11-2022

Continual learning with neural networks

PHAM HONG QUANG
Singapore Management University, hqpham.2017@phdis.smu.edu.sg

Continual Learning with Neural Networks

PHAM Hong Quang

SINGAPORE MANAGEMENT UNIVERSITY

2022

Continual Learning with Neural Networks

PHAM Hong Quang

Submitted to School of Computing & Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

SINGAPORE MANAGEMENT UNIVERSITY

2022

Declaration of Authorship

I hereby declare that this dissertation is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in this dissertation.

This dissertation has also not been submitted for any degree in any university previously.

Quang

PHAM Hong Quang

November 2022

Continual Learning with Neural Networks

PHAM Hong Quang

Abstract

Recent years have witnessed tremendous successes of artificial neural networks in many applications, ranging from visual perception to language understanding. However, such achievements have been mostly demonstrated on a large amount of labeled data that is static throughout learning. In contrast, real-world environments are always evolving, where new patterns emerge and the older ones become inactive before reappearing in the future. In this respect, *continual learning* aims to achieve a higher level of intelligence by learning online on a data stream of several tasks. As it turns out, neural networks are not equipped to learn continually: they lack the ability to facilitate knowledge transfer and remember the learned skills. Therefore, this thesis has been dedicated to developing effective continual learning methods and investigating their broader impacts on other research disciplines.

Towards this end, we have made several contributions to facilitate continual learning research. First, we contribute to the classical continual learning framework by analyzing how Batch Normalization affects different replay strategies. We discovered that although Batch Normalization facilitates continual learning, it also hinders the performance of older tasks. We named this the *cross-task normalization phenomenon* and conducted a comprehensive analysis to investigate and alleviate its negative effects.

Then, we developed a novel *fast and slow learning* framework for continual learning based on the *Complementary Learning Systems* [62, 14] of human learning. Particularly, the fast and slow learning principle suggests to model continual learning at two levels: general representation learning and learning of individual experience. This principle has been the main tool for us to address the challenges of learning new skills while remembering old knowledge in continual learning. We first realized the fast-and-slow learning principle in Contextual Transformation Networks (CTN) as an efficient and effective online continual learning algorithm. Then, we proposed DualNets, which incorporated representation learning into continual learning and proposed an effective strategy to utilize general representations for better supervised learning. DualNets not only addresses CTN's limitations but is also applicable to general continual learning settings. Through extensive experiments, our findings suggest that DualNets is an effective and achieved strong results in several challenging continual learning settings, even in the complex scenarios of limited training samples or distribution shifts.

Furthermore, we went beyond the traditional image benchmarks to test the proposed fast-and-slow continual learning framework on the online time series forecasting problem. We proposed Fast and Slow Networks (FSNet) as a radical approach to online time series forecasting by formulating it as a continual learning problem. FSNet leverages and improves upon the fast-and slow learning principle to address two major time series forecasting challenges: fast adaptation to concept drifts and learning of recurring concepts. From experiments with both real and synthetic datasets, we found FSNet's promising capabilities in dealing with concept drifts and recurring patterns.

Finally, we conclude the dissertation with a summary of our contributions and an outline of potential future directions in continual learning research.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BN	Batch Normalization
CN	Continual Normalization
CL	Continual Learning
CLS	Complementary Learning Systems
CTN	Contextual Transformation Network
ER	Experience Replay
FSNet	Fast and Slow learning Network
SGD	Stochastic Gradient Descent
TA	Task-Aware
TF	Task-Free

Acknowledgements

Obtaining a PhD with impactful research was not an easy task, and without the immense support of many people, this would not have been possible.

First, I would like to express my deepest appreciation to my advisor, Prof. Steven C.H. Hoi, for his guidance and support. But even more than that, I thank Prof. Hoi for the opportunity to pursue a PhD in his group, for teaching me how to research, and for creating the environment to grow and become the best researcher I could be. I am also thankful to the members of my dissertation committee: Prof. Jing Jiang, Prof. Akshat Kumar, and Prof. Kun Zhang, for the insightful feedback and fruitful discussions throughout my PhD.

I want to thank my senior teammates, Dr. Chenghao Liu and Dr. Doyen Sahoo, whom I have the privilege to collaborate with. They acted as my secondary supervisors, guided me through difficult times, and made me a better researcher. I also want to thank other collaborators: Dr. Binh Nguyen, Dr. Nguyen Viet Cuong, Duy Nguyen, and Thu Nguyen, for the interesting discussions about various research aspects throughout the year.

My PhD Journey would not be complete without sharing the best and toughest moments with my friends at SMU. I would like to thank my friends, including Dr. Hung Le, Dr. Thong Hoang, Dr. Xiongwei Wu, Dr. Tuan Truong, Dr. Thien Nguyen, Dr. Trong Le, Dr. Nam Doan, Trong Nguyen, Hoang Le, Hieu Do, Tai Nguyen, and Kien Nguyen, for their support not only in research but also in personal development. I particularly enjoy swimming and hiking with Hoang Le and Hieu Do, and puzzle solving with Tai Nguyen and Kien Nguyen.

Last but not least, I thank my family, my parents, and my brother, for their unconditional support and encouragement throughout all these years.

Chapter 1

Introduction

1.1 Motivation

Our world is always evolving, with new information appearing sequentially while older patterns might remain inactive for a long time before re-emerging. Such an ever-changing environment requires humans to constantly adapt to new information while retaining the learned knowledge for future usage. Similarly, Artificial Intelligence (AI) agents must also be able to operate in the same dynamic world. Thus, developing agents that can continually learn through time, accumulate, and retain knowledge has been one of the hallmark challenges in AI.

In recent years, deep learning has demonstrated remarkable success in solving cognitive tasks, even outperforming humans in some individual problems such as visual recognition [92] and Atari games [54, 166]. As a result, deep neural networks have become one of the most popular tools for many AI studies. However, much of the existing works often train the networks in a batch learning (offline) process, where a large amount of data of a pre-defined problem is collected, then the networks are trained until convergence before being evaluated. By doing so, the batch learning setting assumes a static world with all patterns available at the beginning, which is clearly different from practice. Whenever a new pattern emerges, the trained model becomes outdated, and the whole training process has to start over again with both new and old data mixed altogether. In the current big data era, data is generated in huge volumes at high speeds. Therefore, such static models will quickly become obsolete, and re-training them from scratch to accommodate new information is computationally infeasible. Consequently, without a mechanism allowing AI agents to learn efficiently in the real world, practical and effective machine intelligence cannot be achieved.

Different from the traditional batch training, continual learning focuses on learning a data stream consisting of multiple problems. On-the-fly learning makes continual learning models highly scalable and applicable to solving different tasks. However, deep neural networks are not naturally equipped which the capabilities to achieve continual learning. They catastrophically forget the learned information [9, 21, 15] and lack the ability to accumulate knowledge to facilitate future learning [125], which is referred to as the *stability-plasticity* dilemma [1, 6]. Despite extensive efforts in continual learning research, existing studies suffer from *two* major drawbacks. First, they often achieve a poor stability-plasticity trade-off or require extensive memory/computational resources to achieve satisfactory performances. Second, they are often developed for a specific scenario and may not perform well in general settings and applications. This dissertation investigates the problem of training deep neural networks to learn continually. We aim to develop a deeper understanding of the stability-plasticity dilemma and devise a general continual learning framework that is effective and applicable to different continual learning scenarios. The rest of this Chapter is organized as follows. In Section 1.2, we detail the continual learning problem and summarize its challenges. Then, we outline the main contribution of this dissertation in Section 1.3. Finally, we conclude this Chapter with an outline of the dissertation structure in Section 1.4.

1.2 Continual Learning

1.2.1 Overview

Continual learning, also referred to as lifelong learning or incremental learning, is the general problem of training a model on data streams consisting of multiple tasks. Such a data stream is called the *continuum*. The main goal of continual learning is to train a model that can solve all observed tasks so far at any time during training. For example, a model is trained to classify *cat vs doc* and then followed by *lion vs tiger*. Different from the traditional batch training, the learner is not restricted to solving only pre-defined tasks but always has to solve new problems while maintaining the ability to solve the learned ones.

Although continual learning is a natural task for humans, it is extremely difficult for AI agents, especially deep neural networks. Early studies have noticed the *catastrophic forgetting* phenomenon when training neural networks on sequential data: newly learned knowledge overwrites the older one, resulting in the inability to retrieve learned patterns. One popular hypothesis is that the *distributed representation* property, while supporting the generalization ability to new data, is also a culprit causing catastrophic forgetting [10, 13, 21]. Particularly in neural networks, a pattern is encoded using many neurons, and each neuron, in turn, contributes to the activation of several patterns. Therefore, learning a new pattern will modify several neurons, which interferes with the activation of the learned ones, resulting in the inability to retrieve the learned patterns. Lastly, the catastrophic forgetting effect is more severe for deep neural networks than the shallower ones since changes in the shallow layers may cascade, resulting in significant changes in the final output [197].

Another equally important aspect of continual learning is the ability to facilitate forward knowledge transfer across tasks to improve the learning outcomes. Since traditional deep neural networks require many training samples and computational resources to achieve good performances, they have been shown to achieve poor performances and slow convergences when naively training on data streams [115, 124]. In contrast, continual learning presents additional information as the task structure, which the learner should exploit to improve its performance. Particularly, a model already trained on a large number of tasks should leverage its learned knowledge to solve a new problem faster than another one trained from scratch. This ability is often referred to as positive *forward transfer*. Moreover, learning to solve new tasks will accumulate additional knowledge that can be useful to improve the performances of past tasks, which is called positive *backward transfer*. Unfortunately, achieving positive forward and backward transfer is difficult since they contradict each other, creating the infamous *stability-plasticity* dilemma [1, 6]. Focusing on learning new tasks will make the model too plastic and prone to catastrophic forgetting, causing negative backward transfer. Similarly, since old tasks' data is limited, a too stable model that only preserves the learned knowledge will hinder the learning of new tasks, causing negative forward transfer. Therefore, the main goal of continual learning is to develop general, efficient algorithms that achieve a good trade-off between facilitating the positive forward transfer and alleviating negative backward transfer (catastrophic forgetting) under a reasonable complexity.

Continual learning is not only a stand-alone research problem but also a ubiquitous challenge in many real-world applications. After promising progress on the image recognition task, continual learning has been gaining much interest and moving to more domains such as medical image analysis [176, 94] and practical applications such as recommendation systems [190, 200]. For example, consider an online retail platform that deploys a recommendation system to suggest useful items to users. Most users do not always buy items continuously. Instead, they only shop when needed or once in a while as a leisure activity. Therefore, most users can have long periods of inactivity, which requires the model to remember the users' preferences. Moreover, new users are always joining the platform but only have limited interactions in the beginning. Thus, it is beneficial that the system can quickly transfer knowledge from existing users to provide meaningful suggestions. Such aspects of real-world recommendation systems are highly similar to continual learning, which has motivated the developments of the intersection between continual learning and recommendation systems. Consequently, continual learning has become ubiquitous in many real-world scenarios with a strong impact on many applications.

1.2.2 Challenges

Although simply storing all data and retraining before testing is a trivial approach to continual learning, it is infeasible in practice because of the computationally or privacy concerns. As such, we list below a list of *the most important* desiderata, which must be met in a continual learning system.

1. **Constant or sublinear memory growth** We refer to the memory as the storage required to store the model's parameter and additional information of each task to assist future learning. Ideally, except for the classifier,

the model should not be expensively expanded after each task, and the memory size should grow *sublinearly* with the number of tasks.

2. **Positive forward transfer** The model should leverage its previous knowledge to improve the learning of newer tasks. Thus, a continual learner should learn newer skills more efficiently than a learned trained from scratch.

3. **Graceful forgetting - limited backward transfer** Naturally, continual learning will cause negative backward transfer and result in catastrophic forgetting. Therefore, an effective continual learner should be able to maintain its learned knowledge and alleviate forgetting.

Simultaneously satisfying the above criteria is the main challenge of continual learning: developing efficient algorithm that achieves better trade-off between stability and plasticity. In addition, we also consider other *minor* desiderata that may not be satisfied simultaneously and can be relaxed in certain applications.

1. **Online learning** Samples in the continuum should arrive sequentially in small mini-batches, and revisiting old samples is not allowed except those stored in a memory buffer.

2. **Task-agnostic continual learning / no oracle at test time** In certain settings or model's components[1], the task identifier is not required to make predictions. In such cases, the continuum only contains the tuple of inputs and outputs $(\boldsymbol{x}_i, y_i)$, making it task-agnostic and the model should not request the task identifier, especially at test time.

3. **Problem agnostic** A continual learning method should be developed for a general problem and not be restricted to a specific dataset.

[1]such as the batch normalization, which we will elaborate in details in Chapter 3

While the major desiderata must all be satisfied, existing works usually make some relaxations to the minor desiderata. For example, *class-incremental* learning methods [82, 187] are often developed in the batch setting where all samples of a task arrive at each step, and the model can train that task for many epochs before moving on to a new one. In this dissertation, we mainly focus on the strictly online setting where both the task and data within each task arrive sequentially. Such a setting is more realistic and ubiquitous in real-world scenarios [132].

