x - v$, and the gradient of the second term, $\alpha a\|x\|_2$ is $\alpha a \frac{x}{\|x\|_2}$. This translates the above equation to:

$$x - v + \alpha a \frac{x}{\|x\|_2} = 0$$
$$\left(1 + \alpha a \frac{1}{\|x\|_2}\right) x = v$$

Solving for $\|x\|_2$ gives:

$$\|x\|_2 = \frac{\|v\|_2}{1 + \alpha a \frac{1}{\|x\|_2}}$$
$$\|x\|_2 + \alpha a = \|v\|_2$$
$$\|x\|_2 = \|v\|_2 - \alpha a$$

To ensure that $\|x\|_2$ is not negative, the positive part function is applied. This leads to a case distinction in the proximal operator's solution. If $\|v\|_2 > \alpha a$, then $\|x\|_2$ is positive and we can proceed with the solution:

$$x = \frac{v}{1 + \alpha a \frac{1}{\|v\|_2 - \alpha a}}$$
$$x = \left(1 - \frac{\alpha a}{\|v\|_2}\right) v$$

For cases where $\|v\|_2 \leq \alpha a$, minimization results in $x = 0$, as further reduction in $\|x\|_2$ does not minimize the function.

Hence, the analytic solution for the proximal operator of the scaled L_2 norm is:

$$\text{Prox}_{\alpha a\|\cdot\|_2}(v) = \left(1 - \frac{\alpha a}{\|v\|_2}\right)_+ v$$

Lemma 3.2.
For a scaled L_2 norm of the vector x shifted by $\bar{x}$, given by $f(x) = a\|x - \bar{x}\|_2$ with $a > 0$, the proximal operator simplifies to a weighted average between the vector v and $\bar{x}$:

$$\text{prox}_{\alpha f}(v) = \bar{x} + \left(1 - \frac{\alpha a}{\|v - \bar{x}\|_2}\right)_+ (v - \bar{x}) \ , \tag{3.24}$$

where $(x)_+ = \max\{0, x\}$.

Proof

The proximal operator for $f(x) = a\|x - \bar{x}\|_2$ at a point v with parameter α is defined as:

$$\text{Prox}_{\alpha f}(v) = \arg\min_x \left\{ \frac{1}{2}\|x - v\|_2^2 + \alpha a\|x - \bar{x}\|_2 \right\}$$

By introducing $y = x - \bar{x}$, the optimization problem transforms into:

$$\arg\min_y \left\{ \frac{1}{2}\|y + \bar{x} - v\|_2^2 + \alpha a\|y\|_2 \right\}$$

Simplifying the quadratic term gives:

$$\arg\min_y \left\{ \frac{1}{2}\|y - (v - \bar{x})\|_2^2 + \alpha a\|y\|_2 \right\}$$

Now the problem looks like the standard proximal operator for the L_2 norm, but with v replaced by $v - \bar{x}$. We know from the proximal operator for the L_2 norm that:

$$\text{Prox}_{\alpha a\|\cdot\|_2}(v - \bar{x}) = \left(1 - \frac{\alpha a}{\|v - \bar{x}\|_2}\right)_+ (v - \bar{x})$$

Substituting y back with $x - \bar{x}$ gives the solution:

$$\text{Prox}_{\alpha f}(v) = \bar{x} + \left(1 - \frac{\alpha a}{\|v - \bar{x}\|_2}\right)_+ (v - \bar{x})$$

This is the analytic solution for the proximal operator for the shifted and scaled L_2 norm. The solution effectively shifts v towards $\bar{x}$ by the amount determined by αa and the L_2 norm of $v - \bar{x}$, then adds $\bar{x}$ back to re-center the solution.

Lemma 3.3.
For a scaled L_1 norm $g(\Theta) = a\|\Theta\|_1$ with $a > 0$,

$$\text{prox}_{\alpha g}(v) = \Theta^* = \{\Theta_1^*, \Theta_2^*, \cdots, \Theta_n^*\}, \tag{3.25}$$

where each component Θ_i^* is defined as $\text{sign}(v_i)\max\{|v_i| - \alpha a, 0\}$, for v in $\mathbb{R}^n$ and $\alpha > 0$.

Proof

The proximal operator for a scaled L_1 norm, $f(x) = a\|x\|_1$, at a point v with parameter α is defined as:

$$\text{Prox}_{\alpha f}(v) = \arg\min_x \left\{ \frac{1}{2}\|x - v\|_2^2 + \alpha a\|x\|_1 \right\}$$

Unlike the L_2 norm, the L_1 norm introduces complexity in direct differentiation due to its non-differentiability at zero. Nonetheless, the proximal operator for the L_1 norm is well-known and corresponds to the soft-thresholding operator due to the separability of the L_1 norm.

Defined for any scalar v_i and the scalar parameter αa, the soft-thresholding operator $S_{\alpha a}$ is defined as:

$$S_{\alpha a}(v_i) = \text{sign}(v_i)\max(\|v_i\| - \alpha a, 0)$$

For a vector v, the soft-thresholding operator is applied to each component:

$$[S_{\alpha a}(v)]_i = \text{sign}(v_i)\max(\|v_i\| - \alpha a, 0)$$

Thus, the proximal operator's analytical solution for the scaled L_1 norm is:

$$\text{Prox}_{\alpha a\|.\|_1}(v) = S_{\alpha a}(v),$$

where $S_{\alpha a}(v)$ applies the soft-thresholding operator to each component of v. This results in each component of v being shrunk towards zero by αa, with any component whose magnitude is less than αa being set to zero.

Lemma 3.4.
For a function $h(x) = af(x) + bg(x)$, where f and g are convex functions and a and b are positive scalars, the proximal operator of h at a point v can be expressed as:

$$\text{Prox}_{\alpha h}(v) = \text{Prox}_{af}\left(\text{Prox}_{bg}(v)\right) \tag{3.26}$$

Proof:

See Appendix D of [87].

3.2.7.2 PGD Iteration for a Single Filter

Using the insights from the preceding lemmas, the proximal gradient update can now be derived for each filter across each layers.

Considering the training of a single convolutional filter f in layer l at task index t, which is omitted in the following for the sake of readability, the regularization loss term $\mathcal{L}_{\text{reg}}(\Theta)$ in Equation (3.20), which consists of the last 3 terms of Equation (3.16), can be written as:

$$\mathcal{L}_{\text{reg}}(\Theta_{l,f}) = \begin{cases} \lambda_s \Omega_{l,f}^{1:t-1} \, \|\Theta_{l,f} - \Theta_{l,f}^{1:t-1}\|_2 & \text{if } \Theta_{l,f} \in \Theta_{+}^{1:t-1} \\ \\ \lambda_p \left(\zeta_l \|\Theta_{l,f}\|_1 + \psi_l \|\Theta_{l,f}\|_2\right) & \text{if } \Theta_{l,f} \in \Theta_{-}^{1:t-1} \end{cases} \tag{3.27}$$

Applying Lemma 3.2 for the upper part of Equation (3.27) and Lemmas 3.1, 3.3, and 3.4 for the lower part translates the parameter update rules in Equation (3.22), when considering a single filter, into the following two distinct cases:

- If $\Theta_{l,f} \in \Theta_{+}^{1:t-1}$

$$\begin{aligned} \Theta_{l,f}^{k+1} &:= \text{prox}_{\alpha \, \mathcal{L}_{\text{reg}}} \, (\breve{\Theta}_{l,f}^{k+1}) \\ &= \Theta_{l,f}^{1:t-1} + \left(1 - \frac{\alpha \lambda_s \Omega_{l,f}^{1:t-1}}{\|\breve{\Theta}_{l,f}^{k+1} - \Theta_{l,f}^{1:t-1}\|_2}\right)_+ \left(\breve{\Theta}_{l,f}^{k+1} - \Theta_{l,f}^{1:t-1}\right) \\ &= \Theta_{l,f}^{1:t-1} + \beta(\breve{\Theta}_{l,f}^{k+1} - \Theta_{l,f}^{1:t-1}) \\ &= \Theta_{l,f}^{1:t-1} + \beta \, \breve{\Theta}_{l,f}^{k+1} - \beta \, \Theta_{l,f}^{1:t-1} \\ &= \Theta_{l,f}^{1:t-1} \, (1 - \beta) + \beta \, \breve{\Theta}_{l,f}^{k+1} \\ &\text{with} \quad \beta = \left(1 - \frac{\alpha \lambda_s \Omega_{l,f}^{1:t-1}}{\|\breve{\Theta}_{l,f}^{k+1} - \Theta_{l,f}^{1:t-1}\|_2}\right)_+ \end{aligned} \tag{3.28}$$

- If $\Theta_{l,f} \in \Theta_{-}^{1:t-1}$

$$\begin{aligned} \Theta_{l,f}^{k+1} &:= \text{prox}_{\alpha \, \mathcal{L}_{\text{reg}}} \, (\breve{\Theta}_{l,f}^{k+1}) \\ &= \text{prox}_{\alpha \lambda_p \zeta_l \|\breve{\Theta}_{l,f}\|_1} \left(\text{prox}_{\lambda_p \psi_l \|\breve{\Theta}_{l,f}\|_2} \left(\breve{\Theta}_{l,f}^{k+1}\right)\right) \\ &= \text{prox}_{\alpha \lambda_p \zeta_l \|\breve{\Theta}_{l,f}\|_1} \left[\left(1 - \frac{\lambda_p \psi_l}{\|\breve{\Theta}_{l,f}^{k+1}\|_2}\right)_+ \breve{\Theta}_{l,f}^{k+1}\right] \\ &= \{\text{sign}(u_i) \max\{|u_i| - \alpha \lambda_p \zeta_l, 0\}\}_{i=1}^{n} \\ &\text{with} \quad u_i = \left(1 - \frac{\lambda_p \psi_l}{\|\breve{\Theta}_{l,f}^{k+1}\|_2}\right)_+ \breve{\Theta}_{l,f}^{k+1} \end{aligned} \tag{3.29}$$

From Equation (3.28) it can be observed how full stability on the filter (i.e. $\Theta_{l,f}^{k+1} = \Theta_{l,f}^{1:t-1}$) is guaranteed if $\beta = 0$, whereas full plasticity (i.e. $\Theta_{l,f}^{k+1} = 0$) is achieved in Equation (3.29) if $u_i = 0$.

The process for updating the parameters of each filter in every layer can be summarized by the following pseudocode:

Algorithm 1: Filter Optimization Algorithm at task t

```
Input: learned parameters $\Theta^{1:t-1}$ up to task $t-1$,
       learning rate $\alpha$,
       regularizer strength $\lambda_s, \lambda_p$
Result: Updated parameters $\Theta^{k+1}$
1 for each epoch $k$ do
2     $\breve{\Theta}^{k+1} = \Theta^k - \alpha \nabla \mathcal{L}^t_{\text{task}}(\Theta^k)$    ▷ Update parameters using task specific loss;
3     for each filter $f$ in layer $l$, $\Theta^{k+1}_{l,f}$ do
4         Compute $\psi_l = 1 - \frac{f}{L-1}$    ▷ Adjust degree of sharing and discriminating
5                                                   features as given in Equation (3.15);
6         Update $\Theta^{k+1}_{l,f}$ using Equation (3.28) or Equation (3.29);
7     end
8     Process unimportant filter as given in Section 3.2.7.3
9 end
```

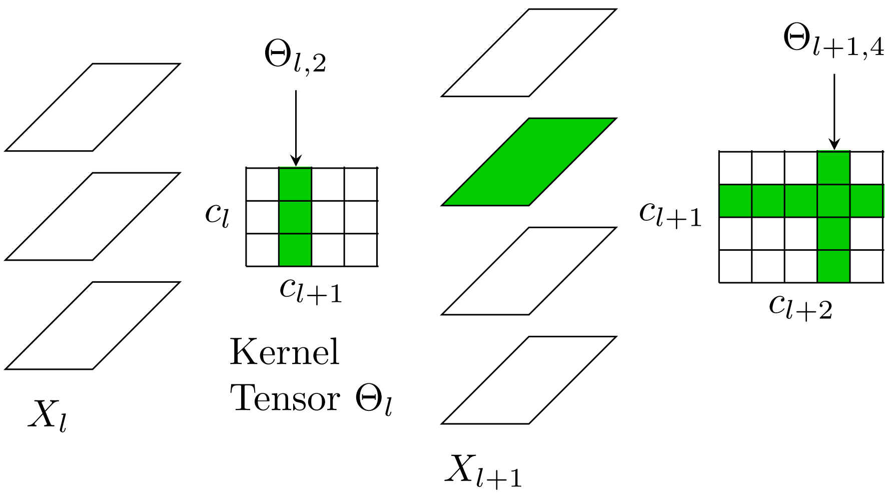


Figure 3.5: Processing unimportant filters: Shown are two layers of a CNN with three and four input feature maps respectively, illustrated as X_l and X_{l+1}. Each of these layers contains four and five filters respectively. The second filter in layer l is detected as unimportant in learning tasks up to t. As a result, the weights of this filter's kernel (denoted as $\Theta_{l,2}$) are reset, and the corresponding feature map kernel (c_{l+1} in X_{l+1}) is zeroed. The same procedure is applied to the filter four in layer $l+1$ (denoted $\Theta_{l+1,4}$), which is similarly identified as unimportant and processed by zeroing its weights and corresponding feature map kernel (c_{l+2} in X_{l+2}).

3.2.7.3 Processing Unimportant Filters

After learning each task $\mathcal{T}^t$ for filters $\Theta^{1:t}_{l,f} \in \Theta^{1:t}_{-}$, i.e. filters that have been identified as unimportant for learning tasks up to t, the following two actions are performed, as illustrated in Figure 3.5:

- Kernel of $\Theta_{l,f}^{t} \in \Theta_{-}^{t}$ (which were detected as unimportant in learning task $\mathcal{T}^{1:t}$) are randomly reinitialized. This allows them to be reactivated and ready to be used by the training procedure for learning a new task. In other words, the learning process for this kernel is "restarted" and can encapsulate new knowledge when new tasks arrive.
- The outputs (feature maps) of unimportant filters (in layer l) are zeroed in the following layer (layer $l+1$). This nullifies any contribution that the unimportant filters from the previous layer would have had on the network's computations. By setting the entire channel or feature map to zero, it becomes sparse. This sparsity is then propagated to the next layer, because the convolution operation with the next set of filters will involve a lot of multiplications by zero, which will also result in zeros. In addition, when a channel is zeroed, the gradient flow through that channel during backpropagation is also stopped. This means that the weights in the filters that would have been connected to that channel are not updated during training, further enforcing sparsity.

Both of these actions result in a more efficient network that runs faster and uses less computing power during training and inference for each new task, while also preserving some capacity for upcoming tasks.

3.3 Experimental Evaluation

The method developed in this chapter (GESCL) is now validated on several benchmark datasets against relevant continual learning baselines. This section presents the baselines and datasets used, along with a thorough analysis of the experiment results, highlighting GESCl's ability to avoid catastrophic forgetting and its competitiveness in terms of average accuracy.

3.3.1 Baselines

GESCL is evaluated against the state of the art regularization-based continual learning methods. Regularization baselines are suitable baselines because, similar to GESCL, they do not rely on model expansion or rehearsal. The following baselines are considered (see 2.9.2 for more details): Elastic Weight Consolidation (EWC) [43], Learning Without Forgetting (LWF) [46], Memory Aware Synapse (MAS) [44], Hard Attenttion to the task (HAT) [73], PackNet [45] and CL via neural pruning (CLNP) [47].

3.3.2 Datasets

The performance of the method presented in this chapter (GESCL) is evaluated against the above baselines on the standard continual learning benchmarks listed below. These datasets were selected for their diversity, complexity, and simulation of real-world scenarios. The selection process involved a thorough review of recent and influential papers in the field to identify commonly used datasets. Although a list of datasets used in this thesis was already presented in Chapter 2, this section details how each dataset in the present experiments is divided into tasks.

- `Split CIFAR-100 [64]:` This benchmark uses the CIFAR-100 dataset, which consists of 100 classes of 32×32 color images. It contains 100 classes, with 600 images per class. These 100 classes are divided into 20 tasks, with each task containing 5 different classes.

- `8-Datasets [88]:` A mixture of 8 different vision datasets: Facescrub [66], CIFAR-10 [64], MNIST [71], SVHN [65], Fashion MNIST [70], CIFAR 100 [64], traffic signs, and notMNIST [89], with each dataset representing a separate task. To standardize the datasets, MNIST, notMNIST, and Fashion MNIST, which are originally 28×28 grayscale (single-channel) images, are transformed into three-channel formats by duplicating the single channel across three channels and zero-padding to convert them to 32×32.

- `Split-MiniImageNet [82]:` A compact version of the Imagenet [90] dataset, consisting of $100k$ images from 200 different classes. It is divided into 20 tasks of 10 classes each.

- `Split Omniglot [65]:` This benchmark consists of thousands of handwritten characters from 50 different alphabets, both real and fictional, each with a different number of classes/characters. Handwritten characters from each alphabet form a separate task. This results in 50 tasks.

Table 3.1 summarizes the main characteristics of the five datasets, including the number of tasks, the size of the input images, the number of classes per task, and the size of the training, validation, and test datasets.

Table 3.1: Summary Statistics Across the Four Datasets

	`8-Datasets`	`Split CIFAR-100`	Split Mini-Imagenet	Split Omniglot
Tasks	8	20	20	50
Input	$3 \times 32 \times 32$	$3 \times 32 \times 32$	$3 \times 84 \times 84$	$1 \times 28 \times 28$
Classes/Task	5 to 100	5	10	14 to 47
Train	286,065	40,000	72,000	24,000
Valid	83,854	10,000	24,000	4,230
Test	83,854	10,000	24,000	4,230

3.3.3 Network Architectures

A scaled-down version of the ResNet18 architecture, similar to that used in [91] and [76], is adopted across all datasets, as it reflects real-world scenarios and is frequently used in the literature, facilitating meaningful comparisons. In this modified architecture, the number of features in each layer is reduced to one-third of the original ResNet18. In addition, the average pooling operation before the classifier layer is configured to be 2×2. Furthermore, the dimensions of the last linear layer change depending on the input height and width.

3.3.4 Training Details

In all experiments, the batch size and number of epochs are set to 128 and 100 epochs, respectively, to match the experimental protocols of the baseline studies. Additionally, an early stopping strategy is implemented in all experiments to optimize training efficiency. The augmentation pipeline used in all experiments includes random cropping, random horizontal flipping, and a color jitter of 0.4.