1.3 Contribution Summary

This dissertation aims to develop a deeper understanding of continual learning. To this end, we first contribute the understanding of classical continual learning algorithms by analyzing how Batch Normalization (BN) affects continual learning, which presents a new perspective of catastrophic forgetting. Then, we develop a *fast-and-slow learning* framework motivated by the *Complementary Learning Systems* theory [14, 62] of human learning. We then realize this framework to several continual learning algorithms that are efficient, effective, and applicable to many scenarios.

Figure 1.1 depicts a taxonomy of our contributions to continual learning and deep learning. For clarity, we refer to existing studies as the *Classical framework*, and *Fast-and-slow learning framework* refers to the approach developed in this dissertation. In the following, we highlight the contributions made in this dissertation.

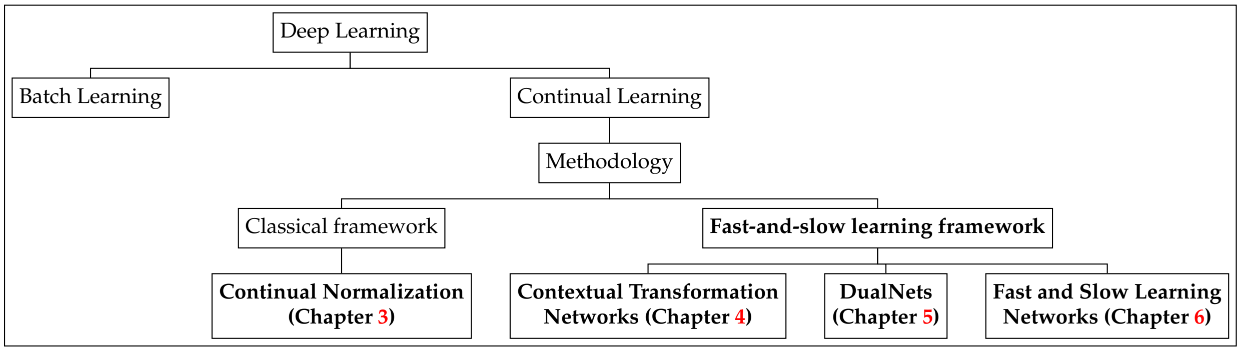

Deep Learning
Batch Learning
Continual Learning
Methodology
Classical framework
Fast-and-slow learning framework
Continual Normalization (Chapter 3)
Contextual Transformation Networks (Chapter 4)
DualNets (Chapter 5)
Fast and Slow Learning Networks (Chapter 6)

FIGURE 1.1: An overview of our dissertation contributions to continual learning, which is highlighted in bold.

- **Continual Batch Normalization** BN has been crucial in successfully training deep neural networks, allowing for faster convergence and better performances. Although recent efforts have been trying to train deep networks without normalization layers in standard batch learning, we argue that BN and its variants are still critical for continual learning. Particularly, we show that BN facilitates training and forward transfer, allowing the networks to achieve higher overall performance compared to a network without any NLs. However, we also discover that BN is also a source of catastrophic forgetting because it performs two different operations during training and testing. Therefore, we propose a novel normalization layer that enjoys the benefit of BN while alleviating its negative effect.

Related Publications

– **Quang Pham**, Chenghao Liu, Steven C.H. Hoi. Continual Normalization: Rethinking Batch Normalization for Online Continual Learning. In International Conference on Learning Representation (ICLR), 2022 [223].

- **Contextual Transformation Networks (CTN)** While static architecture approaches are efficient, they are not as competitive as dynamic architecture methods because they only model the generic features while neglecting task-specific features. We propose to augment the standard backbone in static architecture approaches with a controller component that can efficiently model the task-specific features. The two components are trained in a novel optimization procedure so that the features complement each other, which results in significant performance improvements.

  Related Publications

  - **Quang Pham**, Chenghao Liu, Doyen Sahoo, Steven C.H. Hoi. Contextual Transformation Networks for Online Continual Learning. In International Conference on Learning Representation (ICLR), 2021 [214].

- **Continual Learning: Fast and Slow** CTN has demonstrated that learning both generic and task-specific features in a static backbone network is possible and beneficial. However, we argue that generic features are still prone to forgetting because they are driven to change and accommodate new patterns from incoming tasks. However, slow learning systems are supposed to learn features that are common to all tasks and are less likely to change too drastically when learning new tasks. For this, we propose to train the slow network by minimizing a *self-supervised learning loss* that models the intrinsic properties in the data. Moreover, we also propose a novel task-agnostic adaptation mechanism that transforms the slow, generic features to become instance-specific.

  Related Publications

  - **Quang Pham**, Chenghao Liu, Steven C.H. Hoi. Continual Learning, Fast and Slow. Under review at IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).
  - **Quang Pham**, Chenghao Liu, Steven C.H. Hoi. DualNet: Continual Learning: Fast and Slow. In Conference on Neural Information Processing Systems (NeurIPS), 2021 [215].

- **Learning Fast and Slow for Online Time Series Forecasting.** Time series is ubiquitous in many applications due to the sequential nature of data. We address two major time series forecasting challenges: (i) fast adaptation to concept drifts; and (ii) learning recurring concepts by formulating them as a continual learning problem. We then propose FSNet, which improves upon the fast-and-slow learning principle to address such challenges effectively.

  Related Publications

  - **Quang Pham**, Chenghao Liu, Steven C.H. Hoi. Learning Fast and Slow for Online Time Series Forecasting. Under review [224].

1.4 Dissertation Structure

In Chapter 1, we introduced the continual learning problem and summarized the contributions of this dissertation. Next, we present a comprehensive literature review for continual learning in Chapter 2. Here we formalize the continual learning problem and provide a taxonomy of existing continual learning studies. We also discuss how continual learning relates to other machine learning paradigms. The chapter is followed by our main contributions: Contextual

Transformation Networks in Chapter 4, Fast and Slow Continual Learning in Chapter 5, Learning Fast and Slow for Online Time Series Forecasting in Chapter 6, and Continual Normalization in Chapter 3. Lastly, we discuss promising future research directions in Chapter 7.

Chapter 2

Literature Review

This chapter introduces the continual learning problem and reviews its literature. We mainly focus on the development of continual learning methods and studies analyzing continual learning. We also briefly discuss the Complementary Learning Systems (CLS) theory, which is the main motivation behind our methods. Lastly, we discuss how continual learning relates to other machine learning paradigms to give a broader view of the field.

2.1 Continual Learning

2.1.1 Problem Formulation

We now describe the general problem setting of continual learning [21, 11, 149, 17]. We will discuss the supervised learning setting since it is the main focus of this dissertation. Let $\mathcal{X}_t \subset \mathbb{R}^D$ and $\mathcal{Y}_t \subset \mathbb{N}$ be the t-th input feature and label spaces respectively. The t-th task is defined as a joint distribution $P_t(X, Y) = P(X_t, Y_t)$ over $\mathcal{X}_t \times \mathcal{Y}_t$, and is simply denoted as $\mathcal{T}_t$. Moreover, we

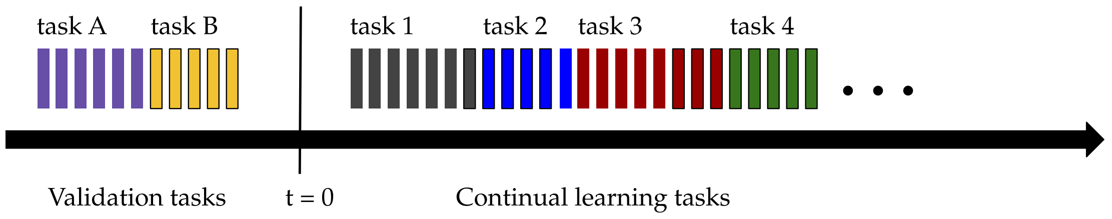


FIGURE 2.1: The continual learning setting considered in this dissertation. Task A and B are used for hyper-parameters cross-validation and can be revisited multiple times. Task 1, 2, etc. are used for continual learning. Both the continual learning tasks their data arrive sequentially.

will use the term “task" and “problem" interchangeably throughout this dissertation. In continual learning, a continuum is defined as a training data stream $\mathcal{D}^{tr} = \{\boldsymbol{x}_i, t_i, y_i\}_{i=1}^{\infty}$ where each training sample is a tuple formed by an input feature $\boldsymbol{x}_i \in \mathcal{X}_{t_i}$, a task descriptor t_i (optional), and a target label $y_i \in \mathcal{Y}_{t_i}$. The task descriptor describes the problem of learning y from $\boldsymbol{x}$, and can take different forms such as a set of semantic attributes of objects in the task [29], or simply an index of the task [80], which we use in this formulation. While observing the data samples in $\mathcal{D}^{tr}$ in an online manner, our goal is to learn a predictor parameterized by $\boldsymbol{\omega} \in \mathbb{R}^N$ as $f_{\boldsymbol{\omega}} : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{Y}$ such that the predictor can correctly predict y from $\boldsymbol{x}$ on all observed tasks, i.e. $f_{\omega}(\boldsymbol{x}) = y, \forall t' \leq t, (\boldsymbol{x}, y) \sim P_{t'}(X, Y)$. It is important to note that since this dissertation focuses on deep neural networks, $\boldsymbol{\omega}$ is a vectorized representation of the model parameter where each parameter is reshaped into a vector and then concatenated altogether. Moreover, $\boldsymbol{\omega}$ refers to the *total parameter* of the model and can include different components. For example, in Chapter 4, the total parameter includes the base model $\boldsymbol{\phi}$, the controller $\boldsymbol{\theta}$, and a set of classifiers $\{\boldsymbol{\varphi}_i\}_{i=1}^{T}$, making $\boldsymbol{\omega} = \{\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_T\}$.

Most existing studies in literature focus on the *locally i.i.d* continuum (locally independent, identically distributed continuum), in which every triplet

$(\boldsymbol{x}_i, t_i, y_i)$ satisfies $(\boldsymbol{x}_i, y_i) \overset{iid}{\sim} P_{t_i}(X, Y)$. Continuums with correlated samples are not yet widely studied, and is beyond the scope of this dissertation. In the literature, there exist several different continual learning protocols. Here we categorize them based on two questions:

i) Is the task identifier t_i given during training and testing? And

ii) Is the training within each task performed online?

For question (i), when we do not know which task the sample belongs to, the protocol is called "task-free" and there is a shared classifier (last layer) for all tasks [126]. Otherwise, the protocol is called "task-aware" where the task identifier is given and only the corresponding classifier is selected during inference. In question (ii), data of a task can either be fully available when task changes or can arrives sequentially. When all the task data is available, training within tasks can be done in an offline fashion with multiple epochs through the data. On the other hand, when samples of each task arrives sequentially, the training of each task is performed online in only one epoch and the whole continuum becomes a data stream. The terms "online continual learning" and "batch continual learning" used in this dissertation refer to how the sample of each task are presented during training. Empirically, the online continual learning is more challenging than its offline counterpart due to the difficulty of optimizing deep neural network online. In addition, the task-aware evaluation often has better performances than the task-free evaluation due to the additional information from the task identifiers.

Lastly, hyperparameter cross-validation is an important problem in continual learning, regardless of the protocol considered. Particularly, we must not use data of future tasks when searching for the hyperparameter. Here we follow [129] and assume that we have access to a small amount of tasks prior to continual learning. Such tasks will not be encountered again during actual continual learning and only be used for cross-validation. Figure 2.1 demonstrates the continual learning protocol considered throughout this dissertation where task A and task B are provided prior to continual learning for cross-validation and can be revisited many times. The actual continual learning process starts at $t = 0$ with tasks and their training samples arrive sequentially in an online fashion.

2.1.2 Evaluation Metrics

To measure the model performance, we adopt four standard metrics: Average Accuracy ACC(↑) [80], Backward Transfer BWT(↑) [80], Forgetting Measure FM(↓) [129], and Learning Accuracy LA(↑) [156]. Denote $a_{i,j}$ as the model's accuracy evaluated on the test set $\mathcal{D}_j^{te}$ after it has been trained on the most recent sample of task $\mathcal{T}_i$. Then, the above metrics are defined as:

- **Average Accuracy (higher is better):** the average accuracy of all observed tasks:
$$\mathrm{ACC}(\uparrow) = \frac{1}{T}\sum_{i=1}^{T} a_{T,i}.$$
- **Forgetting Measure (lower is better):** the average forgetting compared to each task's best performance in history:
$$\mathrm{FM}(\downarrow) = \frac{1}{T-1}\sum_{j=1}^{T-1} \max_{l\in\{1,\dots T-1\}} a_{l,j} - a_{T,j}.$$

- **Backward Transfer BWT(↑) (higher is better):** the average of changes in each task at the end compared to when it is learned:

$$\text{BWT}(\uparrow) = \frac{1}{T-1}\sum_{j=1}^{T-1} a_{T,j} - a_{j,j},$$

- **Learning Accuracy (higher is better):** measures the performance of a model on a task right after it finishes training that task:

$$\text{LA}(\uparrow) = \frac{1}{T}\sum_{i=1}^{T} a_{i,i}.$$

The Averaged Accuracy (ACC(↑)) measures the model's overall performance across all tasks and is a common metric to compare among different methods. Backward Transfer (BWT(↑)) and Forgetting Measure (FM(↓)) measure the model's forgetting as the averaged performance changes of old tasks. Finally, Learning Accuracy (LA(↑)) measures the model's ability to acquire new knowledge. Note that in the task free setting, the task identifiers are *not* given to the model at any time and are only used to measure the evaluation metrics.

2.2 Continual Learning Studies

2.2.1 Overview

Pioneer works in [9, 21, 15] discovered that binding new patterns to a neural network would overwrite the learned ones and named this the catastrophic forgetting phenomenon. Therefore, neural networks tend to be overly plastic, giving them a great ability to learn many patterns at a cost of struggling to

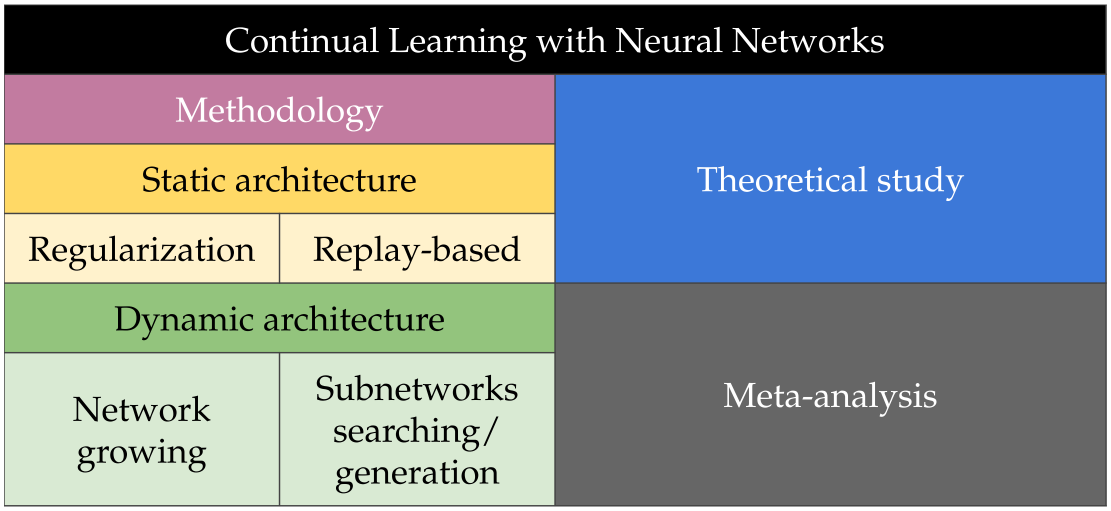


FIGURE 2.2: A diagram illustrating the taxonomy of existing continual learning studies, white text indicates a category of study.

remember when learning sequentially. With the recent developments of powerful computational devices such as GPU, and the availability of large datasets to train deep neural networks, continual learning and catastrophic forgetting have received increased interests. [41] is one of the first study to revisit continual learning with modern neural networks architecture and demonstrate the catastrophic forgetting effects when learning two tasks. Since then, several theoretical works have generalization bounds [208, 167] to better understand continual learning via the PAC-Bayes framework [48, 64] or set theory [184]. Moreover, early continual learning methods addressed the catastrophic forgetting problem by constraining the network's internal representation to be more local rather than globally distributed [40, 42]. However, such methods only found successful in a sequence of two tasks while struggling to generalize to longer sequences.

Recent efforts have been focused on studying continual learning on more challenging problems and longer sequence of tasks (usually more than 10). Existing methods can be broadly categorized into *two* major approaches based on the network parameter extracting the features of each task. Note that this categorization is based on the backbone network (feature extractor), which does not include the classifiers, since each task may have its own classifier. A first family is the *static architecture* methods where a single backbone network is shared for all tasks. Such methods often require an episodic memory to store important information of old tasks, which is later interleaved with the training of newer tasks. The episodic memory can contain a subset of old samples (replayed-based methods), or the importance of each parameters to previous tasks (regularization methods). The second family is the *dynamic architecture* approach where a (partially) different backbone network is used for each task. The subnetworks can be either identified from a bigger network, or progressively grown as new tasks arrive. Concurrently, continual learning is also actively studied in neuroscience, which develops better understanding of continual learning in human. One notable approach is the CLS theory [14, 62], which we will also briefly discuss in Section 2.2.5. Figure 2.2 illustrate a taxonomy of existing continual learning studies with neural networks. In the following, we describe the continual learning methods and meta-analysis studies in more details while leaving to each chapter the description of its closely related works.

2.2.2 Static Architecture Approach

Regularization Methods Regularization methods prevent catastrophic forgetting to previous tasks via a regularizer added to the main lost function. Generally, regularization methods optimize the following equation at task $\mathcal{T}_t$:

$$\mathcal{L}_t(\boldsymbol{\omega}) = \tilde{\mathcal{L}}_t(\boldsymbol{\omega}) + \lambda \mathcal{R}_t(\boldsymbol{\omega}), \tag{2.1}$$

where $\tilde{\mathcal{L}}_t(\cdot)$ denotes the loss of task $\mathcal{T}_t$ such as the cross-entropy loss, λ is the balancing factor, $\mathcal{R}_t(\boldsymbol{\omega})$ is the regularizer for task $\mathcal{T}_t$, and $\mathcal{L}(\cdot)$ denotes the regularized loss[1]. Early works [23, 77] designed the regularizer $\mathcal{R}_t$ to be the knowledge distillation loss [50] using the previous model and current data, which serves as a proxy to control the changes of new models' outputs on old data. This approach and its variants [61] have shown promising results and even integrated into a bigger pipeline [201, 103]. However, [118] shows that this regularizer only works when tasks are highly similar and its performance can quickly deteriorate when tasks differ slightly.

A more rigorous approach to Equation 2.1 is following the Bayesian view of continual learning [104, 109, 99, 133] where the posterior of previous tasks serves as a prior for the current task, which provides a natural recursive formulation to solve continual learning. This approach can be implemented via variational inference on feed-forward networks [109], or achieved via the following regularizers:

$$\mathcal{R}_t(\boldsymbol{\omega}) = \sum_i (\boldsymbol{\omega}_i - \boldsymbol{\omega}^*_{i,t-1})^\intercal H(\boldsymbol{\omega}^*_{t-1})(\boldsymbol{\omega}_i - \boldsymbol{\omega}^*_{i,t-1}) \text{ or} \tag{2.2}$$

$$\mathcal{R}_t(\boldsymbol{\omega}) = \sum_{t'<t} \sum_i (\boldsymbol{\omega}_i - \boldsymbol{\omega}^*_{i,t'})^\intercal H(\boldsymbol{\omega}^*_{t'})(\boldsymbol{\omega}_i - \boldsymbol{\omega}^*_{i,t'}), \tag{2.3}$$

where $\boldsymbol{\omega}_i$ is the i-th parameter, $\boldsymbol{\omega}^*_{i,t'}$ is the optimal i-th parameter of task $\mathcal{T}_{t'}$, and $H(\boldsymbol{\omega}^*_{t'})$ is the importance measure of the parameter $\boldsymbol{\omega}^*_{t'}$ to task t'. In general, with a model $\boldsymbol{\omega} \in \mathbb{R}^N$, the importance measurement $H(\boldsymbol{\omega}^*_{t'})$ describes the importance of each parameter in $\boldsymbol{\omega}$ to the task $\mathcal{T}_{t'}$ relative to other parameters, which requires N^2 entries. In Equation 2.2, the regularizer is derived exactly from the Bayesian rule [104], which only requires to constrain from the immediate preceding task. On the other hand, Equation 2.3 has one regularizer per

[1]for simplicity, we omit the dependence of the loss on the sample in this case

task and over-emphasizes older tasks' importance according to the Bayes rule, which can be useful when prioritizing minimizing catastrophic forgetting.

The important question in such regularization methods is how to estimate the importance of each parameter to a task $\mathcal{T}_{t'}$, which is denoted as $H(\boldsymbol{\omega}^*_{t'})$. In the Bayesian formulation, the parameter importance corresponds to the inverse of the covariate matrix [104], and is obtained by performing the Laplace propagation [20] on the current task's posterior. However, inverting the full covariate matrix is infeasible for deep neural networks. Therefore, many studies have been devoted to develop more efficient and even better estimates of the parameter importance such as approximating the covariate matrix to be diagonal [76, 104] or block diagonal [114]. Other techniques involve directly measure the the contribution of each parameter to the change of the loss function [89, 150], or the model's output [90].

It is worth noting that each regularizer requires storing a snapshot of the model parameter $\boldsymbol{\omega}^*_{t'}$ and their corresponding importance estimates $H(\boldsymbol{\omega}^*_{t'})$. As a result, the cost of storing one regularizer is twice the number of parameters in the model, which can be expensive for deep neural networks.

Replay-based Methods Different from regularization methods, replay-based methods store a small amount of old data and interleave them with the training of new tasks. The simplest replay form is *experience replay* [12], also called *rehearsal* [15], which simply re-trains the old samples while learning the newer ones. Advances rehearsal techniques involve using nearest neighbor classifier [82], additional regularizers using soft labels [120, 170, 175], training schemes to improve knowledge transfer [156]. Since repeatedly training on a small amount of data in the memory can be prone to overfitting, [80, 124] proposed to use the memory data to constraint the model to not increase the losses on those data while training new tasks. [129] proposed a more efficient version of [80] by considering the averaged loss, which significantly speeds up the training time. Interestingly, experience replay is a quite general strategies that also found successful in other application domains such as Reinforcement Learning, which uses experience replay to decorrelate the observations, smoothing changes in the distribution, or alleviate catastrophic forgetting [54, 69, 157].

In applications where storing the original data is not possible (e.g. due to privacy concerns), *pseudo-rehearsal* trains another model to generate synthetic data for experience replay. With the recent success of generative models such as Generative Adversarial Networks [45], generating synthetic, high-dimensional data such as high-resolution images become possible and has shown promising results in continual learning [84, 179, 171, 148]. However, this approach requires incorporating training a generative model such as GANs continually, which is extremely challenging since training GANs itself can be modeled as a continual learning problem [198] and may result in a circular reasoning fallacy. Another approach is to directly optimize the synthetic samples such that they maximize performance on a separate memory units [187].

2.2.3 Dynamic Architecture Approach

Different from the static architecture approach, dynamic architecture methods use a partially separate backbone network to extract the feature of each task. In an extreme case, a separate network is allocated for each task [66], which completely avoids forgetting since the task-specific networks are frozen upon learning the new tasks. However, this comes at a cost of linear growth in the number of model parameters, which may not be applicable in practice. Another extreme is starting from a very large network and identifying a configuration of subnetworks that is suitable for each task. The subnetworks can be identified by performing search [72, 153].

However, a majority of dynamic architecture methods lie between the above extreme, with a partial parameter sharing among all subnetworks and a mechanism to generate the task specific parameters. The task-specific networks can be accumulated into a single set of parameter via superposition [131, 131], generated from another network [194], masked [117], or gradually grown from the original network [123, 137, 122, 143]. Note that such methods do not completely eliminate forgetting since there is still a partially parameter sharing across tasks. However, empirically, they still demonstrate a strong performance with very little forgetting.

Lastly, dynamic architecture approaches are often expensive to train in practice due to the cost of constantly expanding the network of searching for a subnetwork configuration. Therefore, they often show competitive performance in the offline continual learning while struggling in the online setting [129].

2.2.4 Meta-Analysis of Continual Learning

Meta-analysis of continual learning plays an important role in broadening our understanding of continual learning. Such studies do not heavily focus on developing a novel continual learning method but they focus on analyzing how different components of the models and the continuum affect the overall performance. As a result, they provide deeper insights to existing methods and the catastrophic forgetting phenomenon.

The first type of meta analysis in continual learning considers the continuum's properties and how they affect the overall performances. Particularly, they study the task order [161], tasks' difficulties, and the degree dissimilarity among the tasks [147]. Such studies provided insights the design of future continual learning benchmarks.

The second meta analysis explores the relationship between continual learning and other learning paradigms. [192] studied the local optima of continual learning and multitask learning and discovered that they were connected by a linear path of low error in the parameter space. [139] studied the relationship between meta learning and continual learning and unified them in a hierarchical Bayes framework.

Since computer vision and Reinforcement Learning (RL) are the more popular domains to study continual learning, the third type of meta analysis explores the catastrophic forgetting beyond vision and RL benchmarks and convolutional neural networks backbone. Specifically, continual learning and catastrophic forgetting have been explored in recurrent neural networks [158, 209], NLP applications [146, 159, 168], and recommendation systems [190, 200].

The last type of meta-analysis explores how catastrophic forgetting correlates to different training elements. [197, 113] conducted a comprehensive study of catastrophic forgetting in different layers of the backbone network and how it is alleviated by existing methods. [193] instead studied how different training parameters such as dropout, learning rate decay, etc., effect the training in continual learning. In this dissertation, Chapter 3 is also a meta analysis study where we study the effect of batch normalization to continual learning.