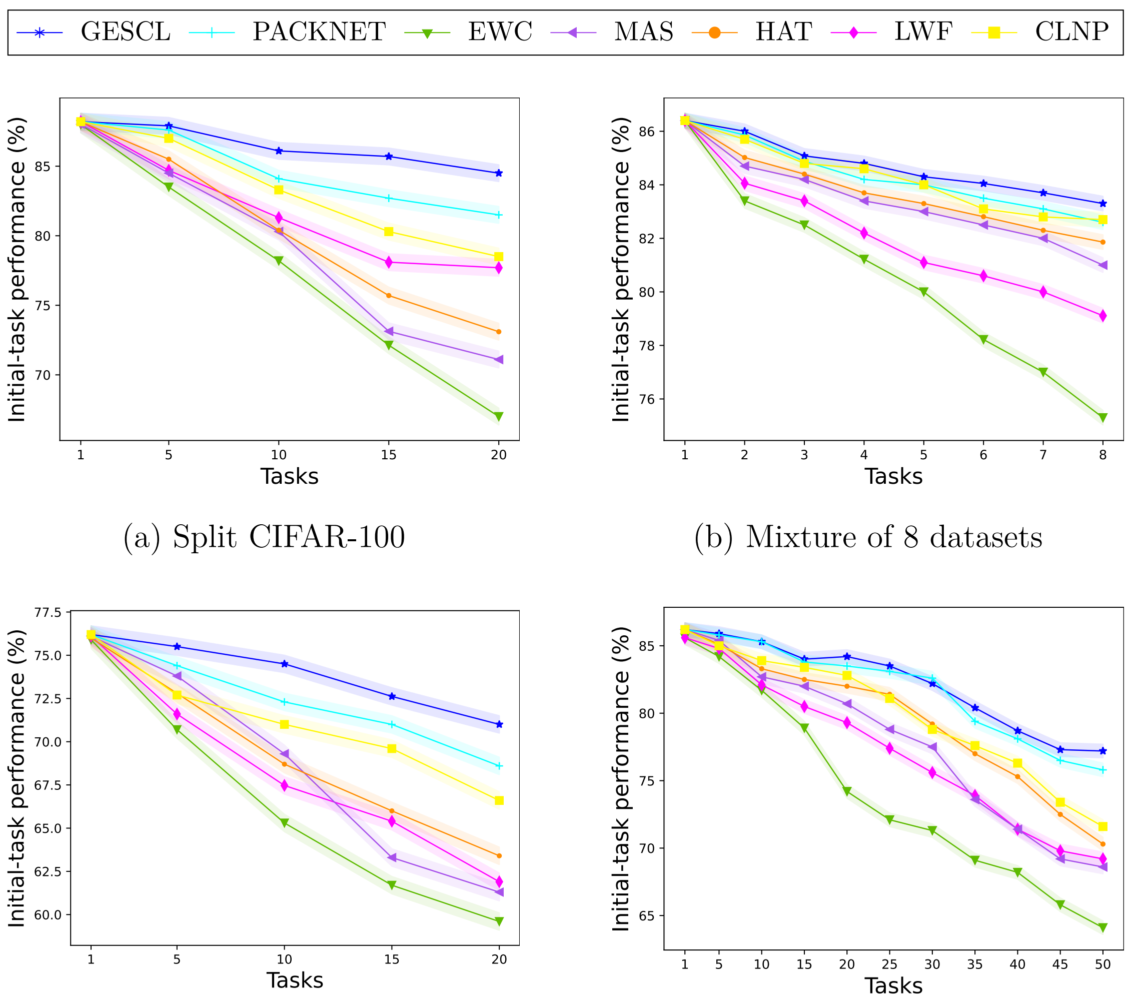
The experiments are run across five different seed values to ensure robustness and variability in the results. All experiments were conducted on a single-GPU configuration using an NVIDIA GeForce GTX 1080 Ti.
For each baseline, optimal hyperparameters were selected via a grid search performed on the validation set, with selections tailored to each dataset to ensure the chosen configuration achieved the highest average accuracy across various settings. It was found that for GESCL, the same hyperparameter configuration worked well across all settings and datasets.

3.3.5 Experimental Results

The primary objectives of the experiments described here are to evaluate: (i) the overall effectiveness of the introduced GESCL methods, as indicated by the average classification accuracy (A) across all tasks, and (ii) the degree to which GESCL mitigates catastrophic forgetting. These two experiments are conducted on the datasets referenced in Section 3.3.2 and compare the performance of GESCL with six state-of-the-art CL continual learning algorithms detailed in Section 3.3.1.
In the experiments, the typical continual learning scenario as presented in Section 3.1 is followed. To recall, in a CL scenario, a model is required to learn a sequence of tasks with unknown data distribution. The assumption is made that after training on task t, D^t_{train} and D^t_{valid} become inaccessible whereas D^t_{test} is used to evaluate the model performance on $\mathcal{T}^t$ after training on $\mathcal{T}^{t'}$, with $t' > t$. Furthermore, it is assumed that the model capacity is fixed.

3.3.5.1 Overcoming Catastrophic Forgetting

As a first experiment, the ability of GESCL to address catastrophic forgetting is explored. To assess catastrophic forgetting, a common approach is to evaluate how the accuracy of each task, or a specific task, changes as additional tasks are learned [92]. In the present experiment, the focus is on observing the first task. This is shown in Figure 3.6, which illustrates how the accuracy on the initial task evolves for each dataset during the continual learning experiment. Specifically, it shows how the accuracy of the initial task evolves after sequentially learning $7, 19, 19$, and 49 tasks in mixture datasets, split cifar-100, split MiniImageNet, and Split Omniglot, respectively. As can be seen (Figure 3.6), after sequential training on all tasks, GESCL is the most stable and less forgetful, showing little forgetting on the ability to perform the first task.



(a) Split CIFAR-100 (b) Mixture of 8 datasets (c) Split MiniImageNet (d) split Omniglot

Figure 3.6: Evaluation of catastrophic forgetting by measuring performance retention on the Initial task during the continual learning experiments. Results show for each dataset how the classification accuracy of the first task evolves as further tasks are being learnt. Overall, GESCL show the strongest resilience against catastrophic forgetting. When facing new tasks, its performance on the initial task does not degrade as much as in concurrent approaches.

Another common measure of forgetting is the *average forgetting rate*, as described in Chapter 2. To recap, it measures the rate at which a continual learning system forgets previously learned information when exposed to new data. The lower the average forgetting rate, the better the model is at retaining old knowledge while acquiring new information.

Table 3.2 shows the average forgetting rate for GESCL and all baselines across the four benchmarks.

Table 3.2: Average forgetting rate. The results are averaged over 5 runs.

	CIFAR-100	8-Dataset	Omniglot	MiniImagenet	CLAD-C
EWC [43]	23.12	24.22	34.02	16.02	17.4
LWF [46]	13.3	13.67	19.28	27.11	11.1
MAS [44]	14.31	16.02	23.42	26.91	12.62
HAT [73]	9.86	10.30	21.01	25.57	7.91
CLNP [47]	7.40	8.21	15.02	22.51	4.42
PackNet [45]	2.92	3.35	9.22	16.02	2.14
GESCL	**2.43**	**2.77**	**7.04**	**10.31**	**2.09**

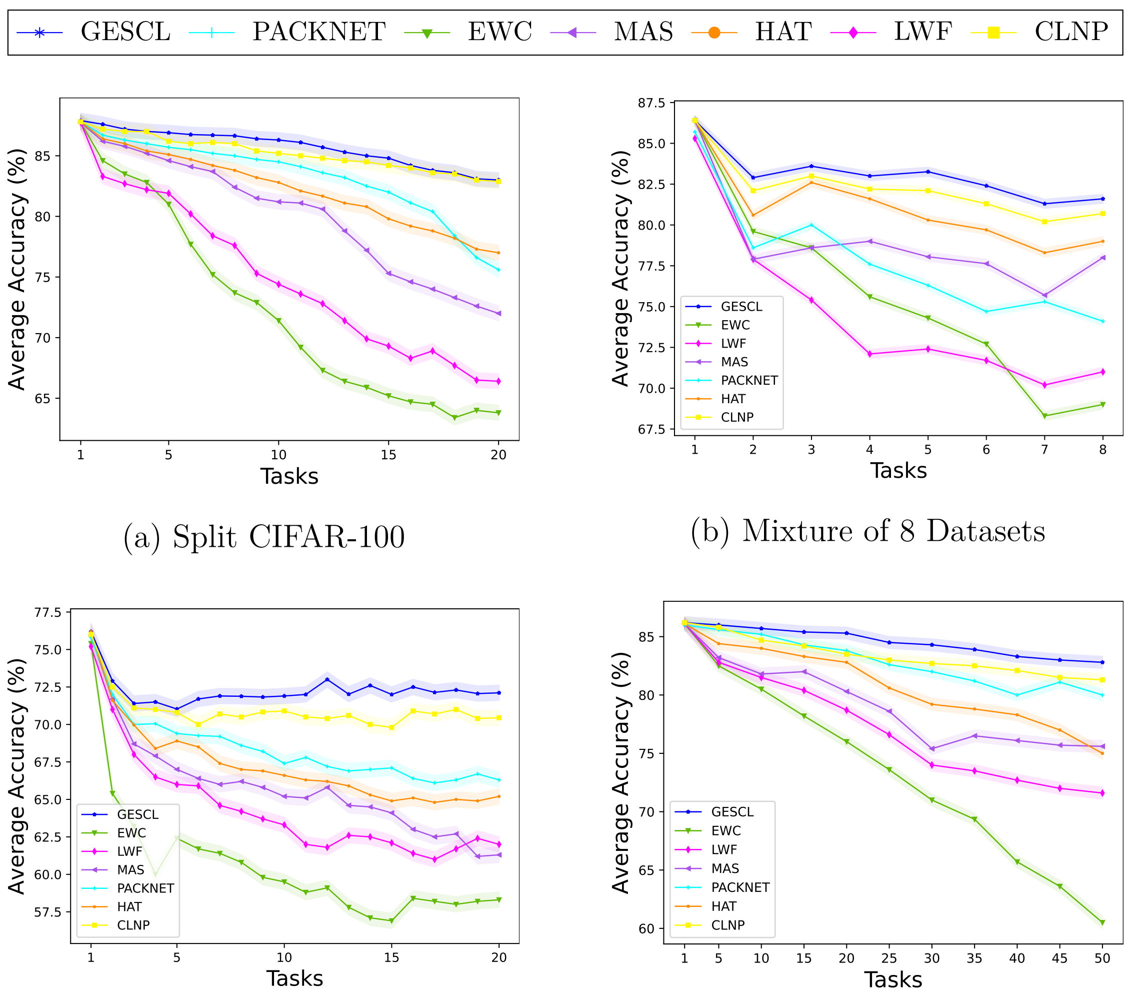
As can be seen (from Table 3.2) GESCL achieves the lowest forgetting rate across all baselines. This highlights the effectiveness of GESCL's approach in retaining learned knowledge, suggesting its potential superiority in addressing the challenges of catastrophic forgetting in various continual learning scenarios.

Overall, thanks to the filter importance adaptive stability regularizer (introduced in Section 3.2.4), which shrinks the change of vital parameters from task to task, GESCL outperforms state-of-the-art baselines accuracy retention degrees. PackNet [45] tends to be a strong competitor especially on 8-dataset and Omniglot, also achieving high performance retention degree, wheras EWC [43] in contrast, suffer from severe degradation of performance especially in Omniglot and Imagenet. This highlights EWC's limitation when the number of tasks increases significantly, likely due to the absence of a plasticity measure in the networks. Other competitors to GESCL, such as CLNP and HAT, incorporate pruning techniques but come with significant computational overhead and retraining, as described in Chapter 2.

3.3.5.2 Overall Classification Accuracy

This experiment aims to validate the effectiveness of GESCL in terms of classification accuracy during the continual learning experiments. To evaluate the classification accuracy, the all-important average accuracy metric [76] described in Chapter 2 is used.

Figure 3.7 provides for different datasets, a view of how the average accuracy evolves from task to task during the CL experiments. The results indicate that GESCL outperforms baseline algorithms by a significant margin. Interestingly, CLNP [47] reaches almost the same performance as GESCL on the last task on Split CIFAR-100, despite being significantly outperformed by the latter in earlier tasks. Another interesting observation is that CLNP [47] is the strongest competitor regarding average accuracy while it is surpassed by PackNet [45] on all datasets in terms of avoiding forgetting as shown in Figure 3.6. In both average accuracy and forgetting, GESCL demonstrates superior performance compared to the aforementioned CL reference baselines.



(a) Split CIFAR-100

(b) Mixture of 8 Datasets

(c) Split MiniImageNet

(d) split Omniglot

Figure 3.7: Evaluation of average accuracy on the test sets across four datasets during the continual learning experiments. It can be observed that GESCL significantly outperforms the reference baselines in terms of overall performance when learning diverse tasks sequentially.

An empirical conclusion that can be made out of Figures 3.6 and 3.7 is that GESCL achieves strong overall continual learning results, thanks to the way it addresses catastrophic forgetting and learning of several new tasks via the stability- and plasticity regularizers pair embedded in the networks learning procedure.

3.3.5.3 More Comparison with Expansion-Based Methods

The baselines used so far are all regularization-based CL methods. This section compares GESCL with expansion-based continual learning strategies to explore different approaches and highlight how GESCL performs against expansion-based methods.

Specifically, it compares GESCL with "Calibrating CNNs for Lifelong Learning" (CCLL) [93] and "Efficient Continual Learning with Modular Networks and Task-Driven Priors" (ECLMN) [94]. These methods were chosen for their ease of implementation and demonstrated superiority over many expansion-based approaches. A comparison of GESCL, CCLL, and ECLMN was performed on the Split CIFAR100 and Split MiniImageNet datasets. The results are shown in Table 3.3. The hyperparameters for these methods were adjusted to ensure that their model capacities were at least equivalent to that of GESCL.

Table 3.3: Comparing GESCL with expansion-based methods on Split CIFAR100 dataset and split Mini-Imagenet. The comparison metrics are accuracy (A) and network capacity, both expressed as percentages.

Methods	Split CIFAR100		Split Mini-Imagenet	
	A (%)	Capacity (%)	A (%)	Capacity (%)
ECLMN [94]	81.7	109	71.2	134
CCLL [93]	78.4	104	69.7	132
GESCL	**83.5**	**100**	**72.2**	**100**

Table 3.3 shows that GESCL, even without network expansion, still outperforms these two expansion-based continual learning baselines on the aforementioned datasets. Specifically, GESCL achieves the highest accuracy on both Split CIFAR100 and Split Mini-Imagenet, with scores of 83.5% and 72.2% respectively, indicating that it outperforms the expansion baselines in terms of learning effectiveness. For Split CIFAR100, GESCL outperforms ECLMN by 1.8 percentage points and CCLL by 5.1 percentage points in accuracy, while requiring 9 percentage points less network capacity than ECLMN and 4 percentage points less than CCLL. On the Split Mini-Imagenet dataset, GESCL's accuracy shows a modest improvement over ECLMN by just 1 percentage point, but it significantly outperforms CCLL by 2.5 percentage points. Additionally, GESCL operates with a network capacity that is substantially lower than both ECLMN and CCLL, by 34 percentage points and 32 percentage points respectively. These capacity values indicate that GESCL is more efficient, maintaining a baseline capacity of 100%, while ECLMN and CCLL require more capacity to achieve lower accuracies. In summary, these results highlight GESCL's superior accuracy on both datasets while using a more compact or optimized network capacity.

3.4 Ablation Study and Analysis

This section provides a comprehensive ablation study of the method developed in this chapter (GESCL) along with a detailed analysis of its training and inference times compared to some baselines.

3.4.1 Ablation Study

An ablation study was performed to examine the contribution of each component of GESCL in its overall performance on each datasets. Particularly, the focus was on understanding how (i) the stability regularizer λ_s, (ii) the plasticity regularizer λ_p, (iii) the filter importance balance ν, contribute to the base model. To inspect this, variants of GESCL with different combinations of those components, each component being either enabled (✓) or disabled (×), were implemented. The results are reported in Table 3.4, where A denotes the average classification accuracy and F indicates the average forgetting rate, respectively described in Sections 3.3.5.2 and 3.3.5.1. Results for those two metrics as shown in Table 3.4 demonstrate the effectiveness of each component at improving the classification accuracy and overall forgetting of GESCL.

Table 3.4: Ablation study evaluating the impact of different components on the average accuracy (A) and average forgetting (F) of GESCL variants on Split CIFAR 100, 8-datasets, Split MiniImageNet, and Omniglot after learning the last task $\mathcal{T}^T$. The concerned components are the plasticity regularizer (λ_p), the stability regularizer (λ_s), and the filter importance balance (ν), each denoted by ✓ if included and × if not.

λ_s	λ_p	ν	Split Cifar 100		8-Datasets		Split MiniImageNet		Split Omniglot	
			A(%)	F	A(%)	F	A(%)	F	A(%)	F
✓	✓	✓	**83.5**	2.43	**81.6**	2.77	**72.2**	7.04	**82.8**	10.31
✓	✓	×	78.2	2.41	75.1	2.69	69.2	6.88	78.6	10.1
×	✓	✓	43.9	57.7	40.3	54.2	34.7	65.2	52.9	68.3
✓	×	✓	68.9	0.11	64.7	0.24	55.2	0.405	58.4	0.485
×	×	✓	26.3	78.3	24.8	77.8	23.6	81.3	29.5	84.2
✓	×	×	29.2	**0.08**	27.6	**0.13**	25.8	**0.22**	31.8	**0.27**
×	✓	×	28.2	68.2	27.93	73.1	22.9	77.4	23.8	79.4

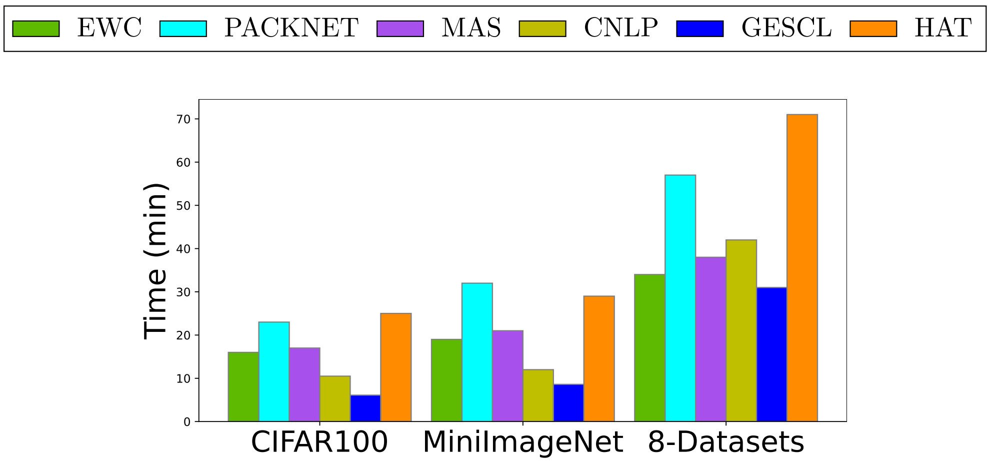
The table shows that the filter importance balance (ν) has the least impact on accuracy, indicating that the network retains sufficient stability and plasticity even in its absence. Omitting this component treats the relevance of filters as equally important before and after learning a particular task. When the filter importance balance (ν) is removed: 5.3 percentage points for CIFAR100, 6.5 percentage points for 8 datasets, 3.0 percentage points for MiniImageNet, and 4.2 percentage points for Omniglot. Moreover, excluding (in addition to the importance balance filter (ν)) also the plasticity regularizer (λ_p) leads to the most significant reduction in forgetting. In such a configuration, the network prioritizes performance retention on previously learned tasks at the expense of adaptability to new tasks, as indicated by the significant decrease in accuracy.