Meta-analysis studies have shown to broaden our understanding of continual learning and demonstrated continual learning in various AI problems.

They also provided a deeper understanding of the catastrophic forgetting phenomenon, showing that it happened from various factors, ranging from distributed representation property to different training hyper-parameter settings.

2.2.5 Complementary Learning Systems

Although continual learning has shown to be an extremely difficult task for neural networks, humans is a natural continual learner. Therefore, extensive efforts in neuroscience have been devoted to study the human brain to understand its mechanism supporting continual learning. Among which, the CLS theory [36, 62] is one of the most prominent study because of its effectiveness and empirical support.

According to the CLS theory, there are two components in the human brain interacting with one another to facilitate learning over time. First, the *hippocampus* focuses on the *fast* acquisition of the specific experiences. Such experiences are pattern-specific and can be easily overwritten when the hippocampus accumulate newer experiences. Therefore, there is a need for a second component, called the *neocortex* to support long-term knowledge retention. Via the memory consolidation process, which could happen during sleeping, the hippocampus' pattern-specific experiences are transferred to the neocortex to form a general representation that can last for a long time, which is referred to as the *slow* learning. Consequently, the hippocampus supports the learning ability in human while the neocortex supports the ability to remember and generalize to novel experience. The two components always interact with each other to facilitate the learning and remembering ability in human. This hierarchical design of the brain is also supported in other studies in neuroscience [30]. Figure. 2.3 illustrates the fast-and-slow learning paradigm induced by the CLS theory.

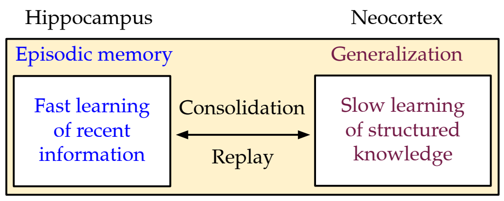

Hippocampus
Neocortex
Episodic memory
Generalization
Fast learning of recent information
Consolidation
Replay
Slow learning of structured knowledge

FIGURE 2.3: An illustration of the fast and slow learning according to the CLS theory. Figure is inspired from [149].

Most methods described in Section 2.2.2 and 2.2.3 only employ a single learning component. We refer to these methods as the *classical framework*. On the other hand, the CLS theory instead suggest successful continual learning in human is supported by *two* learning components. Therefore, we adopt this idea to develop a fast-and-slow learning framework as a novel approach to continual learning. Particularly, the fast-and-slow learning framework models the hippocampus and neocortex by deploying two systems that is responsible for learning different patterns. This principle is the core of our algorithms proposed in Chapter 4, 5, and 6.

2.3 Relation to Other Machine Learning Paradigms

Continual learning is closely related to other machine learning paradigms, which have been concurrently studied and developed in other fields. In the following, we will discuss each paradigm briefly, and compare them to continual learning.

Multi-task learning Multi-task learning aims at simultaneously training a model on a set of tasks such that the overall performances are maximized. The tasks are pre-defined and all data samples are provided prior to training. Some tasks can be positive or negative correlated to one another. Therefore the model should leverage the positive transfer among the positive correlated tasks while alleviating the negative transfers from the negative correlated ones. In literature, given the same backbone, the model trained with multi-task learning is often referred to as a soft upper bound of the continual learning model.

Transfer Learning Transfer learning aims at improving the learning outcomes of a given target task by levering the knowledge from a set of related tasks, which are call the source tasks. Therefore, transfer learning mostly focus on facilitating positive forward transfer, which is one of the continual learning's goal. However, transfer learning studies often do not focus on the performance on the source tasks and only transfer knowledge to a single target task. In contrast, continual learning aims at maximizing the performances on all tasks. Moreover, the target task in continual learning always changes with the learned tasks at this step become the source task for the incoming ones.

Domain Adaptation Domain adaptation is one subfield of transfer learning where the source and target tasks share the same input and label spaces $\mathcal{X}, \mathcal{Y}$, but each task is a different domain on the joint distribution[2]. For example, the source domains can be images of digits in black and while while the target domain is those digits in colored backgrounds. Similar to transfer learning, domain adaptation is unidirectional and only focus on one target domain.

Online Convex Optimization Online Convex Optimization (OCO), concerns the learning of a linear model over a stream of data samples belonging to a single task. Compared to OCO, continual learning contains multiple tasks, which makes the sampling procedure in the whole continuum non-i.d.d. Moreover,

[2]Recall that a task is defined as a distribution $P(X, Y)$.

online learning aimed minimized the regret on the training data[3] while continual learning aims at maximizing the generalization to the test data.

Meta Learning (Few-shot learning) In literature, there exist several slightly different definitions for meta learning. The definition we use here is based on the work of [22, 172] which studies the “few-shot learning problem". Meta learning aims at training a model that can quickly learn to solve new tasks by utilizing its *internal* knowledge obtained from solving many prior problems. Therefore, meta learning mainly concerns about facilitating positive forward knowledge transfer while neglecting the ability to retain obtained knowledge.

Lastly, meta learning and continual learning settings can be complementary to each other. For example, meta continual learning [136, 138] and continual meta learning [101] studies the intersection of both fields by applying meta learning algorithms to continual learning or designing meta learning algorithms without forgetting. However, such problems are beyond the scope of this dissertation, which focuses on the traditional continual learning setting.

2.4 Summary

This chapter presented the continual learning problem and a brief survey of the related studies. In particular, we first gave a formal definition of continual learning and introduced several benchmarks with the evaluation metrics. We then briefly discussed the literature, including existing approaches and meta analyses of continual learning. This chapter concluded with an overview of different machine learning paradigms and how continual learning fitted to this bigger picture.

[3]regret is defined as the difference between the model’s cumulative loss versus the best model in hindsight.

Chapter 3

Continual Normalization

We first look at the classical continual learning framework of experience replay and study how common training components affect continual learning. Particularly, we are interested in Normalization Layers (NLs) as it is a crucial component for the success of deep models on image datasets. We consider the online continual learning setting with image benchmarks to analyze the benefits and drawbacks of different NLs to continual learning. From there, we propose the desiderata of an ideal NL for continual learning and propose Continual Normalization (CN), a NL that satisfies such criteria. Extensive experiments demonstrate the strength and weaknesses of different NLs and how CN can be a more suitable NL for online continual learning with images.

3.1 Introduction

While most previous works focus on developing strategies to alleviate catastrophic forgetting and facilitating knowledge transfer [149], scant attention has been paid to the backbone they used. In standard backbone networks such as ResNets [60], it is natural to use Batch Normalization (BN) [51], which has enabled the deep learning community to make substantial progress in many applications [182]. Although recent efforts have shown promising results in training deep networks without BN in a single task learning [204], we argue that BN has a huge impact on continual learning. Particularly, when using an episodic memory, BN improves knowledge sharing across tasks by allowing data of previous tasks to contribute to the normalization of current samples and vice versa. Unfortunately, in this work, we have explored a negative effect of BN that hinders its performance on older tasks and thus increases catastrophic forgetting. Explicitly, unlike the standard classification problems, data in continual learning arrived sequentially, which are *not independent and identically distributed* (*non-i.i.d*). Moreover, especially in the online setting [80], the data distribution changes over time, which is also highly non-stationary. Together, such properties make the BN's running statistics heavily biased towards the current task. Consequently, during inference, the model normalizes previous tasks' data using the moments of the current task, which we refer to as the "cross-task normalization effect".

Although using an episodic memory can alleviate cross-task normalization in BN, it is not possible to fully mitigate this effect without storing all past data, which is against the continual learning purposes. On the other hand, spatial normalization layers such as GN [121] do not perform cross-task normalization because they normalize each feature individually along the spatial dimensions. As we will empirically verify in Section 3.5.2, although such layers suffer less forgetting than BN, they do not learn individual tasks as well as BN because they lack the knowledge transfer mechanism via normalizing along the mini-batch dimension. This result suggests that a continual learning normalization layer should balance normalizing along the mini-batch and spatial dimensions to achieve a good trade-off between knowledge transfer and alleviating forgetting. Such properties are crucial for continual learning, especially in the online setting [79, 156], but not satisfied by existing normalization layers. Consequently, we propose Continual Normalization (CN), a novel normalization layer that takes into account the spatial information when normalizing along the mini-batch dimension. Therefore, our CN enjoys the benefits of BN while alleviating the cross-task normalization effect. Lastly, our study is orthogonal to the continual learning literature, and the proposed CN can be easily integrated into any existing methods to improve their performances across different online protocols.

In summary, we make the following contributions. First, we study the benefits of BN and its cross-task normalization effect in continual learning. Then, we identify the desiderata of a normalization layer for continual learning and propose CN, a novel normalization layer that improves the performance of existing continual learning methods. Lastly, we conduct extensive experiments to validate the benefits and drawbacks of BN, as well as the improvements of CN over BN.

3.2 Notations and Preliminaries

This section provides the necessary background of continual learning and normalization layers.

3.2.1 Notations

We focus on the image recognition problem and denote an image as $\boldsymbol{x} \in \mathbb{R}^{W \times H \times C}$, where W, H, C are the image width, height, and the number of channels respectively. A convolutional neural network (CNN) with parameter $\boldsymbol{\theta}$ is denoted as $f_{\boldsymbol{\theta}}(\cdot)$. A feature map of a mini-batch B is defined as $\boldsymbol{a} \in \mathbb{R}^{B \times C \times W \times H}$. Finally.

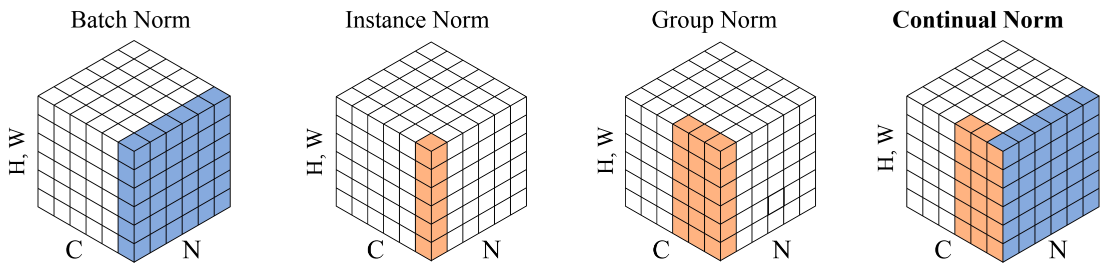


FIGURE 3.1: An illustration of different normalization methods using cube diagram derived from [121]. Each cube represents a feature map tensor with N as the batch axis, C as the channel axis, and (H,W) as the channel axes. Pixels in blue are normalized by the same moments calculated from different samples while pixels in orange are normalized by the same moments calculated from within sample. Best viewed in colors.

each image $\boldsymbol{x}$ is associated with a label $y \in \{1, 2 \ldots, Y\}$ and a task identifier $t \in \{1, 2, \ldots, T\}$, that is optionally revealed to the model, depending on the protocol.

3.2.2 Normalization Layers

Normalization layers are essential in training deep neural networks [182]. Existing works have demonstrated that having a specific normalization layer tailored towards each learning problem is beneficial. However, the continual learning community stills mostly adopt the standard BN in their backbone network, and lack a systematic study regarding normalization layers [188]. To the best of our knowledge, this is the first work proposing a dedicated continual learning normalization layer.

In general, a normalization layer takes a mini-batch of feature maps $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_B)$ as input and perform the Z-normalization as:

$$\boldsymbol{a}' = \gamma \left(\frac{\boldsymbol{a} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \right) + \boldsymbol{\beta}, \tag{3.1}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are the mean and variance calculated from the features in B, and ϵ is a small constant added to avoid division by zero. The affine transformation's parameters $\gamma, \beta \in \mathbb{R}^C$ are $|C|$-dimensional vectors learned by backpropagation to retain the layer's representation capacity. For brevity, we will use "moments" to refer to both the mean $\boldsymbol{\mu}$ and the variance $\boldsymbol{\sigma}^2$.

It is important to note that the Z-normalization step in Eq.(3.1) also depends on the network's parameter $\boldsymbol{\theta}$ at the time of performing the normalization. As a result, the input's distributions of each layer continuously change during training, results in the internal covariate shift phenomenon [51, 165]. To alleviate the effect, BN proposes to not backpropagate through the moments calculation during the backward pass, which has been subsequently adopted in later works.

Batch Normalization BN is one of the first normalization layers that found success in a wide range of deep learning applications [51, 116, 95]. During training, BN calculates the moments across the mini-batch dimension as:

$$\boldsymbol{\mu}_{BN} = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{w=1}^{W}\sum_{H=1}^{H}\boldsymbol{a}_{bcwh}, \quad \boldsymbol{\sigma}^2_{BN} = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{w=1}^{W}\sum_{H=1}^{H}(\boldsymbol{a}_{bcwh} - \boldsymbol{\mu}_{BN})^2. \tag{3.2}$$

At test time, it is important for BN to be able to make predictions with only one data sample to make the prediction deterministic. As a result, BN replaces the mini-batch mean and variance in Eq. (3.2) by an estimate of the global values obtained during training as:

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta(\boldsymbol{\mu}_B - \boldsymbol{\mu}), \quad \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} + \eta(\boldsymbol{\sigma}_B - \boldsymbol{\sigma}), \tag{3.3}$$

where η is the exponential running average's momentum, which were set to 0.1.

Spatial Normalization Layers The discrepancy between training and testing in BN can be problematic when the training mini-batch size is small [74], or when the testing data distribution differs from the training distributions. In such scenarios, the running estimate of the mean and variance obtained during training are a poor estimate of the moments to normalize the testing data. Therefore, there have been tremendous efforts in developing alternatives to BN to address these challenges. One notable approach is normalizing along the spatial dimensions of each sample independently, which completely avoid the negative effect from having a biased global moments. In the following, we discuss popular examples of such layers.

Layer Normalization (LN) [56] Layer Normalization [56] was proposed to address the discrepancy between training and testing in BN by normalization along the spatial dimension of the input feature. Particularly, LN computes the moments to normalize the activation as:

$$\begin{aligned} \mu_{LN} &= \frac{1}{CWH} \sum_{c=1}^{C} \sum_{w=1}^{W} \sum_{h=1}^{H} \boldsymbol{a}_{bcwh} \\ \sigma^2_{LN} &= \frac{1}{CWH} \sum_{c=1}^{C} \sum_{w=1}^{W} \sum_{h=1}^{H} (\boldsymbol{a}_{bcwh} - \mu_{LN})^2 \end{aligned} \tag{3.4}$$

Instance Normalization (IN) [67] Similar to LN, Instance Normalization [67] also normalizes along the spatial dimension. However, IN normalizes each channel separately instead of jointly as in LN. Specifically, IN computes the moments to normalize as:

$$\begin{aligned}\mu_{IN} =& \frac{1}{WH}\sum_{w=1}^{W}\sum_{h=1}^{H}\boldsymbol{a}_{bcwh}\\ \sigma^2_{IN} =& \frac{1}{WH}\sum_{w=1}^{W}\sum_{h=1}^{H}(\boldsymbol{a}_{bcwh} - \mu_{LN})^2\end{aligned} \tag{3.5}$$

Group Normalization (GN) [121] GN proposes to normalize each feature individually by dividing each channel into groups and then normalize each group separately. Moreover, GN does not normalize along the batch dimension, thus performs the same computation during training and testing. Computationally, GN first divides the channels C into G groups as:

$$\boldsymbol{a}'_{bgkhw} \leftarrow \boldsymbol{a}_{bchw}, \text{ where } k = \lfloor * \rfloor \frac{C}{G}, \tag{3.6}$$

Then, for each group g, the features are normalized with the moments calculated as:

$$\mu^{(g)}_{GN} = \frac{1}{m}\sum_{k=1}^{K}\sum_{w=1}^{W}\sum_{h=1}^{H}\boldsymbol{a}'_{bgkhw}, \quad {\sigma^{(g)}_{GN}}^2 = \frac{1}{m}\sum_{k=1}^{K}\sum_{w=1}^{W}\sum_{h=1}^{H}(\boldsymbol{a}'_{bgkhw} - \mu^{(g)}_{GN})^2 \tag{3.7}$$

GN has shown comparable performance to BN with large mini-batch sizes (e.g. 32 or more), while significantly outperformed BN with small mini-batch sizes (e.g. one or two). Notably, when putting all channels into a single group (setting $G = 1$), GN is equivalent to Layer Normalization (**LN**) [56], which normalizes the whole layer of each sample. On the other extreme, when separating each channel as a group (setting $G = C$), GN becomes Instance Normalization (**IN**) [67], which normalizes the spatial dimension in each channel of each feature. Figure 3.1 provides an illustration of BN, GN, IN, and the proposed CN, which we will discuss in Section 3.4.

3.2.3 Switchable Normalization (SN)

While LN and IN addressed some limitations of BN, their success are quite limited to certain applications. For example, LR is suitable for recurrent neural networks and language applications [56, 160], while IN is widely used in image stylization and image generation [67, 85]. Beyond their dedicated applications, BN still performs superior to LR and IN. Therefore, SN [108] is designed to take advantage of both LN and IN while enjoying BN's strong performance. Let $\Omega =$ {BN,LN,IN} , SN combines the moments of all three normalization layers and performs the following normalization:

$$\boldsymbol{a}' = \gamma \left(\frac{\boldsymbol{a} - \sum_{k \in \Omega} \boldsymbol{w} \boldsymbol{\mu}_k}{\sqrt{\boldsymbol{w}' \boldsymbol{\sigma}_k^2 + \epsilon}} \right) + \boldsymbol{\beta}, \tag{3.8}$$

where $\boldsymbol{w}$ and $\boldsymbol{w}'$ are the learned blending factors to combine the three moments and are learned by backpropagation.

3.3 Batch Normalization in Continual Learning

3.3.1 Normalization Layers Benefit Continual Learning

We argue that BN is helpful for forward transfer in two aspects. First, BN makes the optimization landscape smoother [116], which allows the optimization of deep neural networks to converge faster and better [95]. In continual learning, BN enables the model to learn individual tasks better than the no-normalization method. Second, with the episodic memory, BN uses data of the current task and previous tasks (in the memory) to update its running moments. Therefore, BN further facilitates forward knowledge transfer during experience replay: current task data is normalized using moments calculated

Method	ACC	FM	LA	$\Delta_{\mu}^{(1)}$	$\Delta_{\mu}^{(2)}$	$\Delta_{\sigma^2}^{(1)}$	$\Delta_{\sigma^2}^{(2)}$
Single-BN	72.81	18.65	87.74	10.85	0.91	3.69	7.74
Single-BN*	**75.92**	**15.13**	**88.24**				
ER-BN	80.66	9.34	88.23	3.56	0.46	1.41	3.16
ER-BN*	**81.75**	**8.51**	**88.46**				

TABLE 3.1: Evaluation metrics on the pMNIST benchmark. The magnitude of the differences are calculated as $\Delta_{\omega}^{(k)} = ||\omega_{BN}^{(k)} - \omega_{BN^*}^{(k)}||_1$, where $\omega \in \{\mu, \sigma^2\}$ and $k \in \{1, 2\}$ denotes the first or second BN layer. Bold indicates the best scores.

from both current and previous samples. Compared to BN, spatial normalization layers (such as GN) lack the ability to facilitate forward transfer via normalizing using moments calculated from samples of both old and current tasks. We will empirically verify the benefits of different normalization layers to continual learning in Section 3.5.2.

3.3.2 The Cross-Task Normalization Effect

Recall that BN maintains a running estimate of the global moments to normalize the testing data. In the standard learning with a single task where training samples are i.i.d, one can expect such estimates can well-characterize the true, global moments, and can be used to normalize the testing data. However, online continual learning data is non-i.i.d and highly non stationary. Therefore, BN's estimation of the global moments is heavily biased towards the current task because the recent mini-batches only contain that task's data. As a result, during inference, when evaluating the older tasks, BN normalizes previous tasks' data using the current task's moments, which we refer to as the **cross-task normalization effect**.

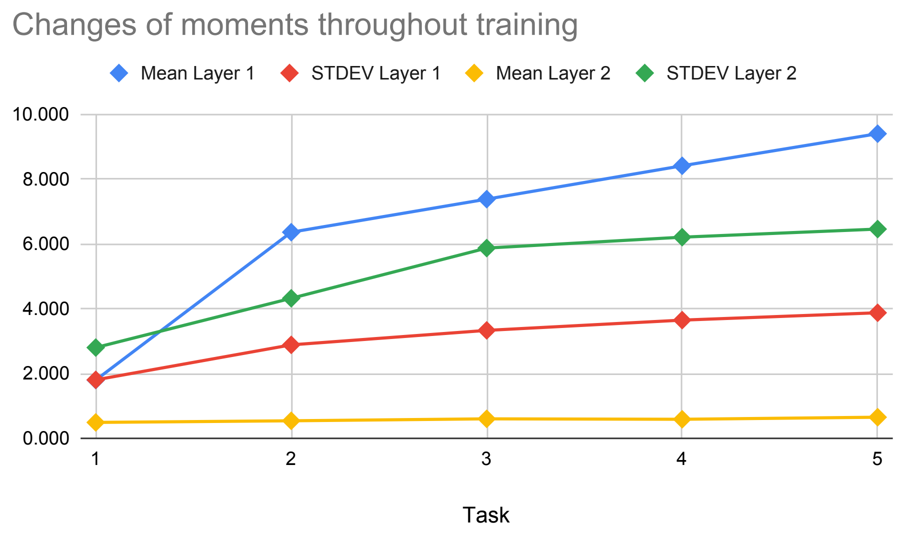


FIGURE 3.2: Changes of moments (L1 distance - y axis) between BN and BN* throughout training on a 2-layers MLP backbone. Standard deviation values are omitted due to small values (e.g. < 0.1)

We consider a toy experiment on the permuted MNIST (pMNIST) benchmark [80] to explore the cross-task normalization effect in BN. We construct a sequence of five task, each has 2,000 training and 10,000 testing samples. We implement a multilayer perceptron (MLP) backbone configured as Input(784)-FC1(100)-BN1(100)-FC2(100)-BN(100)-Softmax(10), where the number inside the parentheses indicates the output dimension of that layer. We consider the Single and ER strategies for this experiment, where the Single strategy is the naive method that trains continuously without any memory or regularization. For each method, we implement an optimal-at-test-time BN variant (denoted by the suffix -BN*) that calculates the global moments using **all training data of all tasks** before testing. Compared to BN, BN* has the same parameters, training procedure and only differs in the moments used to normalize the testing data. We emphasize that although BN* is unrealistic, it sheds light on how cross-task normalization affects the performance of CL algorithms. We report the averaged accuracy at the end of training ACC(↑) [79], forgetting measure FM(↓) [96], and learning accuracy LA(↑) [156] in this experiment.

Table 3.1 reports the evaluation metrics at the end of training. Besides the standard metrics, we also report the moments' difference magnitudes between the standard BN and the global BN variant. We also plot the changing rate of each moment through out learning in Figure 3.2. We observe that the gap between two BN variants is significant without the episodic memory. When using the episodic memory, ER can reduce the gap because the pMNIST is quite simple and even the ER strategy can achieve close performances to the offline model [130]. Moreover, it is important to note that training with BN on imbalanced data also affects the model's learning dynamics, resulting in a suboptimal parameter. Nevertheless, having an unbiased estimate of the global moments can greatly improve overall accuracy given the same model parameters. Moreover, this improvement is attributed to reducing forgetting rather than facilitating transfer: BN* has lower FM but almost similar LA compared to the traditional BN. In Table 3.1 and Figure 3.2, we observe that except the mean of the second layer, the other moments quickly diverge from the optimal BN* as the model observes more tasks. Moreover, the discrepancy in the first layer can cascade through the network depth and result different deeper features. This result agrees with recent finding [197] that deeper layers are more responsible for causing forgetting because their hidden representation deviates from the model trained on all tasks. Overall, these results show that normalizing older tasks using the current task's moments causes higher forgetting, which we refer to as the "cross-task normalization effect".

3.3.3 Desiderata for Continual Learning Normalization Layers

While BN facilitates continual learning, it also amplifies catastrophic forgetting by causing the cross-task normalization effect. To retain the BN's benefits while alleviating its drawbacks, we argue that an ideal normalization layer for continual learning should be adaptive by incorporating each feature's statistics into its normalization. Being adaptive can mitigate the cross-task normalization effect because each sample is now normalized differently at test time instead of being normalized by a set of biased moments. In literature, adaptive normalization layers have shown promising results when the mini-batch sizes are small [121], or when the number of training data is extremely limited [169]. In such cases, normalizing along the spatial dimensions of each feature can alleviate the negative effect from an inaccurate estimate of the global moments. Inspired by this observation, we propose the desiderata for a continual learning normalization layer:

- Facilitates the performance of deep networks by improving knowledge sharing within and across-task (when the episodic memory is used), thus increasing the performance of all tasks, e.g. ACC($\uparrow$) ;
- Is adaptive at test time: each data sample should be normalized differently. Moreover, each data sample should contribute to its normalized feature's statistic, thus reducing catastrophic forgetting;
- Does not require additional input at test time such as the episodic memory, or the task identifier.

To simultaneously facilitate training and mitigating the cross-task normalization effect, a normalization layer has to balance between both across mini-batch normalization and within-sample normalization. As we will show in

Section 3.5.2, BN can facilitate training by normalizing along the mini-batch dimension; however, it is not adaptive at test time and suffers from the cross-task normalization effect. On the other hand, GN is fully adaptive, but it does not facilitate training compared to BN. Therefore, it is imperative to balance both aspects to improve continual learning performance. Lastly, we expect a continual learning normalization layer can be a direct replacement for BN, thus, it should not require additional information to work, especially at test time.