Furthermore, it can be seen that removing only the stability regularizer (λ_s) leads to a significant increase in forgetting, particularly noticeable in Split CIFAR 100 and Split MiniImageNet, and also negatively affects the average accuracy. This decrease in average accuracy is due to the significantly degraded performance on earlier tasks, which biases the overall accuracy. On the other hand, removing only the plasticity regularizer (λ_p) tends to reduce accuracy, but also reduces forgetting in most cases. This suggests that while the plasticity regularizer contributes to better learning and accuracy, it may also lead to more significant forgetting.

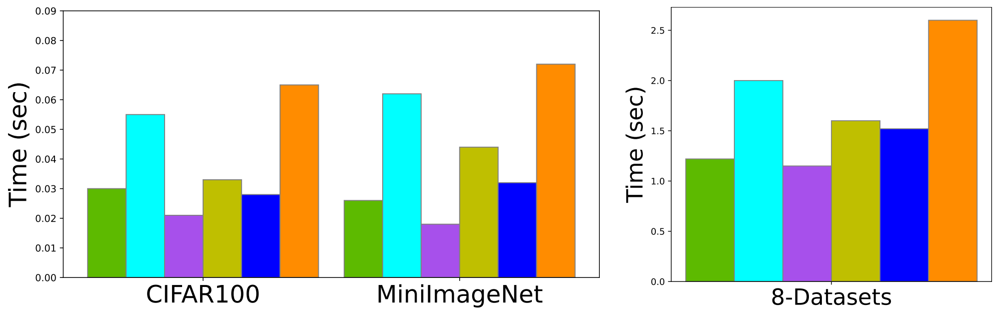
Removing both the plasticity (λ_p) and stability regularizer (λ_s) leads to very low accuracy and extremely high forgetting rates across all datasets, particularly on Split MiniImageNet and Omniglot, which emphasizes the critical role these regularizers play in memory retention. The absence of both λ_s and ν results in a significant drop in accuracy, with a notable increase in forgetting for most datasets. This indicates that while λ_s is essential for memory retention, ν seems to contribute more towards maintaining accuracy.

All of the above results indicate that the combination of all three components is generally the most effective for achieving a balance between high accuracy and low forgetting.

3.4.2 Training and Inference Time



(a) Training Time



(b) Inference Time

Figure 3.8: comparison of the training time (in minutes) and inference time (in seconds) of the GESCL method against various baseline models across three different datasets: CIFAR100, MiniImageNet, and 8-Datasets.

Figure 3.8 (a) shows the amount of time each method (GESCL and baselines) takes to train on CIFAR100, MiniImageNet, and 8 datasets. The baselines shown are EWC [43], PACKNET [45], MAS [44], CLNP [47], and HAT [73] . It can be seen that GESCL consistently achieves the shortest training times compared to the other methods on all three datasets, indicating a highly efficient training process that requires less time to achieve its learning goals. Specifically, on the CIFAR100 dataset, GESCL completes training in approximately 6 minutes. Even with the increased complexity of the MiniImageNet dataset, GESCL continues to outperform other methods in terms of training speed. Similarly, for the 8-datasets, which are a mixture of different datasets, GESCL again requires the least amount of training time among the baselines. These observations confirm that GESCL is adept at handling diverse datasets while maintaining an advantageous training time efficiency relative to other methods.

Regarding inference time, Figure 3.8 (b) illustrates the average time each method takes to make predictions after training. Inference time is critical because models are often expected to provide fast and accurate responses, especially in real-world scenarios. For the CIFAR100 dataset, while GESCL's inference time is not the fastest, it remains competitive, striking a balance between speed and accuracy. This suggests a well-rounded approach that emphasizes both training efficiency and operational responsiveness. For the MiniImageNet dataset, although GESCL inference time is not the fastest, it remains within an acceptable range, ensuring practical usability. A similar pattern emerges for the 8 datasets, where GESCL's inference time is significantly faster than many competitors. This consistent performance across different datasets highlights GESCL's ability to maintain a competitive speed of inference while ensuring robust model performance.

In summary, GESCL may be slightly slower than some baselines in terms of inference time, but this slight delay is offset by its superior accuracy and robustness, as demonstrated in the previous section. In addition, GESCL outperforms all baselines in training time and outperforms several in inference time. This underscores GESCL's high efficiency and effectiveness in both the learning and operational phases, demonstrating its adaptability and resource management capabilities.

3.4.3 GESCL Limitations

The methods described in this chapter (GESCL) operate under the constraint of a fixed neural network capacity. Although a sparsity regularizer has been developed to increase the plasticity of the network so that it can learn as many tasks as possible, the fixed capacity constraint inevitably leads to a bottleneck when the system is overloaded with tasks beyond its capacity limit. This saturation point indicates that the network cannot handle new tasks without sacrificing performance or accuracy on previously learned tasks. In other words, a fixed capacity constraint limits the number of tasks that the continual learning system can handle sequentially, but also compromises the system's ability to maintain performance and accuracy on previously learned tasks. This reveals a fundamental weakness in scalability and adaptability, and suggests an urgent need for a more flexible learning architectures.

To address this limitation, the next chapter introduces an innovative approach that integrates GESCL with a dynamic network expansion strategy. This adaptive solution methodically increases the network's capacity in response to incoming tasks, ensuring sustained scalability and enhanced adaptability. By adopting such a strategy, the system is not only relieved from the constraints of fixed capacity but is also better equipped to handle a growing number of tasks without degrading its efficiency or effectiveness. This approach represents a significant step forward in continual learning, providing a more robust and resilient learning systems.

3.5 Chapter Summary

A new approach dubbed "GESCL" for continual learning of fixed capacity neural network has been presented. The proposed algorithm derived a novel regularization term to deal with the stability-plasticity dilemma. This regularizationn term consist of two major element: a stability and a plasticity term. The stability term avoid forgetting of previous tasks by preventing filters/neurons detected as important for past tasks to deviate too much when learning a new task. The sparsity term leverages the over-parameterization of neural networks to efficiently sparsify the network and tunes unimportant filters making them relevant for future tasks. A key ingredient of the newly derived regularization term is a new method for computing the importance of each filter/neuron in learning a given task. Experiments on popular CL vision benchmarks show that GESCL leads to significant improvements over state-of-the-art method in terms of overall continual learning performance, as measured by classification accuracy as well as in terms of avoiding catastrophic forgetting.

Chapter 4

A New Hybrid Regularization- and Expansion-based CL Method

In this chapter, the problem of solving continual learning when the number of tasks is relatively large compared to the network's capacity is described. Furthermore, a novel approach that combines regularization and expansion-based strategy to address this problem is proposed. This approach, termed "Sparsification and Expansion-based Continual Learning (SECL)", is then evaluated against expansion-based continual learning benchmarks described in related work (Chapter 2).

Parts of this chapter has been published in [2], [3] and [4].

4.1 Problem Statement

Chapter 3 introduced a novel regularization-based method (called GESCL) to address the challenges of continual learning in fixed-capacity neural networks. To maximize the number of learned tasks, a sparsity mechanism was implemented. This mechanism promotes sparsity during training, ensuring that only a minimal capacity of the network is used for each task. As a result, the network can learn a relatively large number of tasks, depending on its initial capacity, as shown in the experimental part of Chapter 3 (Section 3.3.5).

However, despite this sparsity technique, if the number of tasks to be learned is very large or greatly exceeds the capacity of the network, the regularization-based method cannot handle tasks arriving beyond the saturation point of the network without erasing previously acquired knowledge, leading to catastrophic forgetting.

To illustrate this, Figure 4.1 shows the distribution of filter importance for the method developed in the previous chapter (GESCL) when dealing with a fixed-capacity network. This figure is based on experiments using CIFAR-10/100 with a simple convolutional neural network consisting of 6 blocks of 3×3 convolutions with 32, 64, 128 filters respectively, each followed by ReLU activation and a 2×2 max-pooling layer. Specifically, the figure displays the number of filters deemed important after learning each task $\mathcal{T}^t$.

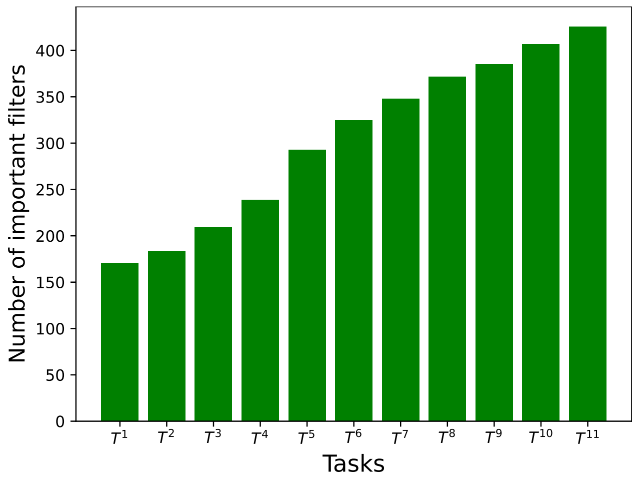


Figure 4.1: Distribution of filter importance for CIFAR-10/100 using GESCL Methods Across Tasks $\mathcal{T}^1$ to $\mathcal{T}^{11}$, where CIFAR-10 is the initial task and Cifar100 is divided into 10 tasks to form the subsequent tasks. The visual representation shows an increasing trend in the number of important filters as the network processes more tasks, highlighting the necessity for adaptive network capacity.

The figure clearly demonstrates that as more tasks are learned, an increasing number of filters are utilized and marked as important. Initially, 34% of filters are required to learn the first task $\mathcal{T}^1$. If the next task, $\mathcal{T}^2$, is very similar to the previous one, the number of additional filters needed to learn $\mathcal{T}^2$ will be relatively small. In contrast, if $\mathcal{T}^2$ is much more complex or dissimilar from $\mathcal{T}^1$, more filters will be required to capture the distinct features of $\mathcal{T}^2$. In both cases, as more tasks are learned, the number of important filters increases, nearing the total capacity of 448 filters. If further tasks are introduced, the network might eventually reach its maximum capacity. At this point, the continual learner network will be unable to learn new tasks without erasing previous acquired knowledge. To address this, an expansion strategy that effectively manages capacity limitations while preserving previously acquired knowledge is necessary.

A major challenge in expansion-based continual learning approaches is deciding when to expand the network. When learning task $\mathcal{T}^t$, if the previously learned parameters $\Theta^{1:t-1}$ is able to effectively describe the new task, then network expansion, i.e., increasing the parameter space, may not be necessary. On the other hand, if the previously learned features cannot effectively describe the new task, then introducing additional capacity (filters) becomes essential to accommodate the features needed to learn the new task. This raises a critical question: when a new task $\mathcal{T}^t$ arrives, how can one determine whether the existing capacity is adequate for learning this task? In view of this, there is a need for a robust heuristic to detect when the available capacity is insufficient to effectively assimilate a given task, thereby triggering network expansion.

Two others challenge are determining where to expand (requiring strategic identification of critical layers), and how much capacity to add (necessitating precise resource allocation). To address these three challenges, a new approach, Sparsification and Expansion-based

Continual Learning (SECL), combining regularization and an expansion-based strategy coupled with several heuristics has been developed. In particular, the contributions of the approach are the following:

- SECL, similar to GESCL, addresses catastrophic forgetting (CF) through a regularization method that disables gradient updates on crucial filters, preventing changes to critical representations learned from previous tasks.
- To ensure optimal resource use and avoid unnecessary expansion, SECL employs sparsity and rewiring. Unlike conventional dense architectures, it maintains a sparse network structure during each task's learning process, enhancing plasticity without requiring expansion. Sparse filters can then be reactivated for learning future tasks.
- After learning each task, SECL evaluates the model's plasticity by assessing the available network capacity. If needed, it enhances plasticity by expanding the network with additional filters, ensuring optimal performance for new tasks.
- Finally, SECL provides a simple but effective heuristic for determining the optimal timing, locations, and amount of capacity needed for expanding a continual learning network.

4.2 Method: Sparsification and Expansion-based Continual Learning (SECL)

This section provides a detailed description of the SECL method, which integrates regularization and expansion strategies to address the plasticity-stability challenge in continual learning.

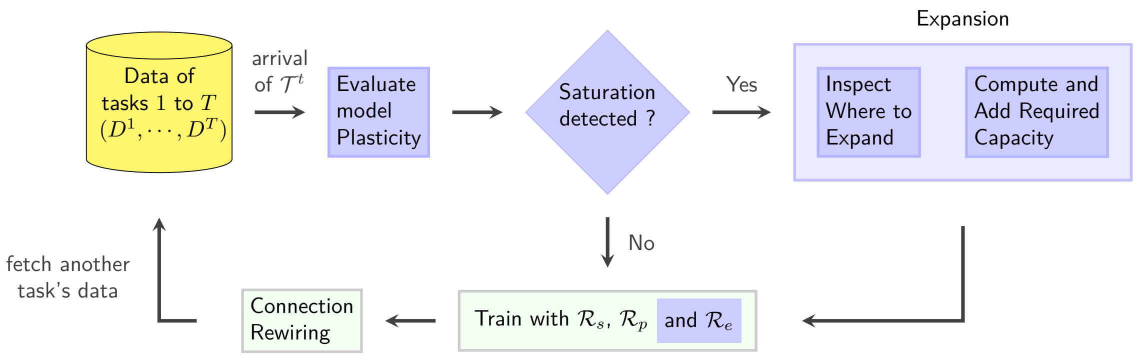

Data of tasks 1 to T $(D^1, \cdots, D^T)$
arrival of $\mathcal{T}^t$
Evaluate model Plasticity
Saturation detected ?
Yes
Expansion
Inspect Where to Expand
Compute and Add Required Capacity
No
Train with $\mathcal{R}_s$, $\mathcal{R}_p$ and $\mathcal{R}_e$
Connection Rewiring
fetch another task's data

Figure 4.2: Flow diagram illustrating the SECL method, which builds upon the GESCL method presented in Chapter 3. The green blocks represent components introduced in Chapter 3 (GESCL), while the blue blocks indicate the new components added in the current chapter (SECL).

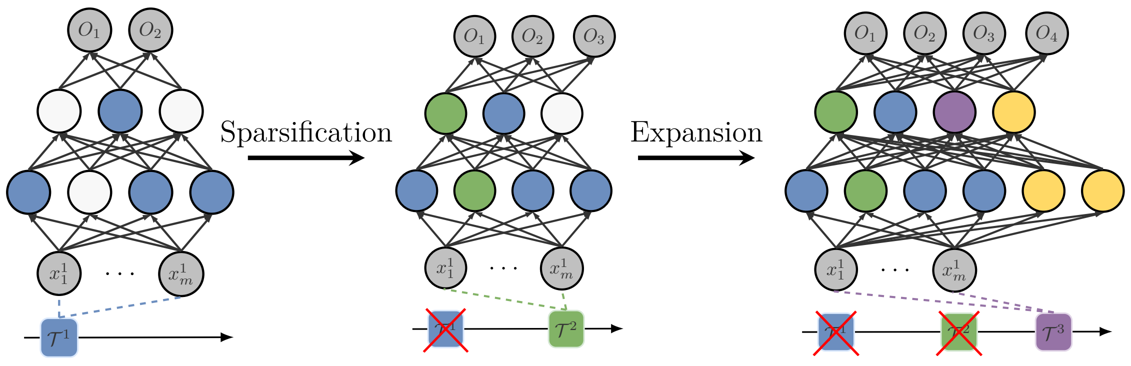


Figure 4.3: **Overview of SECL.** On the left, all neurons are initially unimportant. SECL trains on task $\mathcal{T}^1$ with the sparsity regularizer according to equation (3.15), identifying neurons critical for $\mathcal{T}^1$, shown in blue. In the middle, during learning of task $\mathcal{T}^2$, the network trains with both sparsity and stability regularizers to preserve knowledge from the first task; neurons important for $\mathcal{T}^2$ are highlighted in green. On the right, by the time the task $\mathcal{T}^3$ arrives, almost all neurons have been marked as important for previous tasks, necessitating the addition of new capacity (shown in yellow) to accommodate $\mathcal{T}^3$. At this stage, the network is further trained with the sparsity regularizer to use the additional capacity sparsely. The red “X” marks over task 1 and task 2 in the middle and right panels indicate the absence of data from these tasks at the arrival of these tasks.

4.2.1 Preliminaries

SECL builds upon the method developed in the previous chapter (GESCL). Specifically, to address continual learning challenges with an unlimited number of tasks, SECL initially implements the stability and sparsity/plasticity techniques from GESCL to learn as many tasks as possible. This is achieved through a stability regularizers $\mathcal{R}_s$ and a plasticity regularizers $\mathcal{R}_p$, similar to those in GESCL, which are incorporated into the SECL training objective, as detailed later in this chapter. Additionally, when the network reaches saturation – a condition detected by a newly developed heuristic – SECL activates an expansion mechanism to increase the network’s capacity. This process is illustrated in Figure 4.2.

4.2.2 SECL: Preventing Forgetting

Similar to GESCL, SECL prevents forgetting by ensuring that filters identified as important in learning task up to $\mathcal{T}^{t-1}$, do not change significantly when learning of a new task $\mathcal{T}^t$. This is achieved by enhancing the training objective with a regularization term $\mathcal{R}_s$ known as the stability regularizer as shown in Chapter 3.

4.2.3 SECL: Sparsification/Plasticity

As seen in Chapter 3, even if a network has enough capacity to learn a given task, care must be taken to use it sparsely. Therefore, instead of directly expanding the network as traditional expansion-based CL methods do, SECL employs a sparsification mechanism to use the network’s capacity in a sustainable manner. The sparsification mechanism, is inspired by model pruning and compression techniques (detailed in Section 2.8), which aim to reduce the size of a neural network by removing redundant or less important parameters. These methods typically involve equipping the objective function with a sparsity regularizer acting on the network parameters.

The sparsification process is shown in Figure 4.3, where task 1 is learned with the minimum possible capacity, allowing task 2 to be learned without expansion. Network expansion is then triggered at the arrival of task 3 when the available capacity is saturated.