3.4 Continual Normalization (CN)

CN works by first performing a spatial normalization on the feature map, which we choose to be group normalization. Then, the group-normalized features are further normalized by a batch normalization layer. Formally, given the input feature map $\boldsymbol{a}$, we denote $\mathrm{BN}_{1,0}$ and $\mathrm{GN}_{1,0}$ as the batch normalization and group normalization layers without the affine transformation parameters[1], CN obtains the normalized features $\boldsymbol{a}_{\mathrm{CN}}$ as:

$$\boldsymbol{a}_{\mathrm{GN}} \leftarrow \mathrm{GN}_{1,0}(\boldsymbol{a}); \quad \boldsymbol{a}_{\mathrm{CN}} \leftarrow \gamma \mathrm{BN}_{1,0}(\boldsymbol{a}_{\mathrm{GN}}) + \boldsymbol{\beta}. \tag{3.9}$$

The GN component does not use the affine transformation to make the intermediate feature $\mathrm{BN}_{1,0}(\boldsymbol{a}_{\mathrm{GN}})$ well-normalized across the mini-batch and spatial dimensions. Moreover, performing GN first allows the spatial-normalized features to contribute to the BN's running statistic, which further reduce the cross-task normalization effect. An illustration of CN is given in Figure 3.1.

By its design, one can see that CN satisfies the desiderata of a continual learning normalization layer. Particularly, CN balances between facilitating

[1]which is equivalent to setting $\gamma = 1$ and $\boldsymbol{\beta} = 0$

training and alleviating the cross-task normalizing effect by normalizing the input feature across mini-batch and individually. Therefore, CN is adaptive at test time and produces well-normalized features in the mini-batch and spatial dimensions, which strikes a great balance between BN and other instance-based normalizers. Lastly, CN uses the same input as conventional normalization layers and does not introduce extra learnable parameters that can be prone to catastrophic forgetting.

We now discuss why CN is a more suitable normalization layer for continual learning than recent advanced normalization layers. SwitchNorm [108] proposed to combine the moments from three normalization layers: BN, IN, and LN to normalize the feature map (Eq. 3.8). However, the blending weights $\boldsymbol{w}$ and $\boldsymbol{w}'$ make the output feature not well-normalized in any dimensions since such weights are smaller than one, which scale down the moments. Moreover, the choice of IN and LN may not be helpful for image recognition problems. Similar to SN, TaskNorm [169] combines the moments from BN and IN by a blending factor, which is learned for each task. As a result, TaskNorm also suffers in that its outputs are not well-normalized. Moreover, TaskNorm addresses the meta learning problem and requires knowing the task identifier at test time, which violates our third criterion of requiring additional information compared to BN and our CN.

3.5 Experiment

We evaluate the proposed CN's performance compared to a suite of normalization layers with a focus on the online continual learning settings where it is more challenging to obtain a good global moments for BN. Our goal is to evaluate the following hypotheses: (i) BN can facilitate knowledge transfer better

TABLE 3.2: Dataset summary

	#task	img size	#training imgs	#testing imgs	#classes	Section
pMNIST	5	28×28	10,000	50,000	10	Section 3.3.2
Split CIFAR10	5	3×32×32	50,000	10,000	10	Section 3.5.3
Split CIFAR100	20	3×84×84	50,000	10,000	100	Section 3.5.2
Split Mni IMN	20	3×84×84	50,000	10,000	100	Section 3.5.2
Split Tiny IMN	10	3×64×64	100,000	10,000	200	Section 3.5.3
COCOseq	4	3×224×224	35,072	6,346	70	Section 3.5.4
NUS-WIDEseq	6	3×224×224	48,724	2,367	49	Section 3.5.4

than spatial normalization layers such as GN and SN (ii) Spatial normalization layers have lower forgetting than BN; (iii) CN can improve over other normalization layers by reducing catastrophic forgetting and facilitating knowledge transfer; (iv) CN is a direct replacement of BN without additional parameters and minimal computational overhead.

3.5.1 Dataset Summary

Table 3.2 summaries the datasets used throughout our experiments. In the following, we also summarize the key settings in our experiments.

Toy pMNIST Benchmark In the toy pMNIST benchmark, we construct a sequence of five task by applying a random but fixed permutation on the original MNIST [19] dataset to create a task. Each task consists of 1,000 training samples and 10,000 testing samples as the original MNIST dataset. In the pre-processing step, we normalize the pixel value by dividing its value by 255.0. Both the Single and Experience Replay (ER) methods were trained using the Stochastic Gradient Descent (SGD) optimizer with mini-batch size 10 over one epoch with learning rate 0.03. This benchmark follows the "single-head" setting [96], thus we only maintain a single classifier for all tasks and task-identifiers are not provided in both training and testing.

Online Task-IL Continual Learning Benchmarks (Split CIFAR100 and Split Mini IMN) In the data pre-processing step, we normalize the pixel density to $[0-1]$ range and resize the image size to $3 \times 32 \times 32$ and $3 \times 84 \times 84$ for the Split CIFAR100 and Split minIMN benchmarks, respectively. No other data pre-processing steps are performed. All methods are optimized by SGD with mini-batch size 10 over one epoch. The episodic memory is implemented as a Ring buffer [129] with 50 samples per task.

Online Task-IL and Class-IL Continual Learning Benchmarks (Split CIFAR-10 and Split Tiny IMN) We follow the protocol as the DER++ work [170] except the number of epochs, which we set to one. Particularly, training is performed with data augmentation on both the data stream and the memory samples (random crops, horizontal flips). We follow the configurations provided by the authors, including the hyper-parameters of mini-batch size, the reservoir sampling memory management strategy, learning rates, etc. For the memory size of 2560, which were not conducted in the original paper, we use the same setting as the case of 5120 memory slots.

Online Task-Free Long-Tailed Continual Learning (COCOseq and NUS-WIDEseq) We follow the same settings as the original work [183] in our experiments. Particularly, we train each method using the Adam optimizer [47] with default hyper-parameters ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-4$), mini-batch size 10 over one epoch. The episodic memory is implemented as a reservoir buffer using the PRS [183] strategy with total 2,000 slots.

3.5.2 Online Task-incremental Continual Learning

Setting We first consider the standard online, task-incremental continual learning setting [80] on the Split CIFAR-100 and Split Mini IMN benchmarks. We follow the standard setting in [129] to split the original CIFAR100 [28] or Mini

IMN [68] datasets into a sequence of 20 tasks, three of which are used for hyper-parameter cross-validation, and the remaining 17 tasks are used for continual learning.

We consider two popular continual learning strategies: (i) Experience Replay (ER) [130]; and (ii) Dark Experience Replay++ (DER++) [170]. Besides the vanilla ER, we also consider DER++, a recent improved ER variants, to demonstrate that our proposed CN can work well across different ER-based strategies. All methods use a standard ResNet 18 backbone [60] (not pre-trained) and are optimized over one epoch with batch size 10 using the SGD optimizer. For each continual learning strategies, we compare our proposed CN with five competing normalization layers: (i) BN [51]; (ii) Batch Renormalization (BRN) [74]; (iii) IN [67]; (iv) GN [121]; and (v) SN [108]. We cross-validate and set the number of groups to be $G = 32$ for our CN and GN in this experiment.

Results Table 3.3 reports the evaluation metrics of different normalization layers on the Split CIFAR-100 and Split Mini IMN benchmarks. We consider the averaged accuracy at the end of training ACC($\uparrow$) [79], forgetting measure FM($\downarrow$) [96], and learning accuracy LA($\uparrow$) [156] as discussed in Chapter 2. Clearly, IN does not perform well because it is not designed for image recognition problems. Compared to adaptive normalization methods such as GN and SN, BN suffers from more catastrophic forgetting (higher FM($\downarrow$)) but at the same time can transfer knowledge better across tasks (higher LA($\uparrow$)). Moreover, BRN performs worse than BN since it does not address the biased estimate of the global moments, which makes normalizing with the global moments during training ineffective. Overall, the results show that although traditional adaptive methods such as GN and SN do not suffer from the cross-task normalization effect and enjoy lower FM($\downarrow$) values, they lack the ability to facilitate knowledge

TABLE 3.3: Evaluation metrics of different normalization layers on the Split CIFAR100 and Split Mini IMN benchmarks. Bold indicates the best averaged scores, † suffix indicates non-adaptive method

ER	Split CIFAR100			Split Mini IMN		
Norm. Layer	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
NoNL	55.87±0.46	4.46±0.48	57.26±0.64	47.40±2.80	3.17±0.99	45.31±2.18
BN†	64.97±1.09	9.24±1.98	71.56±0.75	59.09±1.74	8.57±1.52	65.24±0.52
BRN†	63.47±1.33	8.43±1.03	69.83±2.52	54.55±2.70	6.66±1.84	58.53±1.88
IN	59.17±0.96	11.47±0.92	69.40±0.93	48.74±1.98	15.28±1.88	62.88±1.13
GN	63.42±0.92	7.39±1.24	68.03±0.19	55.65±2.92	8.31±1.00	59.25±0.72
SN	64.79±0.88	7.92±0.64	71.10±0.51	56.84±1.37	10.11±1.46	64.09±1.53
CN(ours)	**67.48±0.81**	**7.29±1.59**	**74.27±0.36**	**64.28±1.49**	**8.08±1.18**	**70.90±1.16**
DER++	Split CIFAR100			Split Mini IMN		
Norm. Layer	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
NoNL	57.14±0.46	4.46±0.48	57.26±0.64	47.18±3.20	2.77±1.68	45.01±3.35
BN†	66.50±2.52	8.58±2.28	73.78±1.02	61.08±0.91	6.90±0.99	66.10±0.89
BRN†	66.89±1.22	6.98±2.23	73.30±0.08	57.37±1.75	6.66±1.84	66.53±1.56
IN	61.18±0.96	10.59±0.77	71.00±0.57	54.05±1.26	11.82±1.32	65.03±1.69
GN	66.58±0.27	**5.70±0.69**	69.63±1.12	60.50±1.91	**6.17±1.28**	63.10±1.53
SN	67.17±0.23	6.01±0.15	72.13±0.23	57.73±1.97	8.92±1.84	63.87±0.64
CN(ours)	**69.13±0.56**	6.48±0.81	**74.89±0.40**	**66.29±1.11**	6.47±1.46	**71.75±0.68**

transfer across tasks, which results in lower LA(↑) . Moreover, BN can facilitate knowledge sharing across tasks, but it suffers more from forgetting because of the cross-task normalization effect. Across all benchmarks, our CN comes out as a clear winner by achieving the best overall performance (ACC(↑)). This result shows that CN can strike a great balance between reducing catastrophic forgetting and facilitating knowledge transfer to improve continual learning.

Complexity Comparison We study the complexity of different normalization layers and reporting the training time on the Split CIFAR100 benchmark in

TABLE 3.4: Running time of ER on the Split CIFAR100 benchmarks of different normalization layers. Δ% indicates the percentage increases of training time over BN

	BN	GN	SN	CN
Time (s)	1583	1607	2036	1642
Δ%	0%	1.51%	28.61%	3.72%

Tabe 3.4. Both GN and our CN have minimal computational overhead compared to BN, while SN suffers from additional computational cost from calculating and normalizing with different sets of moments.

3.5.3 Online Class-Incremental Continual Learning

Setting We consider the online task incremental (Task-IL) and class incremental (Class-IL) learning problems on the Split-CIFAR10 (Split CIFAR-10) and Split-Tiny-ImageNet (Split Tiny IMN) benchmarks. We follow the same experiment setups as [170] except the number of training epochs, which we set to *one*. All experiments uses the DER++ strategy [170] on a ResNet 18 [60] backbone (not pre-trained) trained with data augmentation using the SGD optimizer. We consider three different total episodic memory sizes of 500, 2560, and 5120, and compare different NLs in this experiment.

Result Table 3.5 reports the ACC and FM metrics on the Split CIFAR-10 and Split Tiny IMN benchmarks under both the Task-IL and Class-IL settings. For CN, we consider two configurations with the number of groups being $G = 8$ and $G = 32$. In most cases, our CN outperforms the standard BN on both metrics, with a more significant gap on larger memory sizes of 2560 and 5120. Interestingly, BN is highly unstable in the Split CIFAR-10, Class-IL benchmark with high standard deviations. In contrast, both CN variants show better and more stable results, especially in the more challenging Class-IL setting with a

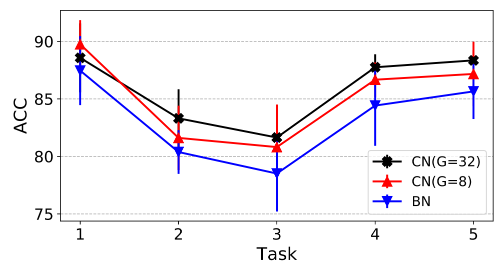
TABLE 3.5: Evaluation metrics of DER++ with different normalization layers on the Split CIFAR-10 and Split Tiny IMN benchmarks. Parentheses indicates the number of groups in CN. Bold indicates the best averaged scores

Buffer	Method	Split CIFAR-10				Split Tiny IMN			
		Class-IL		Task-IL		Class-IL		Task-IL	
		ACC(↑)	FM(↓)	ACC(↑)	FM(↓)	ACC(↑)	FM(↓)	ACC(↑)	FM(↓)
500	BN	48.9±4.5	33.6±5.8	82.6±2.3	3.2±1.9	6.7±0.1	38.1±0.5	40.2±0.9	6.5±1.0
	IN	35.9±2.0	45.0±3.6	78.8±1.6	1.8±0.9	2.2±0.5	**16.9±0.9**	20.6±0.5	**1.7±0.5**
	GN	46.8±5.1	29.7±11.5	82.1±5.2	**1.9±1.2**	7.2±0.4	36.0±2.4	41.1±0.5	4.5±1.8
	SN	42.3±3.9	32.3±11.6	80.6±3.4	2.6±2.0	4.5±1.0	25.0±2.4	33.6±1.3	2.5±1.9
	CN (G=8)	48.9±0.3	**27.9±5.0**	84.7±0.5	2.2±0.3	**7.3±0.9**	37.5±2.3	**42.2±2.1**	4.9±2.2
	CN (G=32)	**51.7±1.9**	28.2±4.0	**86.2±2.2**	2.0±1.3	6.5±0.7	40.1±1.7	40.6±1.4	6.8±1.9
2560	BN	52.3±4.6	29.7±6.1	86.6±1.6	0.9±0.7	11.2±2.3	36.0±2.0	50.8±1.7	2.8±1.2
	IN	36.1±2.8	37.8±8.8	79.1±1.1	0.8±1.4	2.4±0.4	**19.2±1.5**	25.4±1.7	**0.6±0.5**
	GN	53.3±3.5	28.8±3.2	87.3±2.4	**0.6±0.7**	11.0±1.3	35.5±1.2	50.9±3.0	1.9±1.6
	SN	42.3±3.9	30.8±10.4	80.0±6.4	4.7±5.2	4.4±1.4	27.3±2.9	37.6±3.0	3.5±1.8
	CN (G=8)	53.7±3.4	25.5±4.7	87.3±2.7	1.6±1.6	10.7±1.3	38.0±1.2	**52.6±1.2**	1.5±0.5
	CN (G=32)	**57.3±2.0**	**21.6±5.6**	**88.4±1.1**	1.7±1.1	**11.9±0.3**	36.7±1.2	51.5±0.2	2.8±0.9
5120	BN	52.0±7.8	26.7±10.3	85.6±3.3	2.0±1.5	11.2±2.7	36.8±2.0	52.2±1.7	2.6±1.5
	IN	32.2±4.5	41.5±3.6	77.2±3.8	2.1±2.3	2.3±0.6	**18.0±1.3**	25.5±2.0	1.6±0.9
	GN	48.6±2.6	32.2±4.8	86.5±0.8	**0.5±0.5**	9.2±1.8	36.8±1.6	48.0±6.0	4.9±6.3
	SN	42.9±8.9	36.2±8.3	81.1±2.4	3.6±2.4	4.3±0.6	23.3±1.9	37.9±3.0	2.7±1.9
	CN (G=8)	54.1±4.0	24.0±4.0	87.1±2.8	0.7±0.7	**12.2±0.6**	34.6±2.6	53.1±1.8	3.1±1.9
	CN (G=32)	**57.9±4.1**	**22.2±1.0**	**88.3±0.9**	1.3±0.9	**12.2±0.2**	35.6±1.3	**54.9±1.5**	**1.5±1.1**

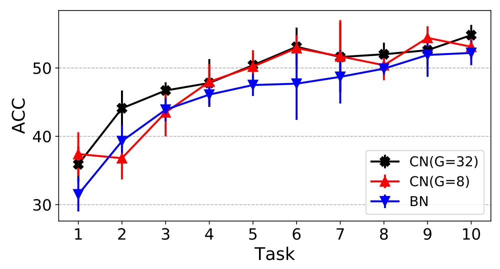
small memory size (indicated by small standard deviation values). We also report the evolution of ACC in Figure 3.3. Both CN variants consistently outperform BN throughout training, with only one exception at the second task on the Split Tiny IMN. CN ($G = 32$) shows more stable and better performances than BN and CN($G = 8$).

3.5.4 Long-tailed Online Continual Learning

We now evaluate the normalization layers on the challenging task-free, long-tailed continual learning setting [183], which is more challenging and realistic since real-world data usually follow long-tailed distributions. We consider the PRS strategy [183] and the COCOseq and NUS-WIDEseq benchmarks, which consists of four and six tasks, respectively. Unlike the previous benchmarks,



(A) Split CIFAR-10, Task-IL



(B) Split Tiny IMN, Task-IL

FIGURE 3.3: The evolution of ACC(↑) on observed tasks so far on the Split CIFAR-10 and Split Tiny IMN benchmarks, Task-IL screnario with DER++ and memory size of 5120 samples.

TABLE 3.6: Evaluation metrics of the PRS strategy on the COCOseq and NUS-WIDEseq benchmarks. We report the mean performance over five runs. Bold indicates the best averaged scores

COCOseq	Majority			Moderate			Minority			Overall		
	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP
BN	64.2	58.3	66.2	51.2	48.1	**55.7**	31.6	31.4	38.1	51.8	48.6	54.9
CN(ours)	**64.8**	**58.5**	**66.8**	**52.5**	**49.2**	**55.7**	**35.7**	**35.5**	**38.4**	**53.1**	**49.8**	**55.1**
NUS-WIDEseq	Majority			Moderate			Minority			Overall		
	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP
BN	24.3	16.1	21.2	16.2	16.5	20.9	**28.5**	**28.2**	**32.3**	23.4	20.9	25.7
CN(ours)	**25.0**	**17.2**	**22.7**	**17.1**	**17.4**	**21.5**	27.3	27.0	31.0	**23.5**	**21.3**	**25.9**

images in the COCOseq and NUS-WIDEseq benchmarks can have multiple labels, resulting in a long-tailed distribution over each task's image label. Following [183], we report the average overall F1 (O-F1), per-class F1 (C-F1), and the mean average precision (mAP) at the end of training and their corresponding forgetting measures (FM). We also report each metric over the minority classes (<200 samples), moderate classes (200-900 samples), majority classes (>900 samples), and all classes. Empirically, we found that smaller groups helped in the long-tailed setting because the moments were calculated over more channels, reducing the dominants of head classes. Therefore, we use

TABLE 3.7: Forgetting measure (FM(↓)) of each metric from the PRS strategy on the COCOseq and NUS-WIDEseq benchmarks, lower is better. We report the mean performance over five runs. Bold indicates the best averaged scores

COCOseq	Majority			Moderate			Minority			Overall		
	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP
BN	**23.5**	**22.8**	8.4	30.0	30.2	9.4	36.2	35.7	13.2	29.7	29.5	9.7
CN(ours)	**23.5**	23.1	**6.5**	**26.9**	**26.9**	**7.4**	**26.8**	**26.5**	**12.2**	**25.6**	**25.7**	**8.0**
NUS-WIDEseq	**Majority**			**Moderate**			**Minority**			**Overall**		
	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP	C-F1	O-F1	mAP
BN	54.6	50.7	12.5	62.2	61.9	15.5	52.5	52.4	12.2	57.6	55.7	11.2
CN(ours)	**52.7**	**48.7**	**8.6**	**58.8**	**58.0**	**10.3**	**51.2**	**50.6**	**11.6**	**57.4**	**55.5**	**8.1**

$G = 4$ groups in this experiment.

We replicate PRS with BN to compare with our CN and report the results in Table 3.6 and Table 3.7. We observe consistent improvements over BN from only changing the normalization layers, especially in reducing FM(↓) across all classes.

3.5.5 Discussion of The Results

Our experiments have shown promising results for CN being a potential replacement for BN in online continual learning. While the results are generally consistent, there are a few scenarios where CN does not perform as good as other baselines. First, from the task-incremental experiment in Table 3.3, DER++ with CN achieved lower FM compared to GN. The reason could be from the DER++'s soft-label loss, which together with GN, overemphasizes on reducing FM and achieved lower FM. On the other hand, CN has to balance between reducing FM and improving LA. Second, training with data augmentation in the online setting could induce high variations across different runs. Table 3.5 shows that most methods have high standard deviations on the Split CIFAR-10 benchmark, especially with small memory sizes. In such scenarios, there could be insignificant differences between the first and second best methods. Also, on the NUS-WIDEseq benchmark, CN has lower evaluation metrics on minority classes than BN. One possible reason is the noisy nature of the NUS-WIDE dataset, including the background diversity and huge number of labels per image, which could highly impact the tail classes' performance. Lastly, CN introduces an additional hyper-parameter (number of groups), which needs to be cross-validated for optimal performance.

3.6 Conclusion

In this Chapter, we investigated the potentials and limitations of NLs in online continual learning with images. We showed that while BN can facilitate knowledge transfer by normalizing along the mini-batch dimension, the cross-task normalization effect hinders older tasks' performance and increases catastrophic forgetting. This limitation motivated us to propose CN, a novel normalization layer especially designed for online continual learning settings. Our extensive experiments corroborate our findings and demonstrate the efficacy of CN over other normalization strategies. Particularly, we showed that CN is a plug-in replacement for BN and can offer significant improvements on different evaluation metrics across different online settings with minimal computational overhead.

Chapter 4

Contextual Transformation Networks

Chapter 3 studied BN within the classical continual learning framework. We have seen that classical methods are inefficient to achieve continual learning. This requires a new framework to facilitate continual learning in deep networks. Therefore, we now shift the focus to the development of the fast-and-slow learning framework and derive efficient, effective continual learning algorithms. From Chapter 2, recall that there are two major continual learning approaches of static and dynamic architectures. On the one hand, fixed architectures rely on a single network to learn models that can perform well on all tasks. As a result, they often only accommodate common features of those tasks but neglect each task's specific features. On the other hand, dynamic architecture methods can have a separate network for each task, but they are too expensive to train and not scalable in practice, especially in the online setting. In this chapter, we aim to bridge the gap between these two major approaches by developing a strategy that enjoys the simplicity of static architectures methods and the strong performances of the dynamic architectures ones.

4.1 Introduction

In the literature, fixed architecture methods employ a shared feature extractor and a set of classifiers, one for each task [80, 129, 130, 125]. Although using a shared feature extractor has achieved promising results, the common and global features are rather generic and not well-tailored towards each specific task. This problem is even more severe when old data is limited while learning new tasks. As a result, the common feature extractor loses its ability to extract previous tasks' features, resulting in catastrophic forgetting. On the other hand, while dynamic architecture methods such as [66, 143, 122] alleviate this problem by having a separate network for each task, they suffer from the unbounded growth of the parameters. Moreover, the subnetworks' design is not trivial and requires extensive resource usage [66, 143], which is not practical in many applications. These limitations motivated us to develop a novel method that can facilitate continual learning with a fixed architecture by modeling the task-specific features.

To achieve this goal, we first revisit a popular result in learning multiple tasks that each task's features are centered around a common vector [26, 32, 48, 145]. This result motivates us to develop a novel framework of Contextual Transformation Networks (CTN), which consists of a base network that learns the common features of a given input and a controller that efficiently transforms the common features to become task-specific, given a task identifier. While one can train CTN using experience replay, it does not explicitly aim at achieving a good trade-off between stability and plasticity. Therefore, we propose a novel dual memory system and a learning method that encapsulate alleviating forgetting and facilitating knowledge transfer simultaneously.

Particularly, we propose two distinct memories: the *episodic memory* and the *semantic memory* associated with the base model and the controller, respectively. Then, the base model is trained by experience replay on the episodic memory while the controllers is trained to learn task-specific features that can generalize to the semantic memory. As a result, CTN achieves a good trade-off between preventing catastrophic forgetting and facilitating knowledge transfer because the task-specific features can generalize well to all past and current tasks. Figure 4.1 gives an overview of the proposed Contextual Transformation Network (CTN).

Interestingly, the designs of our CTN and dual memory are partially related to the Complementary Learning Systems (CLS) theory in neuroscience [14, 62]. Particularly, the controller acts as a *neocortex* that learns the structured knowledge of each task. In contrast, the base model acts as a *hippocampus* that performs rapid learning to acquire new information from the current task's training data. Following the naming convention of memory in neuroscience, our CTN is equipped with two replay memory types. (i) the **episodic memory** (associated with the hippocampus) caches a small amount of past tasks' training data, which will be replayed when training the base networks. (ii) the **semantic memory** (associated with the neocortex) stores another distinct set of old data only used to train the controller such that the task-specific features can generalize well across tasks. Moreover, the CLS theory also suggests that the interplay between the neocortex and the hippocampus attributes to the ability to recall knowledge and generalize to novel experiences [36]. Our proposed learning approach closely characterizes such properties: the base model focuses on acquiring new knowledge from the current task while the controller uses the base model's knowledge to generalize to novel samples.

In summary, this chapter makes the following contributions. First, we propose CTN, a novel continual learning method that can model task-specific features while enjoying neglectable complexity overhead compared to fixed architecture methods (please refer to Table 4.4). Second, we propose a novel objective that can improve the trade-off between alleviating forgetting and facilitating knowledge transfer to train CTN. Third, we conduct extensive experiments on continual learning benchmarks to demonstrate the efficacy of CTN compared to a suite of baselines. Finally, we provide a comprehensive analysis to investigate the complementarity of each CTN's component.