4.2.4 SECL: Network Expansion

Even with the sparsification scheme presented in the previous section (see Section 3.2.5 for more details), network expansion may eventually become unavoidable. This is especially true in two scenarios: (i) the number of learning tasks is extremely high. For example, in a continual learning setting, an autonomous vehicle may need to constantly learn new tasks throughout its lifespan, such as recognizing new types of road signs, adapting to evolving traffic patterns or identifying new obstacles. In such a scenario, saturation and therefore expansion is inevitable. (ii) There is significant variability in tasks. For instance, the features needed to recognize handwritten digits differ significantly from those required to identify objects in images, such as distinguishing various animals in a wildlife dataset. In such cases, the lower-level filters may need to learn entirely different types of features. If these layers have no free filters remaining, additional filters must be added to accommodate the new tasks.

However, as mentioned in Section 4.1, extending a network in a continual learning setting presents specific challenges categorized into three main groups. The remainder of this section discusses each of these challenges in detail.

4.2.4.1 SECL Expansion: Scalability

Expanding network capacity inevitably leads to increased training and storage costs. Many expansion-based continual learning methods freeze the parameters related to old tasks Θ^{old} to maintain performance on those tasks, while adapting only the newly added parameters Θ^{new} to learn the new task. This results in a model architecture whose complexity is proportional to the number of tasks, yielding a highly redundant structure.

To mitigate this significant drawback, SECL applies the sparsity/plasticity-inducing regularizer presented in Section 4.2.3 to the parameters Θ^{new} of filters added during network expansion. This is shown in Figure 4.4. For example, if n additional filters are added to learn a new task $\mathcal{T}^t$, such as recognizing different dog breeds, the sparsification mechanism will ensure that only the smallest necessary subset of these filters is used.

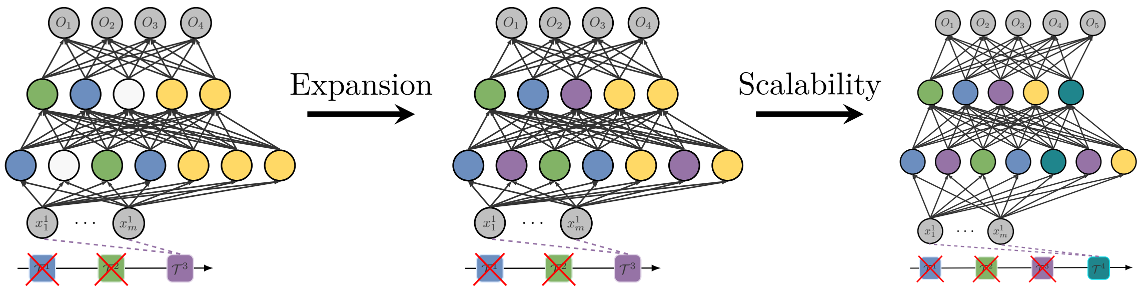


Figure 4.4: Scalability though Sparsity on Θ_{new} (Θ_{new} is shown in yellow). In the left panel, the network is already trained on tasks $\mathcal{T}^1$ and $\mathcal{T}^2$. When task 3 arrives, the available capacity of the network is insufficient, so the network expansion module is triggered. The network is then expanded with yellow neurons. In the middle panel, after training on task 3, the critical neurons for this task are identified (shown in purple). Due to the stability regularizer applied to the newly added capacity Θ_{new}, only a minimal capacity of Θ_{new} is used. The right panel shows the arrival of the task $\mathcal{T}^4$, where the remaining capacity of Θ_{new} is used to learn this task.

If the next task $\mathcal{T}^{t+1}$ is very similar to t, such as recognizing the same dog breeds in different lighting conditions (domain incremental learning), it is possible that a subset of the remaining filters, or in the worst case, the entire set of remaining filters, will be sufficient to learn this new task. This approach prevents the need for adding another set of filters to handle task $\mathcal{T}^{t+1}$, thereby maintaining a more efficient and less redundant network architecture.

This not only optimize overall performance and resource utilization but also significantly enhances the scalability of SECL, allowing it to handle an increasing number of tasks without a proportional increase in complexity or resource consumption, which is crucially needed in an expansion-based continual learning system.

4.2.4.2 SECL Expansion: Heuristicity

Continual learning expansion-based methods typically guide the timing of network expansion by setting specific thresholds on metrics like loss or accuracy [54, 50]. The network is then expanded when these metrics exceed or fall below predefined threshold values. For example, after learning task $\mathcal{T}^t$, if the test accuracy of task $\mathcal{T}^t$ is lower than a threshold value τ (or the loss is higher than τ), it is assumed that there is not enough capacity, which is accompanied by underfitting, necessitating network expansion. While this approach is straightforward, it assumes that the threshold τ suitable for one task will be applicable to others, which is not always the case. One could also set a different threshold for each task. However, this necessitates prior knowledge of the difficulty of each task, which is often unavailable and impractical to estimate in advance.

In SECL, a more adaptive and less assumption-dependent heuristic is developed to guide network expansion. SECL bases its expansion heuristic on a dynamic component of the objective function, specifically the filters' importance vector $\Omega^{1:t-1}$ (see 3.2.3.3). This vector quantifies the significance of each filter in the network in learning tasks 1 up to

$t-1$. By focusing on an intrinsic property of the network's learning process, SECL simplifies the expansion mechanism and reduces the overhead associated with manually setting and adjusting threshold values for different tasks.

This innovative approach allows SECL to adaptively expand the neural network architecture in a manner that is both responsive to the network's immediate needs and efficient in computational terms, leading to an effective continual learning system. This process is outlined in the following three steps:

1. *Importance Thresholding*: After learning task $t-1$, the importance of each filter f in each layer l in learning tasks 1 to $t-1$, denoted as $\Omega_{l,f}^{1:t-1}$, is computed using the filters' post-activation metric, as detailed in Section 3.2.3. This allow to group the network parameters in two categories, respectively, parameters of important filter $\Omega_{l,f}^{1:t-1} \in \Theta_{+}^{1:t-1} \subset \Theta^{1:t-1}$, and parameters of unimportant filter $\Omega_{l,f}^{1:t-1} \in \Theta_{-}^{1:t-1} \subset \Theta^{1:t-1}$, i.e.,
$$\Omega_{l,f}^{1:t-1} \in \begin{cases} \Theta_{+}^{1:t-1}, & \text{if filter } f \text{ is important,} \\ \Theta_{-}^{1:t-1}, & \text{otherwise.} \end{cases} \tag{4.1}$$

2. *Layer-specific Evaluation*: Once the unimportant filters represented by their parameters $\Omega_{l,f}^{1:t-1} \in \Theta_{-}^{1:t-1} \subset \Theta^{1:t-1}$ are identified, their distribution across the network layers is assessed. For each layer l, the number of available (unimportant) filters up to task $t-1$ is defined as:
$$f_{-}^{l,1:t-1} = \sum_{f=1}^{f_{\text{init}}^{l}} \begin{cases} 1, & \text{if } \Omega_{l,f}^{1:t-1} \in \Theta_{-}^{1:t-1} \\ 0, & \text{otherwise} \end{cases}, \tag{4.2}$$
where f_{init}^{l} is the total initial number of filters in layer l.
If, after learning tasks 1 to t—1, the number of available (unimportant) filters $f_{-}^{l,1:t-1}$ falls below a predefined threshold proportion $\tau \in [0,1]$ of the initial capacity f_{init}^{l}, i.e.,
$$f_{-}^{l,1:t-1} < \tau \cdot f_{\text{init}}^{l}, \tag{4.3}$$
it indicates that the layer may be under-capacity for handling new task. In such cases, layer l is marked for expansion.

   To reflect the different capacities and roles of layers in the network, the threshold τ is defined adaptively. Lower values of τ are assigned to shallower layers, which tend to learn more general features and are less prone to saturation. In contrast, deeper layers, which typically encode task-specific features and saturate more quickly, are assigned higher τ values to trigger earlier expansion. Additionally, τ increases progressively as more tasks are learned, ensuring that sufficient capacity is maintained as filter utilization grows over time.
   At this stage, the "when" and "where" questions regarding network expansion are answered by adding additional filters when the number of important filters falls below the specified threshold and adding the filters to the identified under-capacity layers to enhance their ability to manage new tasks effectively.

3. *Adaptive Filter Addition*: Based on this layer-specific evaluation, additional filters f_{add} are introduced into layers where the current capacity is insufficient. The number of added filters $f_{\text{add}}^{l,t} \subset f_{\text{add}}$ in a specific layer l at task t corresponds to the difference between the initial capacity f_{init}^{l} of that filter and its remaining capacity $f_{-}^{l,1:t-1}$ (after training task 1 to t-1). This is formally defined as:

$$\begin{aligned} f_{\text{add}}^{l} &= \left\lceil \beta \cdot \left(\tau \cdot f_{\text{init}}^{l} - f_{-}^{l,1:t-1} \right) \right\rceil \\ &= \left\lceil \beta \cdot \left(\tau \cdot f_{\text{init}}^{l} - (f_{\text{init}}^{l} - f_{+}^{l,1:t-1}) \right) \right\rceil \\ &= \left\lceil \beta \cdot \left(f_{+}^{l,1:t-1} - (1-\tau) \cdot f_{\text{init}}^{l} \right) \right\rceil \end{aligned} \tag{4.4}$$

where τ as defined above is the minimum proportion of free (unimportant) filters to be maintained relative to the initial capacity, and β is an expansion scaling coefficient (e.g., $\beta = 1.0$ for a minimal match, $\beta > 1$ for more aggressive over-expansion), controlling how much additional capacity is added when expansion is triggered.

This process offers a heuristic to guide network expansion by determining “when”, “where” and “how many” additional filters to introduce. The expected result is an improved resource allocation that adapts to the needs of the network as it encounters new tasks and data. By using this method, the network is expected to remain scalable and robust in handling continual learning scenarios, although the specific benefits will be validated in the experimental section.

4.2.4.3 SECL Expansion: Dealing with Additional Parameters

Another crucial aspect of network expansion is the handling of added filters. SECL effectively addresses this issue, as shown in Figure 4.5. While the concept has been implemented in CNNs, the figure uses a dense network for visual clarity, as this architecture facilitates a more straightforward illustration of the underlying idea. The figure shows the addition of neurons to two consecutive layers within the dense network.

In layer l, two neurons (filled in yellow) have been added, while in layer $l+1$, one neuron has been added. Additionally, the figure shows a specialized neuron (dashed in yellow) added to the output layer, dedicated to the current task. This expansion process results in the creation of three distinct parameter matrices, which are central to the network's expanded architecture:

1. $\Theta^{\mathsf{expand,\ green}} = \{\Theta_{l-1,l}^{in}, \Theta_{l,l+1}^{in}\}$: This set of matrices, shown in green connections, encapsulates the incoming connections from the previous layer to each newly added neuron. It defines how the new neurons in layer l will receive and process information from layer $l-1$, and similarly, how the new neurons in layer $l+1$ will connect to the original neurons in layer l.
2. $\Theta^{\mathsf{expand,blue}} = \Theta_{l,l+1}^{in,out}$: This matrix, shown in blue, represents the interconnections between newly added neurons across consecutive layers, allowing direct interaction among these neurons in the expanded network.

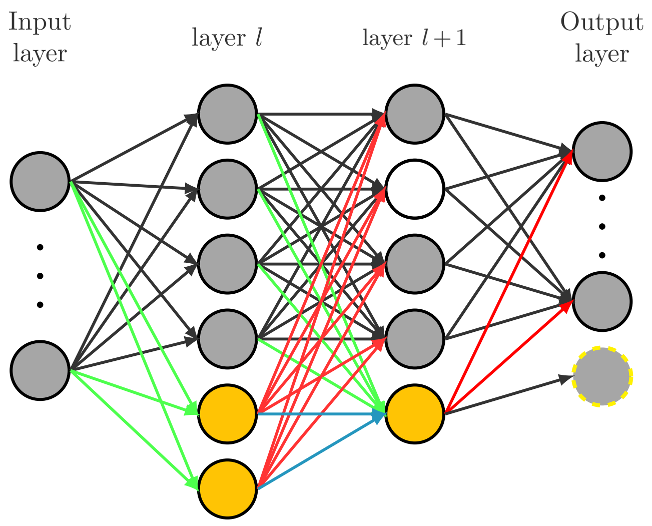


Figure 4.5: Schematic overview of the network expansion module: unimportant neurons are shown in white, important neurons are shown in gray, and newly added neurons are shown in yellow. The dashed yellow neuron represents the output neuron added at the arrival of the task at which the network is expanded.

3. $\Theta^{\textsf{expand},\text{red}} = \{\Theta^{out}_{l,l+1}, \Theta^{out}_{l+1,l+2}\}$: This set of matrices, shown in red connections, captures the outgoing connections from the newly added neurons to the existing neurons in the subsequent layer. It details how the output of the new neurons in layer l will affect layer $l+1$, and how the new neuron in layer $l+1$ influence layer $l+2$.

Each of these three sets of matrices plays a pivotal role in the functionality of the network post-expansion. $\Theta^{\textsf{expand},\text{green}}$ and $\Theta^{\textsf{expand},\text{red}}$ facilitate communication between the new neurons and the rest of the network, ensuring that they can effectively contribute to the network's learning process. $\Theta^{\textsf{expand},\text{blue}}$ is particularly important for enabling complex interactions within the newly expanded layer itself. By clearly distinguishing between these sets of connections, the expansion strategy not only efficiently scales up the network but also maintains an interpretable and organized structure.

This organized structure allows for the added connections/matrices to be handled in an organized way during training. Specifically, $\Theta^{\textsf{expand},\text{green}}$ and $\Theta^{\textsf{expand},\text{blue}}$ can be trained without any restrictions as they do not have a direct influence on previously established knowledge in the network. However, care must be taken when dealing with $\Theta^{\textsf{expand},\text{red}}$. Without proper handling, the information flowing to important neurons in layers $l+1$ and $l+2$ may change during future training, resulting in catastrophic forgetting of previous knowledge.

To prevent this catastrophic forgetting occurrence during training on task t (the task at which the expansion has occur), an additional regularization term, $\mathcal{R}_{\text{expand}}$, is introduced into the objective function:

$$\mathcal{R}_{\text{expand}}(\Theta^t) = \sum_{\Theta_+^{\mathsf{expand},3}} \|\Theta_{l,f}^t\|_2 \tag{4.5}$$

This term strongly constrains the magnitude of $\Theta^{\mathsf{expand},\text{red}}$ specifically on neurons deemed important for learning past tasks $\{\mathcal{T}^1, \cdots, \mathcal{T}^{t-1}\}$, thereby minimizing any potential interference with the retention of previously acquired knowledge. This strategy provides a balanced approach that allows for network expansion while maintaining the integrity of prior learning, resulting in a robust continual learning system that can expand over time without losing its foundational knowledge as will be seen in the experimental section.

4.2.5 Combining Stability, Plasticity and Expansion to Enhance CL Efficiency

Once the network is expanded for task $\mathcal{T}^t$, the new connections/matrices (2D kernels in a CNN) must be included in the training process. Section 4.2.4.3 discuses how SECL addresses this by differentiating between weights that can be trained without restrictions and those that require constraints to avoid harming the overall CL system. Specifically, a regularization term, $\mathcal{R}_{\text{expand}}$, is introduced to the objective function to prevent the newly added connections (particularly those linking new connections to important existing neurons) from causing catastrophic forgetting. Accordingly, the learning objective derived in Chapter 3 (see Section 3.16) to train the GESCL system changes for SECL when training with the network extension towards:

$$\begin{aligned}\mathcal{L}_{\text{CL}}^t(\Theta^{1:t}, \mathcal{D}_{train}^t) = {} & \frac{1}{m^t}\sum_{s=1}^{m^t} l_s(f(x_s^t, \Theta^t), y_s^t) + \lambda_s \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta_{l,f}^t \in \Theta_+^{1:t-1}}}^{F_l} \Omega_{l,f}^{1:t-1} \|\Theta_{l,f}^t - \Theta_{l,f}^{1:t-1}\|_2 \\ & + \lambda_p \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta_{l,f}^t \in \Theta_-^{1:t-1}}}^{F_l} \psi_l \|\Theta_{l,f}^t\|_2 + \lambda_p \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \Theta_{l,f}^t \in \Theta_-^{1:t-1}}}^{F_l} \frac{(1-\psi_l)}{2} \|\Theta_{l,f}^t\|_1 \\ & + \lambda_e \sum_{l=1}^{L} \sum_{\substack{f=1 \\ \forall \Theta_{l,f} \in \Theta_+^{expand,3}}}^{F_l} \|\Theta_{l,f}^t\|_2 \, ,\end{aligned} \tag{4.6}$$

where, $\sum_{l=1}^{L}\sum_{i=1}^{F_l} \|\Theta_{l,f}^t\|_2$ for all $\Theta_{l,f} \in \Theta_+^{\mathsf{expand},3}$ is the expansion regularizer, λ_e is a hyperparameter that regulates the extent of expansion regularization, and the other terms (shown in purple) are known from Chapter 3.

Algorithm 2 presents the overall SECL learning framework:

Algorithm 2: Sparsification and Expansion-based CL (SECL) Algorithm

```
Input: Model parameters $\Theta$;
       Task Dataset $\mathcal{D} = (\mathcal{D}^1, \cdots, \mathcal{D}^T)$;
       Threshold $\tau$
Result: $\Theta^T$    ▷ Model parameters after training on the last task $\mathcal{T}^T$
1  for $t = 1, \cdots, T$ do
2      if $t = 1$ then
3          Train $\Theta^{t=1}$    ▷ train the network parameters using the sparsity regularizer
4                                    only, as defined in Equation (3.15);
5      else
6          Compute $\Omega_{l,f}^{t-1}$    ▷ compute the importance of each filter $f$ in learning tasks
7                                    up to $\mathcal{T}^{t-1}$, as defined in Equation (3.6) ;
8          if number of important filters $< \tau$ then
9              Train $\Theta^{t=1}$ with Equations (3.12) and (3.15)    ▷ Train the network with
10                                   the stability and plasticity
11                                   regularizers
12         else
13             $\Theta^t$ = Expansion $\left(\Theta^t\right)$    ▷ Expand the network as described in
14                                   Section 4.2.4;
15             Train $\Theta^{t=1}$ with Equation (4.6)    ▷ train the network with stability,
16                                   plasticity and expansion regularizers
17         end
18     end
19 end
```

4.2.6 Solving the Resulting Optimization Problem

This section describes the method for solving the optimization problem presented by the final loss function in Equation (4.6).

4.2.6.1 Training with Proximal Gradient Descent (PGD)

To minimize the loss function in Equation (4.6) a proximal gradient descent (PGD) approach is used. Proximal gradient has been introduced in Chapter 3, Section 3.2.7.1.

The optimization problem in Equation (4.2.4) can be reformulated as:

$$\begin{aligned} &\Theta^{*,t} = \arg\min_{\Theta^{*,t}} \mathcal{L}_{\text{CL}}^t(\Theta^{1:t}, \mathcal{D}_{\text{train}}^t) \\ &\text{with} \;\; \mathcal{L}_{\text{CL}}^t(\Theta^{1:t}, \mathcal{D}_{\text{train}}^t) = \mathcal{L}_{\text{task}}^t(\Theta^t) + \mathcal{L}_{\text{reg}}^t(\Theta^{1:t-1}, \Theta^t) \end{aligned} \tag{4.7}$$

where $\mathcal{L}_{\text{task}}^t(\Theta^t)$ – the first term in Equation (4.6) – is the smooth, convex, task-specific loss on $\mathcal{D}_{\text{train}}^t$, and $\mathcal{L}_{\text{reg}}^t(\Theta^{1:t-1}, \Theta^t)$ is the non-smooth regularization loss comprising the remaining four terms of Equation (4.6).

Applying PGD with learning rate α, Equation (4.7) can be iteratively minimized by processing the following two steps, where the first equation is the regular SDG and the second step is a PGD update:

$$\breve{\Theta}^{k+1} := \Theta^{k} - \alpha \nabla \mathcal{L}^{t}_{\text{task}}(\Theta^{k}) \tag{4.8}$$

$$\Theta^{k+1} := \text{prox}_{\alpha\ \mathcal{L}^{t}_{\text{reg}}}\ (\breve{\Theta}^{k+1}). \tag{4.9}$$

Since the convolution filters are non-overlapping, such a proximal gradient update, as shown in Equation (4.9), can be applied independently to each filter in every layer, as discussed in Section 3.2.7.1. The next section will demonstrate how this update rule is derived for each individual filter using Lemmas 3.1, 3.2, 3.3 and 3.4 from Chapter 3.

4.2.6.2 PGD Iteration for a Single Filter

Considering the training of a single convolutional filter f in layer l for task t, where index t is omitted for the sake of readability, the regularization loss term $\mathcal{L}_{\text{reg}}(\Theta_{l,f})$, combining the last four terms in Equation (4.6), is rewritten as:

$$\mathcal{L}_{\text{reg}}(\Theta_{l,f}) = \begin{cases} \lambda_s \Omega^{1:t-1}_{l,f}\ \|\Theta_{l,f} - \Theta^{1:t-1}_{l,f}\|_2 & \text{if } \Theta_{l,f} \in \Theta^{1:t-1}_{+} \\ \lambda_e \|\Theta_{l,f}\|_2 & \text{if } \Theta_{l,f} \in \Theta^{\text{expand},3}_{+} \\ \lambda_p \left(\zeta_l \|\Theta_{l,f}\|_1 + \psi_l \|\Theta_{l,f}\|_2\right) & \text{if } \Theta_{l,f} \in \Theta^{1:t-1}_{-} \end{cases} \tag{4.10}$$

By applying Lemma 3.2 to the upper part and Lemmas 3.1, 3.3, and 3.4 to the lower part of Equation (4.10), the parameter update rules in Equation (4.9) translate into the following three distinct cases:

- If $\Theta_{l,f} \in \Theta^{1:t-1}_{+}$

$$\begin{aligned} \Theta^{k+1}_{l,f} &: = \text{prox}_{\alpha\ \mathcal{L}_{\text{reg}}}\ (\breve{\Theta}^{k+1}_{l,f}) \\ &= \Theta^{1:t-1}_{l,f} + \left(1 - \frac{\alpha \lambda_s \Omega^{1:t-1}_{l,f}}{\|\breve{\Theta}^{k+1}_{l,f} - \Theta^{1:t-1}_{l,f}\|_2}\right)_{+} \left(\breve{\Theta}^{k+1}_{l,f} - \Theta^{1:t-1}_{l,f}\right) \\ &= \Theta^{1:t-1}_{l,f} + \beta(\breve{\Theta}^{k+1}_{l,f} - \Theta^{1:t-1}_{l,f}) \\ &= \Theta^{1:t-1}_{l,f} + \beta\ \breve{\Theta}^{k+1}_{l,f} - \beta\ \Theta^{1:t-1}_{l,f} \\ &= \Theta^{1:t-1}_{l,f}\ (1-\beta) + \beta\ \breve{\Theta}^{k+1}_{l,f} \\ \text{with} \quad \beta &= \left(1 - \frac{\alpha \lambda_s \Omega^{1:t-1}_{l,f}}{\|\breve{\Theta}^{k+1}_{l,f} - \Theta^{1:t-1}_{l,f}\|_2}\right)_{+} \end{aligned} \tag{4.11}$$

- If $\Theta_{l,f} \in \Theta_{+}^{\text{expand},3}$

$$\begin{aligned} \Theta_{l,f}^{k+1} :&= \text{prox}_{\alpha\ \mathcal{L}_{\text{reg}}}\ (\breve{\Theta}_{l,f}^{k+1}) \\ &= \left(\breve{\Theta}_{l,f}^{k+1} - \frac{\alpha \lambda_e \breve{\Theta}_{l,f}^{k+1}}{\|\breve{\Theta}_{l,f}^{k+1}\|_2} \right)_+ \\ &= \left(1 - \frac{\alpha \lambda_e}{\|\breve{\Theta}_{l,f}^{k+1}\|_2} \right)_+ \breve{\Theta}_{l,f}^{k+1} \end{aligned} \tag{4.12}$$

- If $\Theta_{l,f} \in \Theta_{-}^{1:t-1}$

$$\begin{aligned} \Theta_{l,f}^{k+1} :&= \text{prox}_{\alpha\ \mathcal{L}_{\text{reg}}}\ (\breve{\Theta}_{l,f}^{k+1}) \\ &= \text{prox}_{\alpha \lambda_p \zeta_l \|\breve{\Theta}_{l,f}\|_1} \left(\text{prox}_{\lambda_p \psi_l \|\breve{\Theta}_{l,f}\|_2} \left(\breve{\Theta}_{l,f}^{k+1} \right) \right) \\ &= \text{prox}_{\alpha \lambda_p \zeta_l \|\breve{\Theta}_{l,f}\|_1} \left[\left(1 - \frac{\lambda_p \psi_l}{\|\breve{\Theta}_{l,f}^{k+1}\|_2} \right)_+ \breve{\Theta}_{l,f}^{k+1} \right] \\ &= \{\text{sign}(u_i) \max\{|u_i| - \alpha \lambda_p \zeta_l, 0\}\}_{i=1}^{n} \\ \text{with}\ \ u_i &= \left(1 - \frac{\lambda_p \psi_l}{\|\breve{\Theta}_{l,f}^{k+1}\|_2} \right)_+ \breve{\Theta}_{l,f}^{k+1} \end{aligned} \tag{4.13}$$

These Equations (4.11, 4.12, 4.13) provide the final update for each filter in each layer of the convolutional network. From Equation (4.11) it can be observed how full stability (i.e. $\Theta_{l,i}^{k+1} = \Theta_{l,i}^{1:t-1}$) is guaranteed if $\beta = 0$, whereas full plasticity (i.e. $\Theta_{l,i}^{k+1} = 0$) is achieved in Equation (4.13) if $u_i = 0$.

The complete process for updating the parameters of each filter in each layer is summarized by the following pseudocode:

Algorithm 3: Filter Optimization Algorithm in SECL.

```
Input: learned parameters $\Theta^{1:t-1}$ up to $t-1$,
       $\Theta_{+}^{\text{expand},3}$, learning rate $\alpha$, regularizer strength $\lambda_s, \lambda_p, \lambda_e$
Result: Updated parameters $\Theta^{k+1}$
1 for each epoch $k$ do
2     $\breve{\Theta}^{k+1} = \Theta^k - \alpha \nabla \mathcal{L}_{\text{task}}^{t}(\Theta^k)$     ▷ Update parameters using task specific loss;
3     for each filter $f$ in layer $l$, $\Theta_{l,f}^{k+1}$ do
4         Compute $\psi_l = 1 - \frac{f}{L-1}$     ▷ Adjust degree of sharing and discriminating
5                                                  features as given in Equation (3.15);
6         Update $\Theta_{l,f}^{k+1}$ using Equation (4.11), Equation (4.12) or Equation (4.13);
7     end
8     Process unimportant filter as given in Section 3.2.7.3
9 end
```

4.3 Experimental Evaluation

In this section, the SECL method is validated on several benchmark datasets, comparing its performance against relevant continual learning baselines.

4.3.1 Baselines

SECL is evaluated against state-of-the-art regularization (fixed capacity) and expansion-based continual learning methods. Expansion-based approaches are suitable baselines because, like SECL, they rely on expanding the model based on internal dynamics. Regularization baselines, on the other hand, highlight the limitations of fixed-capacity approaches and underscore the need for expansion methods such as SECL.

Specifically, SECL is compared to GESCL (the method presented in Chapter 3) and EWC [43], which are fixed-capacity regularization methods, as well as expansion methods, notably, Compact, Picking and Growing (CPG) [50], Progress and Compress (P&C) [49], Feature Boosting and Compression (FOSTER) [52], and Dynamic Token Expansion (DyTox) [51], chosen for their innovative strategies in balancing model growth and efficiency. These methods are discussed in detail in the Chapter 2.

4.3.2 Datasets

The performance of SECL is evaluated against the aforementioned baselines using the following standard continual learning datasets. These datasets have been described in detail in Chapter 2, Section 2.9.1; here, we outline how they have been divided into tasks for the experiments presented later in this chapter:

- `Permuted MNIST [72]:` Each subsequent task is created by applying a unique permutation to every image in the MNIST dataset. This is an example of domain incremental learning, where tasks share the same set of classes but differ in input domains.
- `Split FaceScrub [66]:` a widely used dataset in face recognition, with the first 100 people having the highest number of occurrences selected to form 100 classes. These classes are then divided into 20 tasks with 5 classes each.
- `Split Cifar100 [64]:` This benchmark uses the Cifar100 dataset, which consists of 100 classes of 32×32 images. It is divided into 20 tasks, with each task containing 5 different classes.
- `Split Omniglot [65]:` It consists of thousands of handwritten characters from 50 different alphabets, both real and fictional, each with a different number of classes/characters. Each class/character is treated as a sequence of different classification problems.
- `ImageNet-32 [68]:` A downsampled (32×32) version of the entire ImageNet [69]. It is split into 200 tasks of 5 classes each. It is used in this chapter to compare methods performance in very long sequence scenario.

Table 4.1 summarizes the main characteristics of the four datasets, including the number of tasks, the size of the input images, the number of classes per task, and the size of the training, validation, and test datasets.

Table 4.1: Summary statistics across the five datasets.

	P-Mnist	FaceScrub	`Cifar100`	Omniglot	`ImageNet 32`
Number of Tasks	10	20	20	50	200
Input Size	28×28	$3 \times 32 \times 32$	$3 \times 32 \times 32$	$1 \times 28 \times 28$	$3 \times 32 \times 32$
Classes/Task	10	5	5	14 to 47	5
Train	40,000	14,000	40,000	24,000	105,065
Valid	10,000	3,000	10,000	4,230	28,000
Test	10,000	3,000	10,000	4,230	28,000

4.3.3 Training Details

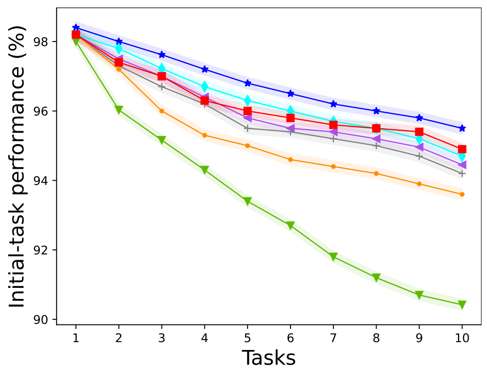
In all experiments, the batch size and number of epochs are set to 128 and 100, respectively. Additionally, an early stopping strategy is implemented to optimize training efficiency. The experiments are run across five different seed values to ensure robustness and variability in the results. All experiments were conducted on a single-GPU configuration using an NVIDIA GeForce GTX 1080 Ti.

For each baseline, optimal hyperparameters were selected via a grid search performed on the validation set, tailored to each dataset to achieve the highest average accuracy across various settings. It was found that for SECL, the same hyperparameter configuration worked well across all settings and datasets, except for the expansion regularizer λ_e, which has more weight in longer tasks scenarios.

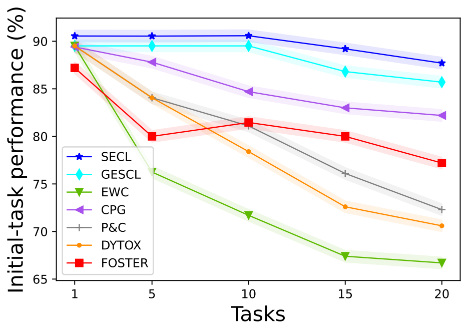
4.3.4 Overcoming Catastrophic Forgetting

As a first experiment, the ability of SECL to address catastrophic forgetting is investigated. A common measure to evaluate forgetting, introduced in Chapter 3, is to assess how the accuracy of the initial task changes as subsequent tasks are learned [92]. This is visualized in Figure 4.6, which shows the accuracy progression of the initial task across different data sets during the continual learning experiment. Specifically, it shows how the accuracy for the initial task shifts after sequentially learning 9, 19, 19, 49, and 199 tasks across the permuted Mnist datasets, Split FaceScrub, Split Cifar100, Split Omniglot, and Split ImageNet-32 datasets. As shown in Figure 4.6, after sequential training on all tasks, SECL remains the most stable with minimal forgetting, consistently maintaining strong performance on the initial task. This is particularly evident in the challenging Split ImageNet-32 dataset, where SECL significantly outperforms its strongest competitors, FOSTER [52] and GESCL, especially from task 80 onward. SECL's ability to maintain high performance on the initial task after learning a large number of tasks is due to its effective network expansion strategy, which employs a robust filter importance heuristic to overcome forgetting and strategically expand the network as needed.

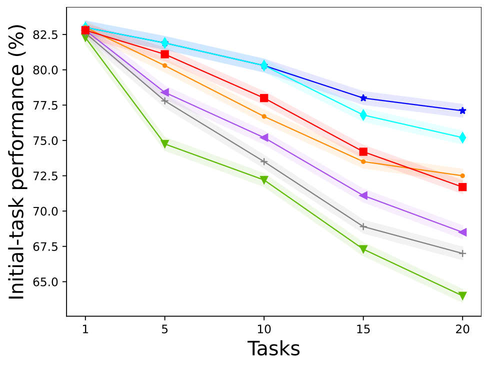
SECL
GESCL
EWC
CPG
DyTox
FOSTER
P&C



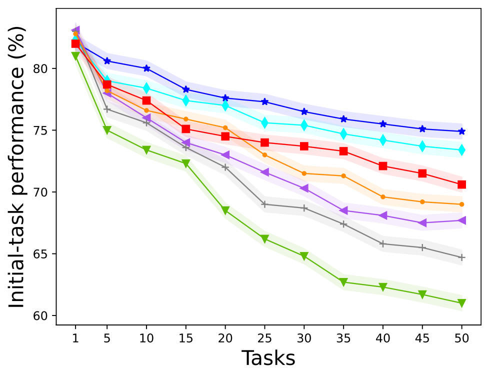
(a) Permuted MNIST



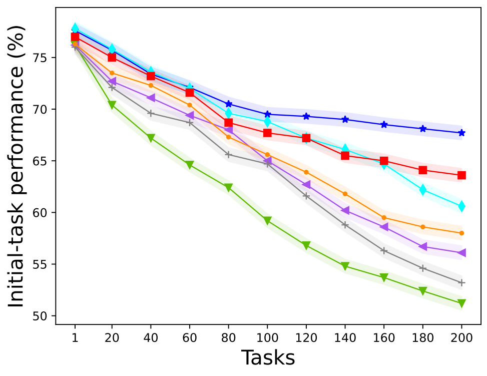
(b) Split FaceScrub



(c) Split CIFAR-100



(d) split Omniglot



(e) Split ImageNet-32

Figure 4.6: Assessing Catastrophic Forgetting by monitoring performance retention on the initial task: Results illustrate how classification accuracy changes as new tasks are introduced. Results are displayed for five various datasets. SECL exhibits the strongest resistance to catastrophic forgetting, maintaining initial task performance more consistently than other methods during the continual learning experiments.

Another well-known measure of forgetting, illustrated in Chapter 3 and detailed in Chapter 2, is the *average forgetting rate*. A lower average forgetting rate indicates a model's superior ability to retain old knowledge while acquiring new information. Table 4.2 presents a comparative analysis of the average forgetting rate across various datasets and baselines as introduced in Sections 4.3.2 and 4.3.1, respectively.

Table 4.2: Average forgetting rate. Results are averged over 5 differents run. SECL shows consistent superiority in minimizing forgetting rates across diverse datasets.

	Permuted Mnist	Cifar100	FaceScrub	Omniglot	ImageNet-32
EWC [43]	0.62	23.12	20.02	34.02	51.71
CPG [50]	0.155	9.17	6.02	21.11	29.11
P&C [49]	0.163	8.62	7.02	20.91	26.91
DyTox [51]	0.142	6.83	5.83	18.57	25.57
FOSTER [52]	0.117	6.21	4.21	14.51	22.51
GESCL	0.12	2.43	1.72	7.04	16.02
SECL	**0.11**	**1.82**	**1.24**	**3.31**	**5.88**

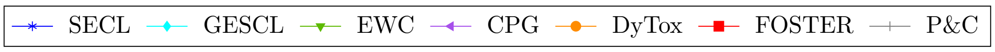
As can be seen from Table 4.2 SECL, consistently demonstrates superior performance across all datasets by achieving the lowest forgetting rates. For the Permuted Mnist dataset, SECL records a remarkably low rate of 0.11, far below EWC's [43] 0.62 and FOSTER's [52] 0.117. Similarly, on the Cifar100 dataset (20 tasks), SECL maintains a forgeting rate of 1.82, outperforming GESCL, FOSTER [52], and others. For FaceScrub (20 tasks), SECL also show the best performance with a forgetting rate of 1.24, whereas competing methods all exhibit higher values. The Omniglot dataset (50 tasks) results reflect SECL's consistent performance, with a low forgetting rate of 3.31. Finally, on the ImageNet-32 dataset, which has 200 tasks in total, SECL achieves a forgetting rate of 5.88, well below the rates of other methods like EWC [43] (51.71) and FOSTER [52] (22.51).

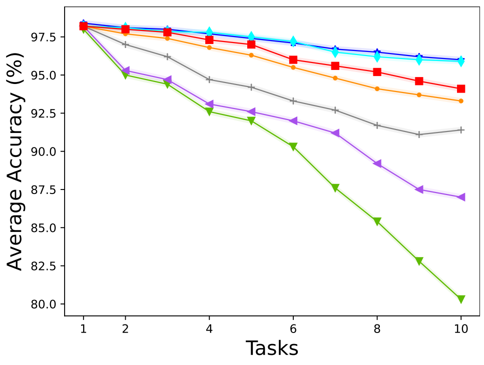
Overall, SECL outperforms state-of-the-art baselines in overcoming catastrophic forgetting, thanks to the filter importance adaptive stability regularizer, which reduces the change in vital parameters from task to task. Additionally, integrated heuristics, based on filter importance, allow for network expansion when necessary, which also helps in overcoming forgetting. DyTox [51] and FOSTER [52] are strong competitors, particularly on large datasets, also achieving high performance retention degree.

4.3.5 Overall Classification Accuracy

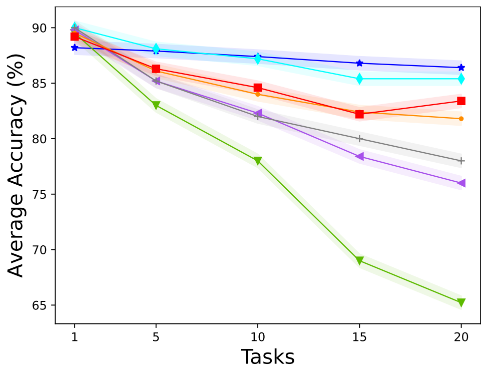
In a second experiment, the effectiveness of SECL is assessed in terms of classification accuracy during CL experiments. To evaluate classification accuracy, the average accuracy metric [76] described in chapter 2 is used, which is a common metric in the CL literature.

Figure 4.7 illustrates the evolution of the average accuracy across tasks during the continual learning experiments for five datasets: Permuted MNIST (P-MNIST), Cifar100, FaceScrub, Omniglot, and ImageNet-32.

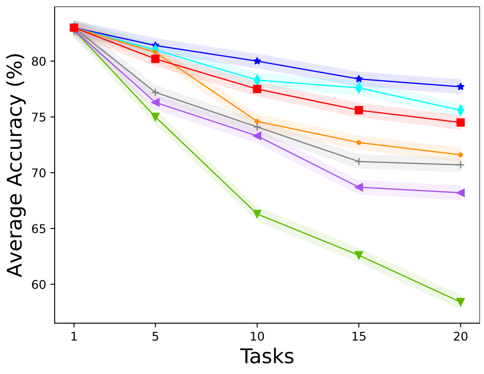




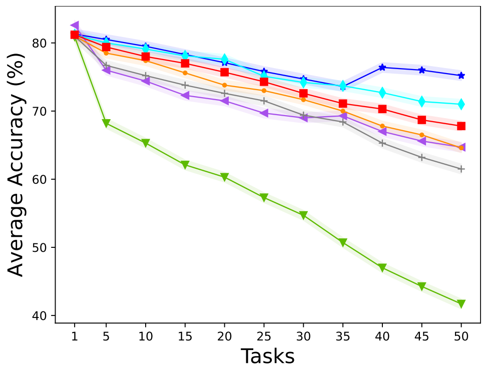
(a) Permuted MNIST



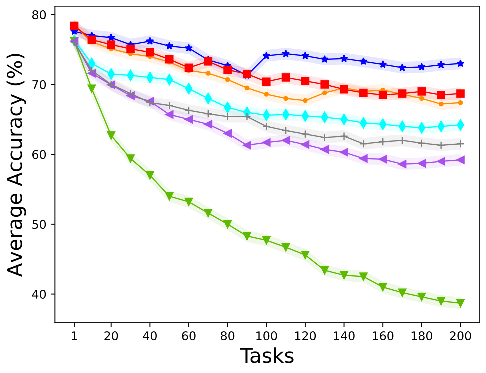
(b) Split FaceScrub



(c) Split CIFAR-100



(d) split Omniglot



(e) Split ImageNet-32

Figure 4.7: Average test classification accuracy vs. the number of observed tasks in 5 continual learning experiments. It can be seen that SECL achieves significantly higher classification results than the compared baselines.

The state-of-the-art continual learning methods presented in Section 4.3.1 are used as baseline references for comparison, enabling an assessment of how SECL performs relative to established techniques in the area of expansion-based continual learning systems.

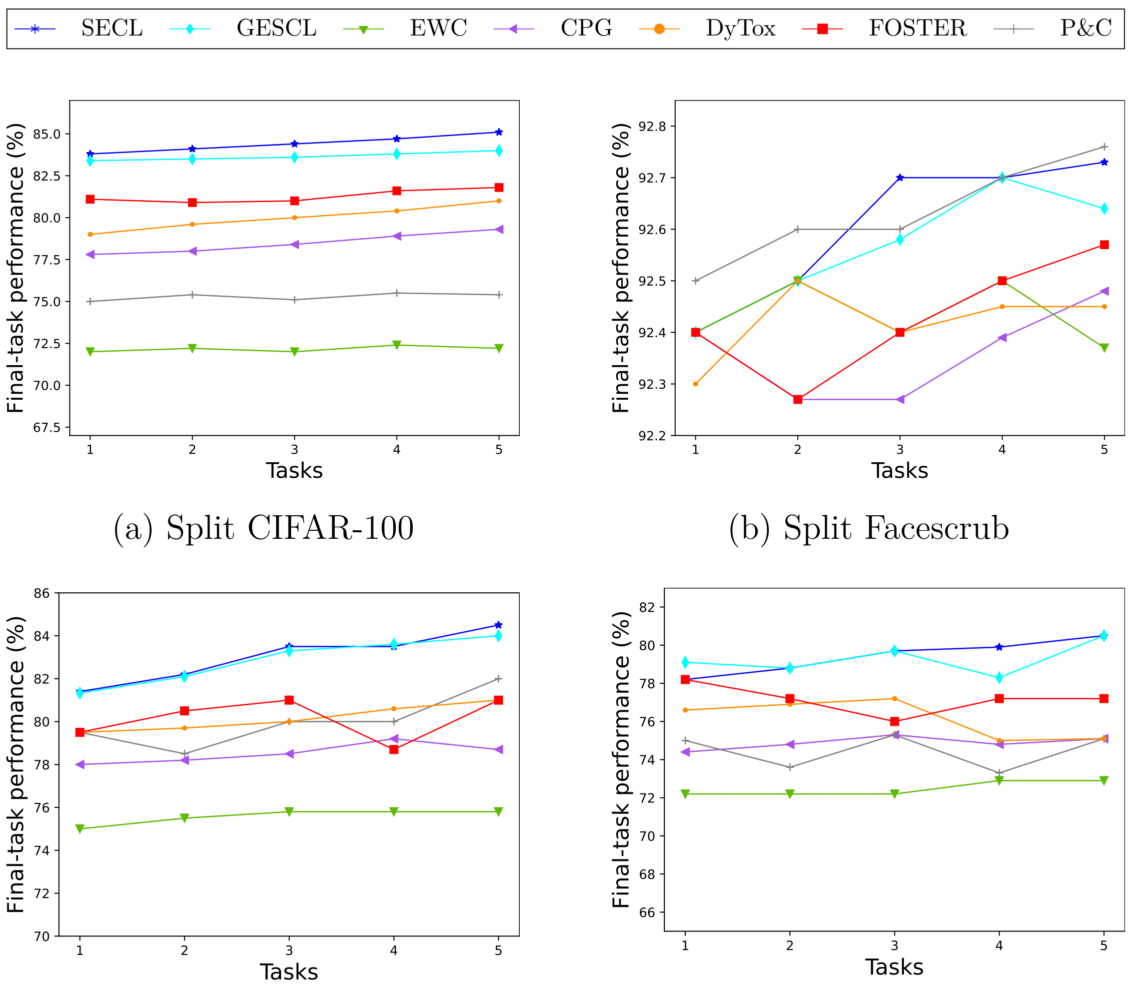
The results indicate that SECL shows strong performance in terms of average accuracy compared to other baselines on all benchmarks. An interesting observation is that GESCL [1] (the method developed in Chapter 3) performs comparably to SECL on P-MNIST, FaceScrub, and Cifar100. This was expected as these datasets are not large enough to activate SECL's extension module. In contrast, for Omniglot, which consists of 50 tasks, GESCL performs similarly to SECL on the first 34 tasks, before being significantly outperformed by SECL on the remaining 16 later tasks. Furthermore, in this experiment, a longer sequence than typically studied in the continual learning literature is considered, specifically ImageNet-32, which consists of 200 tasks. The last row of Figure 4.7 shows the average test accuracy observed throughout the 200-task sequence. As can be seen, the observations established on other datasets hold true for SECL in a very long sequence. Notably, at the later stages of the sequence, SECL exhibits a significant increase in accuracy, allowing it to outperform other expansion baselines while adding considerably less capacity (as will be shown in Section 4.7). This superior average accuracy performance, especially in the later stages of longer continual learning experiments such as Omniglot and ImageNet-32, can be attributed to the strategic heuristic implemented in the SECL learning procedure, which allows for targeted network expansion precisely when and where capacity is needed.

An empirical conclusion that can be drawn from Figures 4.6, 4.7 and Table 4.2 is that SECL achieves strong overall continual learning results thanks to the way it addresses forgetting and the learning of several new tasks through the stability-plasticity regularizers as well as the network expansion module built into the model learning procedure.

4.3.6 Assessing Forward Transfer

This section focuses on forward transfer, evaluating the impact of learning previous tasks on the performance of a new task. Specifically, it aims to determine if the knowledge acquired from prior tasks benefits the learning of a new task due to the transfer of useful features or representations from earlier tasks. This is assessed by comparing the performance on a specific task after learning varying numbers of previous tasks [49]. In the experiments described in this section, the last task is used as the reference task, evaluating the influence of learning prior tasks on this final task.

Figure 4.8 illustrates, for four different continual learning datasets, how the accuracy of the last task evolves when prior tasks are learned. The accuracy is shown after learning only the last task ($x = 1$), the last task plus one additional prior task ($x = 2$), the last task plus two additional prior tasks ($x = 3$), and so on. If the accuracy at $x = 2$ is greater than at $x = 1$, this indicates that learning a prior task has improved the performance on the last task, signifying positive transfer. Conversely, if the accuracy decreases, it indicates negative transfer, meaning that after learning the previous task, the model loses performance on the last task.



(a) Split CIFAR-100

(b) Split Facescrub

(c) Split Omniglot

(d) Split ImageNet32

Figure 4.8: Evaluating Forward Transfer on 4 datasets: The impact of learning previous tasks on a specific task (the last task) is evaluated as a proxy for assessing forward transfer. The x-axis represents the number of tasks learned. The value at $x = 1$ indicating the accuracy of the last task after learning only itself, the value at $x = 2$ indicating the accuracy of the last task accuracy after learning an additional previous task, and so on. Overall, SECL achieves state-of-the-art results in three to four experiments, demonstrating a strong ability to showcase positive transfer.

Each plot in Figure 4.8 compares the performance of SECL with several baselines: GESCL, EWC [43], CPG [50], DyTox [51], FOSTER [52], and P&C [49]. In the Split Cifar100 experiment, SECL consistently outperforms the other methods demonstrating robust forward transfer. Methods like EWC [43] and P&C [49] show significantly lower and less stable performance, indicating their limitations in effectively utilizing knowledge from prior tasks.

The Split FaceScrub dataset reveals a more varied performance among the methods. While SECL achieves strong performance and demonstrates positive transfer, it is outperformed by P&C [49], which shows the highest accuracy, especially as the number of prior learned tasks increases. This could be attributed to P&C's parameter isolation mechanism, which helps retain detailed information about facial features learned in earlier tasks. This approach is particularly effective for tasks with high variation, such as facial features in FaceScrub, as it minimizes interference between tasks and enhances forward transfer significantly. DyTox [51] and FOSTER [52] show improvement with additional prior tasks but do not match the consistent performance of P&C [49] or SECL.

In the Split Omniglot and ImageNet-32 datasets, SECL shows a clear advantage, with accuracy steadily increasing as more prior tasks are learned, indicating strong positive forward transfer. FOSTER [52] and DyTox [51] show some positive trends but with greater variability. EWC [43] remains the least effective, with a relatively flat performance curve, again reflecting its limited effectiveness in forward transfer.

Across all datasets, SECL demonstrates a strong ability to achieve positive transfer, outperforming other methods like GESCL, FOSTER [52] and DyTox [51] in most scenarios. The consistent increase of accuracy on the last task after learning each additional prior task highlights SECL's effectiveness in leveraging prior knowledge to enhance the learning of new tasks.

Although SECL is outperformed by P&C [49] in the Split FaceScrub dataset, it still shows robust performance overall, demonstrating a strong ability to showcase positive transfer and effectively avoid negative transfer. This highlights SECL's potential in real-world CL scenarios where forward transfer is crucial for integrating new knowledge without forgetting previous tasks. An example of such a scenario is adapting an autonomous warehouse robot to handle new types of inventory over time while retaining knowledge of previously encountered items and routes.

4.4 Ablation Study and Analysis

This section explores various aspects of the SECL method, including detailed ablation studies on its components, a deeper look at network expansion, a comparative analysis of capacity usage, and considerations of training and inference times.

4.4.1 In-depth Analysis of Network Expansion Effects

This section closely examines the impact of the network expansion module on model accuracy. A small convolutional neural network (CNN) was employed for this analysis, featuring two 3×3 convolution blocks with 32 filters each, followed by ReLU activation, a 2×2 max-pooling layer, and a dense layer with 64 neurons. Due to its limited size, it was anticipated that the network would quickly reach its capacity limits, providing a clear perspective on the effects of network expansion.

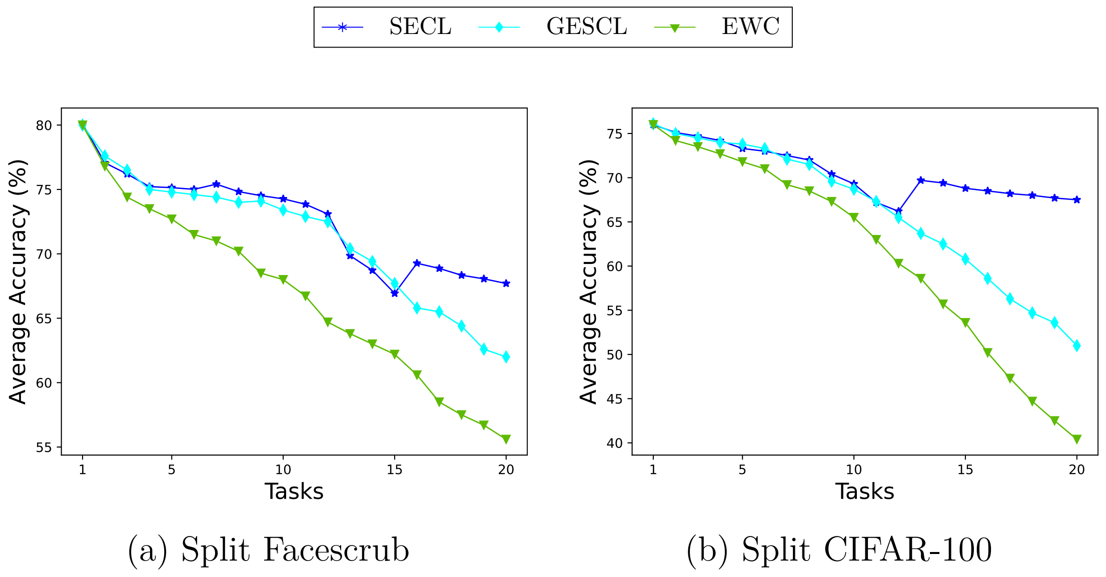


Figure 4.9: Evolution of the average accuracy of SECL, GESCL and EWC [43] on a much smaller network, which tends to reach its maximum capacity quickly. The effects of the network expansion strategy implemented in GESCL can be clearly seen in the later tasks of each dataset.

Figure 4.9 illustrates the evolution of the average classification accuracy for this small network on the Split FaceScrub and split Cifar100 datasets. The performance of the method developed in this chapter (SECL) is compared alongside the GESCL, and EWC [43] baselines.

The network capacity tends to saturate at the 15th, and 12th tasks for FaceScrub, and Cifar100 respectively. Upon reaching these saturation points, the expansion strategy of SECL is implemented, enabling the network to adapt and accommodate new tasks. Concurrently, (in SECL) the network undergoes further refinement through a sparsification mechanism. This process ensures that the newly added capacity is utilized efficiently, optimizing performance and maintaining effectiveness across tasks.

4.4.2 Ablation Study

An ablation study was conducted to investigate the contributions of each component in the SECL framework using the FaceScrub, Cifar100, Omniglot, and ImageNet-32 datasets. The study focused on the effects of the stability regularizer (λ_s), the plasticity regularizer (λ_p), and the expansion regularizer (λ_e) on the base model.

Variants of SECL were implemented with different combinations of these components, where each component was either activated (indicated by ✓) or removed (indicated by an empty space). Additionally, variants of SECL were created by deactivating two out of the three components simultaneously to measure their combined impact on the model's performance.

Table 4.3: Ablation study evaluating the impact of different components on the average accuracy (A) and average forgetting (F) of SECL variants on Split FaceScrub, Split CIFAR 100, Split Omniglot and Split ImageNet-32 after learning the last task $\mathcal{T}^T$. The concerned components are the plasticity regularizer (λ_p), the stability regularizer (λ_s), and the expansion regularizer (λ_e), each denoted by ✓ if included.

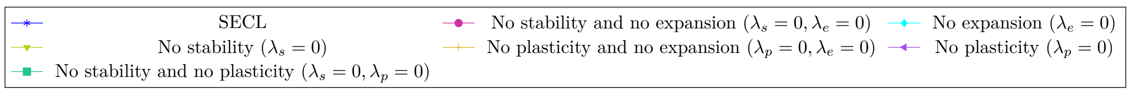
λ_s	λ_p	λ_e	Split FaceScrub		Split Cifar100		Split Omniglot		Split ImageNet-32	
			A(%)	F	A(%)	F	A(%)	F	A(%)	F
✓	✓	✓	**86.47**	**0.007**	**77.72**	**0.0083**	**75.22**	**0.085**	**73.56**	**0.091**
✓	✓		83.71	0.081	75.6	0.089	71.16	0.01	64.28	0.29
	✓	✓	43.09	0.721	41.43	0.786	33.73	0.804	15.97	0.885
✓		✓	68.93	0.044	64.44	0.0491	58.9	0.053	50.4	0.069
		✓	32.13	0.728	29.11	0.77	22.16	0.806	18.9	0.941
✓			65.22	0.11	63.8	0.118	42.70	0.144	35.8	0.206
	✓		34.57	0.731	31.6	0.767	23.8	0.792	12.7	0.905

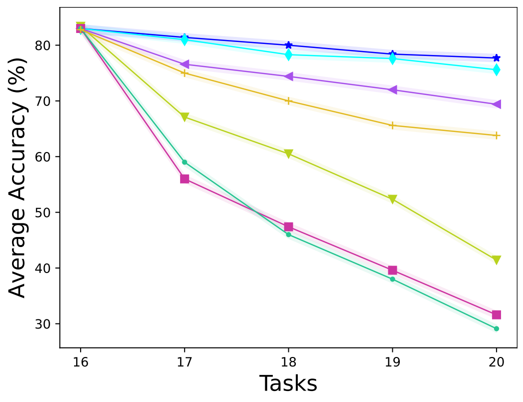
The outcomes of this ablation study are summarized in Table 4.3, where two key metrics are used for evaluation: 'A' which represents the average classification accuracy at $\mathcal{T}^T$ and 'F', which denotes the average forgetting, defined as mean decrease in performance on each task from its peak accuracy to its accuracy after all tasks have been learned [92]. These metrics are defined in detail in Section 2.9 of Chapter 2.

The results for these two metrics, shown in Table 4.3, demonstrate the effectiveness of each component in improving average classification accuracy and minimizing the level of forgetting across tasks. These results clearly indicate that the interplay between stability, plasticity, and expansion regularizers is critical to achieving the superior performance observed in the SECL framework.

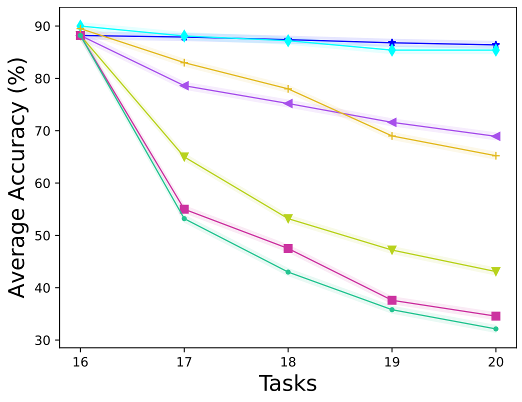
Figure 4.10 provides a detailed view of the ablation study, showing the evolution of the average accuracy for each SECL variant (where one or two of the stability-regularizer, plasticity-regularizer, or expansion-regularizer are activated or removed) across the Cifar100, FaceScrub, Omniglot, and ImageNet-32 datasets. From the figure, it can be seen that the absence of stability and expansion significantly degrades the performance across all datasets. The setups lacking stability ($\lambda_s = 0$) or both stability and plasticity ($\lambda_s = 0, \lambda_p = 0$) show a steep decline in accuracy as the number of tasks increases. This decline is especially pronounced in the Split Cifar100 and Split ImageNet-32 datasets, indicating that stability is crucial for preserving performance across multiple tasks.

Similarly, the absence of expansion ($\lambda_e = 0$) or both expansion and plasticity ($\lambda_e = 0, \lambda_p = 0$) also results in lower accuracy, but the impact is less severe compared to the absence of stability and plasticity. In datasets with a larger number of tasks like Split Omniglot and Split ImageNet-32, the difference in performance between the configurations becomes more noticeable, highlighting the cumulative effect of these components over a greater number of tasks.

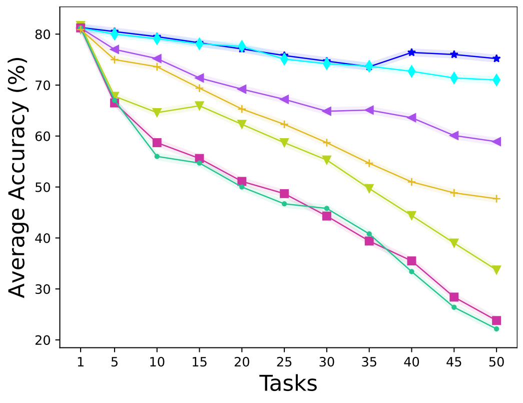

SECL
No stability ($\lambda_s = 0$)
No stability and no plasticity ($\lambda_s = 0, \lambda_p = 0$)
No stability and no expansion ($\lambda_s = 0, \lambda_e = 0$)
No plasticity and no expansion ($\lambda_p = 0, \lambda_e = 0$)
No expansion ($\lambda_e = 0$)
No plasticity ($\lambda_p = 0$)



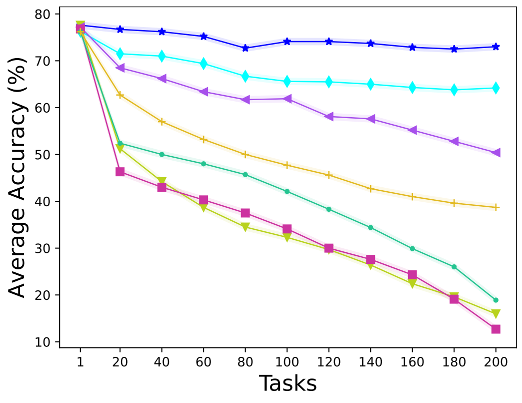
(a) Split CIFAR-100



(b) Split Facescrub



(c) Split Omniglot



(d) Split ImageNet32

Figure 4.10: Task-by-task effects of removing combinations of stability, plasticity, and expansion regularizers on SECL performance in Cifar100, FaceScrub, ImageNet-32, and Omniglot. The absence of stability ($\lambda_s = 0$) or both stability ($\lambda_s = 0$) and plasticity ($\lambda_p = 0$) results in a sharp decrease in accuracy. Although removing expansion ($\lambda_e = 0$) also reduces accuracy, the effect is less severe. The performance gap between configurations increases as the number of tasks grows and is therefore more pronounced in datasets with a large number of tasks.

4.4.3 Comparative Analysis of Used Capacity

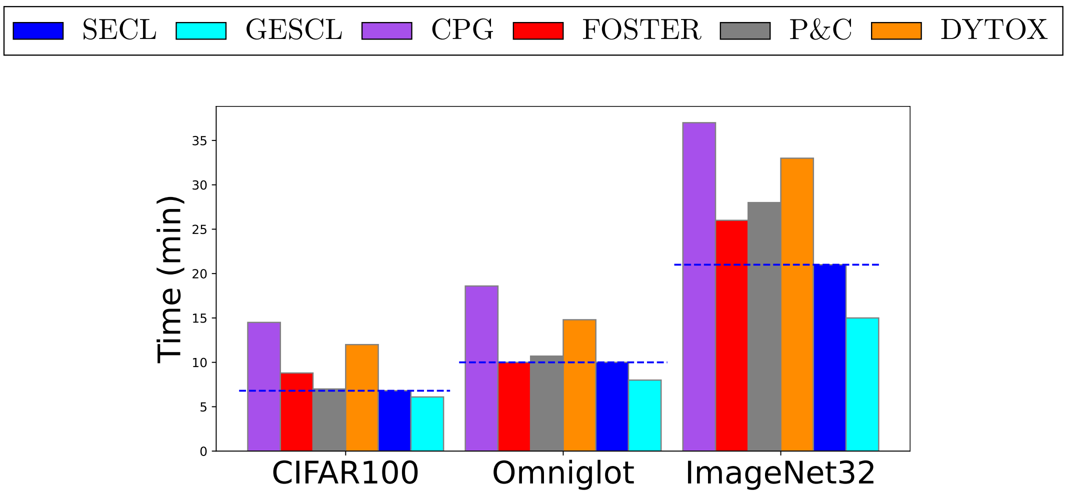
Table 4.4: Comparison of capacity usage across different methods on various datasets. The table highlights that CPG [50] consistently uses the highest capacity, while SECL demonstrates more efficient capacity utilization, maintaining a fixed capacity for smaller datasets and requiring only modest increases for larger datasets such as Omniglot and ImageNet-32, which contain 50 and 200 tasks, respectively.

	SPlit Cifar100	Split FaceScrub	Split Omniglot	Split ImageNet-32
CPG [50]	161	142	167	214
P&C [49]	131	118	144	195
DyTox [51]	121	109	131	182
FOSTER [52]	134	112	128	177
GESCL	100	100	100	**100**
SECL	100	100	109	127

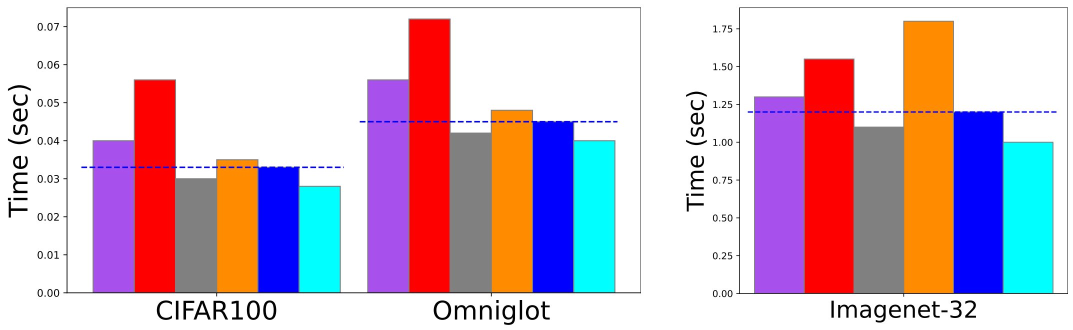
This section presents a comparative analysis of the capacity use of SECL as well as various baselines including GESCL and expansion-based baselines on different datasets. The capacity use by a method during a continual learning experiments (e.g., Split Cifar100) is computed as $\frac{|\Theta_0|}{|\Theta_T|}$, where $|\Theta_0|$ represents the number of model parameters before learning the first task, and $|\Theta_T|$ represents the number of model parameters after learning the last task. GESCL, the method developed in Chapter 3, uses a fixed-size network with a maximum capacity of 100%, which limits its ability to effectively handle a large number of sequential tasks, as observed in the experimental section of Chapter 3. In contrast, SECL and other expansion-based baselines (CPG [50], P&C [49], DyTox [51], FOSTER [52]) increase the parameters space during training, resulting in capacities that can exceed 100%.

The results demonstrate that SECL uses less capacity compared to other expansion-based methods, while still achieving better classification accuracy, as shown in previous experimental sections. More precisely, Table 4.4 shows that, CPG [50] consistently uses the highest capacity across all datasets, with values ranging from 142% to 214%. P&C [49] and DyTox [51] show moderate capacity usage, generally higher than GESCL and SECL but lower than CPG [50]. FOSTER's capacity use is mostly slightly lower than P&C [49] and DyTox [51]. Notably, SECL maintains a fixed capacity of 100 for split Cifar100 and split FaceScrub, as these datasets are not large enough to activate SECL's network expansion module. For Omniglot and ImageNet-32, SECL shows capacities of 109% and 127%, respectively, which is much lower than the capacities usage of other methods. This indicates that SECL, in addition to achieving superior average classification accuracy compared to the baselines (as discussed in Section 4.3.5), is also more efficient in terms of capacity usage, particularly when dealing with datasets of varying sizes without substantial increases in capacity requirements.

4.4.4 Training and Inference Time



(a) Training Time



(b) Inference Time

Figure 4.11: Comparison of the training time (in minutes) and inference time (in seconds) of SECL against various baseline models across three different continual learning datasets: split CIFAR100, split Omniglot, and split ImageNet-32.

Figure 4.11 (a) shows the training time required by each method (SECL and baselines) on the split CIFAR100, split Omniglot, and split ImageNet-32 datasets. The baselines include GESCL [1] and expansion-based methods such as CPG [50], FOSTER [52], P&C [49], and DyTox [51]. As expansion methods increase the model parameters for certain tasks, later tasks have more parameters to learn, resulting in increased time consumption during both training and inference phases. Figure 4.11 (a) indicates that SECL consistently achieves the shortest training times compared to the other expansion-based methods across all three datasets, demonstrating a highly efficient training process that reduces the time required to achieve its learning goals.

Regarding inference time, Figure 4.11 (b) illustrates the average time each method takes to make predictions after training. Inference time is critical because models need to provide fast and accurate responses, especially in real-world scenarios. Across all three datasets, although SECL does not have the fastest inference time (being outperformed by P&C), it remains competitive, balancing speed and accuracy effectively. This consistent performance across different datasets underscores SECL's ability to maintain competitive inference speed while ensuring robust model performance.

In summary, while SECL may be marginally slower than some baselines in inference time, this is balanced by its superior accuracy and robustness, as demonstrated earlier. Additionally, SECL outperforms all expansion-based methods in training time and several in inference time. This showcases SECL's efficiency and effectiveness in both learning and operational phases, emphasizing its adaptability and well organised resource use.

4.4.5 Limitations

When the model's capacity saturates, SECL adds filters at layers where the capacity is deemed insufficient. Specifically, SECL adds capacity at layers where the remaining capacity has fallen below a predefined threshold. The additional capacity is calculated as the difference between the initial and remaining capacity. However, this approach has its drawbacks. For instance, if the capacity saturates when only a small task t remains to be learned, the added capacity may be excessive. This strategy could be improved by dynamically estimating the capacity required for the specific task t and expanding only by the needed amount, thereby avoiding unnecessarily large expansions.

To implement this, a possible approach would be to use task-specific complexity measures, such as the model's loss gradient or task difficulty, to estimate how much additional capacity is necessary. The model could then adjust the expansion size based on these estimations. This would involve monitoring the task's contribution to the overall model performance and predicting how much more capacity is required to reach an optimal state. Such an adaptive mechanism would reduce over-expansion and make SECL more efficient for tasks of varying sizes and complexities.

4.5 Chapter Summary

A new approach, SECL, for continual learning of long task sequences has been presented. SECL addresses the stability-plasticity dilemma by incorporating a unique regularization term composed of three key components: stability, plasticity, and expansion. The stability component mitigates catastrophic forgetting by preventing gradient updates on critical hidden filters, thereby preserving essential representations from previous tasks.

The plasticity component leverage the overparameterization of neural networks to use the available network capacity sparsely before considering expansion. After training on a task t, two sets of filters are identified: (i) those crucial for learning tasks up to t, whose parameters are constrained to maintain stability, and (ii) those unimportant for task t, which are reinitialized for future tasks, ensuring network plasticity.

When the number of unimportant filters becomes rare, the expansion component activates, adding network capacity as needed based on a heuristic calculation. Furthermore, as a key ingredient, the plasticity regularizer component is applied to the parameters added during expansion, maintaining an efficient and non-redundant architecture. This mechanism enhances scalability, allowing SECL to manage more tasks without a proportional increase in network complexity or resource consumption, which is crucial for an expansion-based continual learning system. The approach also reduces training and inference time, as demonstrated in experiments.

Further evaluations on popular continual learning benchmarks highlight SECL's superiority over state-of-the-art methods in preventing catastrophic forgetting and achieving better average classification accuracy.

Chapter 5

Conclusions, Implications and Future Directions

The continual learning approaches proposed in this work enable convolutional neural networks to incrementally acquire new knowledge over time while preserving previously learned information. This work introduces a comprehensive framework consisting of two key approaches: Group and Exclusive Sparse Regularization-based Continual Learning (GECL) and Sparsification and Expansion-based Continual Learning (SECL). GECL manages the learning process by balancing stability and plasticity in networks with fixed capacity. SECL extends GESCL by adaptively expanding network capacity to handle an unlimited number of tasks. Together, these approaches effectively manage network resources, offering scalable solutions for advanced AI applications that require the ability to continuously learn new knowledge over time without forgetting, such as autonomous systems, adaptive robotics, or personalized AI assistants.

5.1 Summary of the Results

Chapter 3 addressed the challenge of continual learning in fixed-capacity neural networks. Additionally, it was assumed that data from previous tasks would be unavailable when learning new tasks, aligning with modern AI regulations such as the right to be forgotten. To address this problem, a novel approach called Group and Exclusive Sparse Regularization-based Continual Learning (GECL) was introduced. This method was detailed in Section 3.2. GECL effectively tackles several well-known challenges in continual learning, such as identifying which filters parameters $\Theta_{+}^{1:t-1}$ are crucial for previous tasks $\mathcal{T}^{t-1}$ and which are not ($\Theta_{-}^{1:t-1}$) after learning a specific task $\mathcal{T}^{t}$. Additionally, GECL addresses parameter importance at the level of entire filters, rather than individual parameters, as seen in other methods.

Another challenge addressed by GECL is the stability-plasticity dilemma, which was tackled by deriving a regularization term composed of two components: one to maintain stability and the other to manage plasticity. By integrating both stability and plasticity into a single regularizer, the approach maintains network performance across tasks without requiring rehearsal (i.e., replaying samples from old tasks while training on new ones). The stability component prevents significant changes to important filters parameters Θ_+, while the plasticity component efficiently reallocates unimportant filters parameters Θ_- for future tasks. Experimental evaluations on widely-used continual learning benchmarks confirmed that GECL achieves superior performance, both in terms of classification accuracy and in mitigating catastrophic forgetting.

Chapter 4 addressed the problem of continual learning for very long task sequences. It was observed that managing a large number of tasks presents challenges not only for GESCL but also for existing continual learning methods. To overcome this limitation, a novel approach called Sparsity Enhanced Continual Learning (SECL) was introduced and detailed in Section 4.2. SECL builds upon GECL by expanding its regularization term with an additional third component that adaptively increases network capacity, allowing it to handle an extended sequence of tasks effectively.

Similar to GECL, after training on a task t, SECL preserves critical filters to maintain stability and reinitializes unimportant filters to enhance plasticity. When the number of unimportant filters becomes insufficient, SECL activates its expansion module, adding network capacity as needed based on a heuristic calculation. Additionally, SECL implements two other heuristics to determine when and where to expand the neural network, ensuring efficient adaptation to new tasks.

Experimental results on popular continual learning benchmarks demonstrated SECL's superiority in mitigating forgetting, improving average classification accuracy and achieving forward transfer. Furthermore, SECL shows significant improvements in scalability and resource efficiency, allowing continual learning systems to handle dynamic and complex environments that require learning a large number of tasks from data streams, such as autonomous driving, real-time language translation, or adaptive robotics.

5.2 Limitations and Outlook

This section discusses the limitations of the presented methods and provides an outlook on potential directions for future work.

5.2.1 Limitations

While the proposed continual learning methods (GESCL and SECL) have demonstrated superior performance in terms of overall continual learning accuracy, forward transfer, and mitigating catastrophic forgetting compared to existing approaches, several limitations yet remain:

- The benchmarks utilized in this thesis are predominantly from the vision domain. Although vision-based benchmarks are widely used in the continual learning community, evaluating these methods on continual learning datasets from other domains, such as natural language processing (NLP), would offer a more comprehensive and reliable assessment.

- The methods developed in this thesis rely on the assumption of overparameterization in neural networks, which may not hold for narrower networks. Narrower networks, with fewer parameters, might lack the capacity to effectively learn from data streams without requiring significant expansion, compared to overparameterized networks. Additionally, narrow networks tend to exhibit different training dynamics, such as increased susceptibility to local minima and a steeper optimization landscape, making it more challenging for optimization algorithms to converge on global minima. This can impact learning stability and convergence, potentially leading to poorer generalization across tasks in a continual learning framework. While empirical evidence suggests that this issue may not be pronounced in continual learning settings, careful consideration of network capacity and generalization capabilities is essential in practical applications. It is worth noting, however, that the assumption of overparameterization is well-founded, as recent advancements in deep learning have demonstrated the effectiveness of overparameterized networks in achieving superior performance and robustness in a variety of complex tasks.
- Another limitation of the methods developed in this thesis is their primary evaluation on classification tasks within a supervised learning framework. This focus may restrict the applicability of the results to other types of tasks, such as regression or unsupervised learning, where different dynamics and challenges may emerge. Additionally, the reliance on supervised learning presupposes the availability of labeled data, which may not always be feasible or practical in real-world scenarios. As a result, the generalizability of these methods to diverse tasks and learning paradigms may be limited, underscoring the need for further exploration and adaptation to broader contexts.

5.2.2 Outlook

The initial findings from the methods developed in this thesis highlight several promising directions for future research. This section outlines these potential directions, exploring how they could further refine and expand the capabilities of the proposed approaches. In particular, addressing the limitations (mentioned in Section 5.2.1) could lead to significant advancements.

5.2.2.1 Generalization to Dense Neural Networks

This thesis has primarily focused on convolutional neural networks (CNNs), where filters are central to the developed methods. The stability-plasticity dilemma, a core challenge in continual learning, was addressed by focusing on filters rather than individual kernel parameters, as is common in many other approaches. For instance, the method for identifying important network parameters after training on each task t involved evaluating entire filters, thereby determining crucial filters instead of individual parameters, as typically done in the literature. Stability was maintained by constraining important filters to undergo minimal changes, while plasticity was achieved by using a sparsity regularizer to keep as many filters as possible unused during training (Chapters 3 and 4) or through network expansion (Chapter 4).

An interesting direction for future research is to generalize these methods to dense neural networks. One possible approach is to replace filters with neurons in dense networks. In CNNs, the methods proposed in this thesis determine the importance of a filter by the activation of its 2D kernel parameters, while in dense networks the importance of a neuron could be represented by its incoming connections. In other words, in a dense network, a neuron corresponds to a filter, and a connection represents a 2D kernel. The importance of a neuron would then be determined by the activation of its incoming connections. A potential concern with this approach is whether the total activation of a neuron accurately reflects its importance in learning a task, as has been demonstrated in the case of CNN filters. This topic requires further investigation to assess the feasibility and effectiveness of this generalization.

5.2.2.2 Enhancing Sparsity through Data-Centric Approaches

Neural network sparsity is a key component of the continual learning methods developed in this thesis, significantly contributing to the network's ability to maintain high plasticity and learn multiple tasks sequentially without requiring architectural expansion. A promising area of research for enhancing sparsity is data pruning, which involves selectively removing redundant or irrelevant features or samples from a dataset, thereby refining the dataset to focus on the most important information or representation for the target variable. This approach promotes weight sparsity, as the weights associated with pruned features are either not learned or effectively reduced to zero. When input data is sparse, meaning many features have zero or near-zero values, the model naturally develops sparse weights. Weights associated with features that contribute minimally to the model's predictions tend to decrease in magnitude or become negligible, supporting the overall sparsity of the network. Data pruning can be applied before or early in the training process, making it highly compatible with continual learning systems and facilitating seamless integration that supports efficient, ongoing task learning.

Data pruning offers a promising area for enhancing the methods developed in this thesis. It could potentially allow the neural network model to handle a larger number of tasks, especially in scenarios where model capacity is fixed, as discussed in Chapter 3. Data pruning could also be integrated into the expansion-based continual learning method proposed in Chapter 4. For example, after reaching saturation from learning tasks $\mathcal{T}^1$ to $\mathcal{T}^{t-1}$, each time new capacity is added for task $\mathcal{T}^t$ onwards, applying data pruning to the newly added tasks could result in a more sparsified network, thereby enhancing the system's overall plasticity. This approach could delay the point of saturation as more tasks are learned, allowing the network to continue learning additional tasks before reaching its capacity limits.

Finally, data pruning could improve the model's efficiency by reducing complexity, potentially leading to faster training times and lower resource requirements, while also increasing interpretability by focusing on the most impactful features.

Bibliography

Publications by the Author

[1] Basile Tousside, Janis Mohr, and Jörg Frochte. “Group and Exclusive Sparse Regularization-based Continual Learning of CNNs”. In: *Proceedings of the Canadian Conference on Artificial Intelligence* (May 2022). https://caiac.pubpub.org/pub/fgmh6oq8.

[2] Basile Tousside, Jörg Frochte, and Tobias Meisen. “CNNs Sparsification and Expansion for Continual Learning.” In: *ICAART (2)*. 2024, pp. 110–120.

[3] Basile Tousside, Lukas Friedrichsen, and Jörg Frochte. “Towards Robust Continual Learning using an Enhanced Tree-CNN.” In: *ICAART (3)*. 2022, pp. 316–325.

[4] Basile Tousside, Jörg Frochte, and Tobias Meisen. “Dynamic Capacity Expansion in Continual Learning: The eSECL Approach”. In: (2025).

[5] Janis Mohr, Basile Tousside, Marco Schmidt, and Jörg Frochte. “Multiple Additive Neural Networks: A Novel Approach to Continuous Learning in Regression and Classification”. In: (2023).

[6] Janis Mohr, Basile Tousside, Marco Schmidt, and Jörg Frochte. “Explainability and Continuous Learning with Capsule Networks.” In: *KDIR*. 2021, pp. 264–273.

General Publications

[7] Michael McCloskey and Neal J Cohen. “Catastrophic interference in connectionist networks: The sequential learning problem”. In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[8] Rahaf Aljundi. “Continual learning in neural networks”. In: *arXiv preprint arXiv:1910.02718* (2019).

[9] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. “Continual lifelong learning with neural networks: A review”. In: *Neural networks* 113 (2019), pp. 54–71.

[10] Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. “Three types of incremental learning”. In: *Nature Machine Intelligence* 4.12 (2022), pp. 1185–1197.

[11] Dongwan Kim and Bohyung Han. “On the Stability-Plasticity Dilemma of Class-Incremental Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2023, pp. 20196–20204.

[12] Dahuin Jung, Dongjin Lee, Sunwon Hong, et al. “New Insights for the Stability-Plasticity Dilemma in Online Continual Learning”. In: *arXiv preprint arXiv:2302.08741* (2023).

[13] Fahad Sarfraz, Elahe Arani, and Bahram Zonooz. “Error Sensitivity Modulation based Experience Replay: Mitigating Abrupt Representation Drift in Continual Learning”. In: *arXiv preprint arXiv:2302.11344* (2023).

[14] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, et al. “On tiny episodic memories in continual learning”. In: *arXiv preprint arXiv:1902.10486* (2019).

[15] Mengyao Zhai, Lei Chen, Frederick Tung, et al. “Lifelong gan: Continual learning for conditional image generation”. In: *Proceedings of the IEEE/CVF international conference on computer vision.* 2019, pp. 2759–2768.

[16] Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat. “Generative models from the perspective of continual learning”. In: *2019 International Joint Conference on Neural Networks (IJCNN).* IEEE. 2019, pp. 1–8.

[17] Samuel Kessler, Bethan Thomas, and Salah Karout. “Continual-wav2vec2: an application of continual learning for self-supervised automatic speech recognition”. In: *arXiv preprint arXiv:2107.13530* (2021).

[18] Anuj Diwan, Ching-Feng Yeh, Wei-Ning Hsu, et al. “Continual Learning for On-Device Speech Recognition Using Disentangled Conformers”. In: *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE. 2023, pp. 1–5.

[19] Ju Xu and Zhanxing Zhu. “Reinforced continual learning”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[20] Jean-Baptiste Gaya, Thang Doan, Lucas Caccia, et al. “Building a subspace of policies for scalable continual learning”. In: *arXiv preprint arXiv:2211.10445* (2022).

[21] Jacob Armstrong and David A Clifton. “Continual learning of longitudinal health records”. In: *2022 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI).* IEEE. 2022, pp. 01–06.

[22] Chaitanya Baweja, Ben Glocker, and Konstantinos Kamnitsas. “Towards continual learning in medical imaging”. In: *arXiv preprint arXiv:1811.02496* (2018).

[23] Tongtong Wu, Massimo Caccia, Zhuang Li, et al. “Pretrained language model in continual learning: A comparative study”. In: *International Conference on Learning Representations.* 2021.

[24] Anastasia Razdaibiedina, Yuning Mao, Rui Hou, et al. “Progressive prompts: Continual learning for language models”. In: *arXiv preprint arXiv:2301.12314* (2023).

[25] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. “Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks”. In: *Journal of Intelligent & Robotic Systems* 105.1 (2022), p. 9.

[26] Sayantan Auddy, Jakob Hollenstein, Matteo Saveriano, Antonio Rodríguez-Sánchez, and Justus Piater. “Continual learning from demonstration of robotics skills”. In: *Robotics and Autonomous Systems* 165 (2023), p. 104427.

[27] Basile Tousside, Yashwanth Dama, and Jörg Frochte. “Towards Explainability in Modern Educational Data Mining: A Survey.” In: *KDIR*. 2022, pp. 212–220.

[28] Basile Tousside, Janis Mohr, Marco Schmidt, and Jörg Frochte. “A Learning Approach for Optimizing Robot Behavior Selection Algorithm”. In: *Intelligent Robotics and Applications: 13th International Conference, ICIRA 2020, Kuala Lumpur, Malaysia, November 5–7, 2020, Proceedings 13*. Springer. 2020, pp. 171–183.

[29] Hayk Asatryan, Basile Tousside, Janis Mohr, et al. “Exploring Student Expectations and Confidence in Learning Analytics”. In: *Proceedings of the 14th Learning Analytics and Knowledge Conference*. 2024, pp. 892–898.

[30] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. “A survey of convolutional neural networks: analysis, applications, and prospects”. In: *IEEE transactions on neural networks and learning systems* 33.12 (2021), pp. 6999–7019.

[31] Mohammad Mustafa Taye. “Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions”. In: *Computation* 11.3 (2023), p. 52.

[32] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. “Expert gate: Lifelong learning with a network of experts”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3366–3375.

[33] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. “Efficient lifelong learning with a-gem”. In: *arXiv preprint arXiv:1812.00420* (2018).

[34] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. “Riemannian walk for incremental learning: Understanding forgetting and intransigence”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 532–547.

[35] Alexander Gepperth and Cem Karaoguz. “A bio-inspired incremental learning architecture for applied perceptual problems”. In: *Cognitive Computation* 8.5 (2016), pp. 924–934.

[36] Amir Rosenfeld and John K Tsotsos. “Incremental learning through deep adaptation”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.3 (2018), pp. 651–663.

[37] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. “Selfless sequential learning”. In: *arXiv preprint arXiv:1806.05421* (2018).

[38] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. “Continual learning with deep generative replay”. In: *Advances in neural information processing systems* 30 (2017).

[39] Wickliffe C Abraham and Anthony Robins. “Memory retention–the synaptic stability versus plasticity dilemma”. In: *Trends in neurosciences* 28.2 (2005), pp. 73–78.

[40] Mark Ring. “Learning sequential tasks by incrementally adding higher orders”. In: *Advances in neural information processing systems* 5 (1992).

[41] Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, et al. “Deep class-incremental learning: A survey”. In: *arXiv preprint arXiv:2302.03648* (2023).

[42] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. “Ablation studies in artificial neural networks”. In: *arXiv preprint arXiv:1901.08644* (2019).

[43] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[44] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. “Memory aware synapses: Learning what (not) to forget”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 139–154.

[45] Arun Mallya and Svetlana Lazebnik. “Packnet: Adding multiple tasks to a single network by iterative pruning”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.

[46] Zhizhong Li and Derek Hoiem. “Learning without forgetting”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[47] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. “Continual learning via neural pruning”. In: *arXiv preprint arXiv:1903.04476* (2019).

[48] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, et al. “Progressive neural networks”. In: *arXiv preprint arXiv:1606.04671* (2016).

[49] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, et al. “Progress & compress: A scalable framework for continual learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4528–4537.

[50] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, et al. “Compacting, picking and growing for unforgetting continual learning”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[51] Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. “Dytox: Transformers for continual learning with dynamic token expansion”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 9285–9295.

[52] Fu-Yun Wang, Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. "Foster: Feature boosting and compression for class-incremental learning". In: *European conference on computer vision*. Springer. 2022, pp. 398–414.

[53] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. "Tree-CNN: a hierarchical deep convolutional neural network for incremental learning". In: *Neural Networks* 121 (2020), pp. 148–160.

[54] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. "Lifelong learning with dynamically expandable networks". In: *arXiv preprint arXiv:1708.01547* (2017).

[55] Zhuo Li, Hengyi Li, and Lin Meng. "Model compression for deep neural networks: A survey". In: *Computers* 12.3 (2023), p. 60.

[56] Giosué Cataldo Marinó, Alessandro Petrini, Dario Malchiodi, and Marco Frasca. "Deep neural networks compression: A comparative survey and choice recommendations". In: *Neurocomputing* 520 (2023), pp. 152–170.

[57] Nima Aghli and Eraldo Ribeiro. "Combining weight pruning and knowledge distillation for cnn compression". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 3191–3198.

[58] Geon Yeong Park, Sangmin Lee, Sang Wan Lee, and Jong Chul Ye. "Training debiased subnetworks with contrastive weight pruning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 7929–7938.

[59] Lucas Beyer, Xiaohua Zhai, Amélie Royer, et al. "Knowledge distillation: A good teacher is patient and consistent". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10925–10934.

[60] Miao Zhang, Li Wang, David Campos, et al. "Weighted mutual learning with diversity-driven model compression". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 11520–11533.

[61] Elias Frantar and Dan Alistarh. "Optimal brain compression: A framework for accurate post-training quantization and pruning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 4475–4488.

[62] Andrey Kuzmin, Markus Nagel, Mart Van Baalen, Arash Behboodi, and Tijmen Blankevoort. "Pruning vs Quantization: Which is Better?" In: *Advances in Neural Information Processing Systems* 36 (2024).

[63] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[64] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[65] Yuval Netzer, Tao Wang, Adam Coates, et al. "Reading digits in natural images with unsupervised feature learning". In: (2011).

[66] Hong-Wei Ng and Stefan Winkler. "A data-driven approach to cleaning large face datasets". In: *2014 IEEE international conference on image processing (ICIP)*. IEEE. 2014, pp. 343–347.

[67] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. “The German traffic sign recognition benchmark: a multi-class classification competition”. In: *The 2011 international joint conference on neural networks*. IEEE. 2011, pp. 1453–1460.

[68] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. “A downsampled variant of imagenet as an alternative to the cifar datasets”. In: *arXiv preprint arXiv:1707.08819* (2017).

[69] Jia Deng, Wei Dong, Richard Socher, et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[70] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017).

[71] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[72] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. “An empirical investigation of catastrophic forgetting in gradient-based neural networks”. In: *arXiv preprint arXiv:1312.6211* (2013).

[73] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. “Overcoming catastrophic forgetting with hard attention to the task”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4548–4557.

[74] Jonathan Schwarz, Wojciech M. Czarnecki, Jelena Luketina, et al. “Progress & Compress: A scalable framework for continual learning”. In: *ArXiv* abs/1805.06370 (2018). URL: https://api.semanticscholar.org/CorpusID:21718339.

[75] Fu Lee Wang, Da-Wei Zhou, Han-Jia Ye, and De-chuan Zhan. “FOSTER: Feature Boosting and Compression for Class-Incremental Learning”. In: *ArXiv* abs/2204.04662 (2022). URL: https://api.semanticscholar.org/CorpusID:248085866.

[76] David Lopez-Paz and Marc’Aurelio Ranzato. “Gradient episodic memory for continual learning”. In: *Advances in neural information processing systems* 30 (2017).

[77] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual learning through synaptic intelligence”. In: *International conference on machine learning*. PMLR. 2017, pp. 3987–3995.

[78] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. “Less-forgetting learning in deep neural networks”. In: *arXiv preprint arXiv:1607.00122* (2016).

[79] Chrisantha Fernando, Dylan Banarse, Charles Blundell, et al. “Pathnet: Evolution channels gradient descent in super neural networks”. In: *arXiv preprint arXiv:1701.08734* (2017).

[80] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. “Variational continual learning”. In: *arXiv preprint arXiv:1710.10628* (2017).

[81] Man Zhou, Jie Xiao, Yifan Chang, et al. “Image de-raining via continual learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4907–4916.

[82] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. “Matching networks for one shot learning”. In: *Advances in neural information processing systems* 29 (2016), pp. 3630–3638.

[83] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. “Provable filter pruning for efficient neural networks”. In: *arXiv preprint arXiv:1911.07412* (2019).

[84] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. “Learning structured sparsity in deep neural networks”. In: *arXiv preprint arXiv:1608.03665* (2016).

[85] Yang Zhou, Rong Jin, and Steven Chu–Hong Hoi. “Exclusive lasso for multi-task feature selection”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 988–995.

[86] Neal Parikh and Stephen Boyd. “Proximal algorithms”. In: *Foundations and Trends in optimization* 1.3 (2014), pp. 127–239.

[87] Hao-Jun Michael Shi, Shenyinying Tu, Yangyang Xu, and Wotao Yin. “A primer on coordinate descent algorithms”. In: *arXiv preprint arXiv:1610.00040* (2016).

[88] Gobinda Saha, Isha Garg, and Kaushik Roy. “Gradient projection memory for continual learning”. In: *arXiv preprint arXiv:2103.09762* (2021).

[89] Yaroslav Bulatov. “Notmnist dataset”. In: *Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09/notmnist-dataset. html* 2 (2011).

[90] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[91] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. “Gradient based sample selection for online continual learning”. In: *Advances in neural information processing systems* 32 (2019).

[92] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. “Understanding the role of training regimes in continual learning”. In: *arXiv preprint arXiv:2006.06958* (2020).

[93] Pravendra Singh, Vinay Kumar Verma, Pratik Mazumder, Lawrence Carin, and Piyush Rai. “Calibrating cnns for lifelong learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15579–15590.

[94] Tom Veniat, Ludovic Denoyer, and Marc’Aurelio Ranzato. “Efficient continual learning with modular networks and task-driven priors”. In: *arXiv preprint arXiv:2012.12631* (2020).

List of Figures

List of Tables