4.2 Method

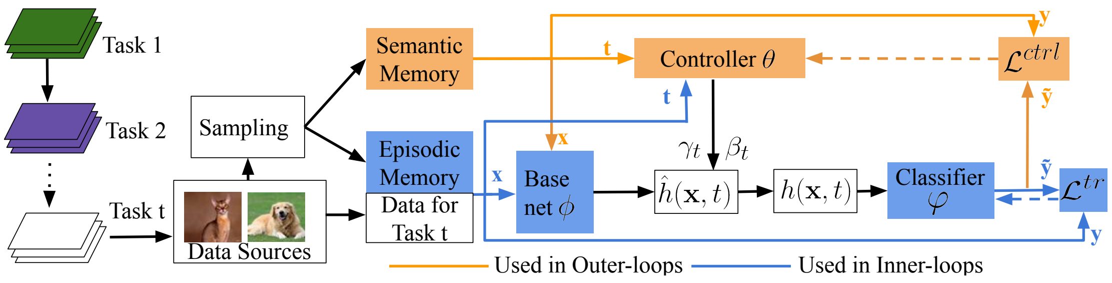


FIGURE 4.1: Overview of the Contextual Transformation Networks (CTN). CTN consists of a controller $\boldsymbol{\theta}$ that modifies the features of the base model ϕ. The base model is trained using experience replay on the episodic memory while the controller is trained to generalize to the semantic memory, which addresses both alleviating forgetting and facilitating knowledge transfer. Best viewed in colors.

Notations. We denote ϕ as parameter of the base model that extracts global features from the input and $\boldsymbol{\theta}$ as the parameter of the controller which modifies the features from ϕ given a task identifier t. The task identifier can be a set of semantic attributes about objects of that task [29] or simply an index of the task, which we use in this chapter as a one-hot vector. A prediction is given as $g_{\boldsymbol{\varphi}_t}(h_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x},t))$, where $g_{\boldsymbol{\varphi}_t}(\cdot)$ is the task $\mathcal{T}_t$'s classifier with parameter $\boldsymbol{\varphi}_t$ such as a fully connected layer with softmax activation, and $h_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x},t)$ is the final feature after transformed by the controller. We denote D_t^{tr} as the training data of task $\mathcal{T}_t$, $\mathcal{M}_t^{em}$ and $\mathcal{M}_t^{sm}$ as the *episodic memory* and the *semantic memory* of task $\mathcal{T}_t$ respectively. The episodic memory and semantic memory maintains two distinct sets of data obtained from task $\mathcal{T}_t$. Our CTN employs an episodic memory to store a small amount of training samples of each task to support continual learning, which is a common practice for experience replay strategies [130]. The episodic memory of task $\mathcal{T}_1,\ldots,\mathcal{T}_{t-1}$ is denoted as $\mathcal{M}_{<t}^{em}$; similarly, $\mathcal{M}_{<t}^{sm}$ denotes the semantic memory of the first $t-1$ tasks.

Remark. Both the $\mathcal{M}_t^{em}$ and $\mathcal{M}_t^{sm}$ are obtained from D_t^{tr} through the learner's internal memory management strategy and contains distinct samples from each other such that their combined sizes do not exceed a pre-defined budget.

4.2.1 Learning Task-Specific Features for Continual Learning

Given a backbone network, one can implement the task-specific features by employing a set of task-specific filters and applying them to the backbone's output. However, this trivial approach is not scalable, even for small networks. In the worst case, it results in storing an additional network per task, which violates the fixed architecture constraint. Since we want to obtain task-specific features with minimal parameter overhead, we propose to use a feature-wise transformation [112] to efficiently extract the task-specific features $\tilde{h}(\boldsymbol{x},t)$ from the common features $\hat{h}(\boldsymbol{x})$ as follows:

$$\tilde{h}(\boldsymbol{x};t) = \frac{\gamma_t}{\|\gamma_t\|_2} \otimes \hat{h}(\boldsymbol{x}) + \frac{\beta_t}{\|\beta_t\|_2} \text{ and } \{\gamma_t,\beta_t\} = c_{\boldsymbol{\theta}}(t), \tag{4.1}$$

where $\otimes$ denotes the element-wise multiplication operator, and $c_{\boldsymbol{\theta}}(t)$ is the controller implemented as a linear layer with parameter $\boldsymbol{\theta}$ that predicts the transformation coefficients $\{\boldsymbol{\gamma}_t, \beta_t\}$ given the task identifier t. Since the task identifiers are one-hot vectors, which are sparse and make training the controller difficult, we also introduce an embedding layer to map the task identifiers to dense, low dimensional vectors. For simplicity, we will use $\boldsymbol{\theta}$ to refer to both the embedding and the linear layer parameters. In addition, instead of storing a set of coefficients $\{\gamma_t, \beta_t\}$ for each task, we only need a fixed set of parameter $\boldsymbol{\theta}$ to predict these coefficients, which results in the fixed parameters in the controller. The coefficients $\{\boldsymbol{\gamma}_t, \beta_t\}$ are ℓ_2-normalized and then transforms the common features $\hat{h}(\boldsymbol{x}, t)$ to become task-specific features $\tilde{h}(\boldsymbol{x}; t)$. Finally, both feature types are combined by a residual connection before passing to the corresponding classifier $g_t(\cdot)$ to make the final prediction:

$$g_{\boldsymbol{\varphi}_t}(\sigma(h(\boldsymbol{x}, t))), \text{ and } h(\boldsymbol{x}, t) = \hat{h}(\boldsymbol{x}, t) + \tilde{h}(\boldsymbol{x}, t), \tag{4.2}$$

where $\sigma(\cdot)$ is a nonlinear activation function such as ReLU. Importantly, when the task-specific features are removed, i.e., $\tilde{h}(\boldsymbol{x}, t) = 0$, Equation 4.2 reduces to the traditional experience replay. Lastly, for each incoming task, CTN has to allocate a new classifier, which is the same for all continual learning methods, and a new embedding vector, which is usually low dimensional, e.g. 32 or 64. Therefore, CTN enjoys almost the same parameter growth as existing continual learning methods with a static backbone network.

4.2.2 Training the Controller

While one can train CTN with experience replay (ER), it does not explicitly address the trade-off between facilitating knowledge transfer and alleviating catastrophic forgetting. This motivates us to develop a novel training method that can simultaneously address both problems by leveraging the controller's task-specific features. First, we introduce a dual memory system consisting of the semantic memory $\mathcal{M}_t^{sm}$ associated with the controller and the episodic memory $\mathcal{M}_t^{em}$ associated with the base model. We propose to train only the base model using experience replay with the episodic memory to obtain new knowledge from incoming tasks. The controller is also trained so that the task-specific features can generalize to unseen samples to the base model stored in the semantic memory. As a result, the task-specific features can generalize to both previous and current tasks, which simultaneously encapsulate both alleviating forgetting and facilitating knowledge transfer. Formally, given the current batch of data for task $\mathcal{T}_t$ as $\mathcal{B}_t$, the training of CTN can be formulated as the following bilevel optimization problem [27]:

$$\begin{aligned} \text{Outer problem:} \quad & \min\nolimits_{\boldsymbol{\theta}} \mathcal{L}^{ctrl}(\{\boldsymbol{\phi}^*, \boldsymbol{\theta}\}; \mathcal{M}^{sm}_{<t+1}) \\ \text{Inner problem:} \quad & \text{s.t} \quad \boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathcal{L}_{tr}(\{\boldsymbol{\phi}, \boldsymbol{\theta}\}, \mathcal{B}_t \cup \mathcal{M}^{em}_{<t}), \end{aligned} \tag{4.3}$$

where $\boldsymbol{\phi}^*$ denotes the optimal base model corresponding to the current controller $\boldsymbol{\theta}$. Since every CTN's prediction always involves both the controller and the base model, we use $\mathcal{L}_{tr}(\{\boldsymbol{\phi}, \boldsymbol{\theta}\}, \mathcal{B}_t \cup \mathcal{M}^{em}_{<t})$ to denote the training loss of the pair $\{\boldsymbol{\phi}, \boldsymbol{\theta}\}$ on the data $\mathcal{B}_t \cup \mathcal{M}^{em}_{<t}$. Similarly, $L^{ctrl}(\cdot)$ denotes the controller's loss. For simplicity, we omitted the dependency of the the loss on the classifiers' parameters and imply that the classifiers are jointly updated with the base model. Since we do not know the optimal transformation coefficients of any task, the controller is trained to minimize the classification loss of the samples via ϕ. We implement both the training and controller's losses as the cross-entropy loss. Notably, Equation 4.3 characterizes two nested optimization problems:

the outer problem, which trains the controller to generalize, and each controller parameter $\boldsymbol{\theta}$ parameterizes an inner problem that trains the base model to acquire new knowledge via experience replay. Moreover, only $\boldsymbol{\phi}$ is trained in the inner problem, while only $\boldsymbol{\theta}$ is updated in the outer problem.

The bilevel optimization objective such as Equation 4.3 has been successfully applied in other machine learning disciplines such as hyperparameter optimization, meta learning [100, 73], and AutoML [144]. In this chapter, we extend this framework to continual learning to train the controller. However, unlike existing works [100, 73, 144], our Equation 4.3 has to be solved incrementally when a new data sample arrives. Therefore, we consider Equation 4.3 as an online learning problem and optimize it using the *follow the leader* principle [2]. Particularly, we relax the optimal solutions of both the inner and outer problems to be solutions from a few gradient steps. When a new training sample arrives, we first train the base model $\boldsymbol{\phi}$ using experience replay for a few SGD steps with an inner learning rate α, each of which is implemented as:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha \nabla_{\boldsymbol{\phi}} \mathcal{L}_{tr}(\{\boldsymbol{\phi}, \boldsymbol{\theta}\}, \mathcal{B}_t \cup \mathcal{M}^{em}_{<t}), \tag{4.4}$$

Then, we optimize the controller $\boldsymbol{\theta}$ such that it can improve $\boldsymbol{\phi}$'s performance on the semantic memory:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}^{ctrl}(\{\boldsymbol{\phi}, \boldsymbol{\theta}\}, \mathcal{M}^{sm}_{<t+1}), \tag{4.5}$$

where β is the outer learning rate. As a result, Equation 4.3 is implemented as an alternative update procedure involving several outer updates to train $\boldsymbol{\theta}$, each of which includes an inner update to train $\boldsymbol{\phi}$. Moreover, performing several updates per incoming sample does not violate the online assumption since we will not revisit that sample in the future, unless it is stored in the memories.

4.2.3 Training the base network

Despite using task-specific features, the base network may still forget previous tasks because of the small episodic memory. To further alleviate catastrophic forgetting in $\boldsymbol{\phi}$, we regularize the training loss $\mathcal{L}_{tr}(\cdot)$ with a behavioral cloning (BC) strategy based on knowledge distillation [50, 120]. Let $\hat{y}$ be the logits of the model's prediction before the softmax layer $\pi(\cdot)$, we regularize the training loss on the episodic memory data in Equation 4.4 as:

$$\mathcal{L}_{tr}(\{\boldsymbol{\phi}, \boldsymbol{\theta}\}, (\boldsymbol{x}, y, k)) = \mathcal{L}(\pi(\hat{y}), y) + \lambda D_{\mathrm{KL}}\left(\pi\left(\frac{\hat{y}}{\tau}\right) \middle\| \pi\left(\frac{\hat{y}_k}{\tau}\right)\right), \tag{4.6}$$

where λ is the trade-off parameter, τ is the softmax's temperature, and $\hat{y}_k$ is a snapshot of the model prediction on the sample $(\boldsymbol{x}, k)$ at the end of task $\mathcal{T}_k$. While the behavioral cloning strategy requires storing $\hat{y}_k$, the memory increase is minimal since $\hat{y}_k$ is a vector with dimension bounded by the total classes, which is much smaller than the image $\boldsymbol{x}$ dimension. Importantly, the behavioural cloning strategy is used to alleviate catastrophic forgetting, which only happens in the base model, not the controller. Particularly, the controller's inputs are task identifiers such as one-hot vectors, which are fully available during learning. In summary, our episodic memory stores the input image $\boldsymbol{x}$, its corresponding label y and the soft label $\hat{y}$, while the semantic memory stores the input-label pair $\boldsymbol{x}, y$.

4.3 Related Work

4.3.1 Feature-wise Transformation

Early works [57, 83] showed that instead of using a task-specific network on the input, one can employ a set of 1×1 filters to extract the task-specific from the common features. However, such approaches still require a *quadratic* complexity overhead in the number of channels, which can be expensive. Another compelling solution is the feature-wise transformation, FiLM [112], which only requires a linear complexity. Thanks to its efficiency, FiLM has been successfully applied in many problems, including meta learning [155, 164], visual reasoning [112], and others fields [98] with remarkable success. Notably, CNAPs [155] proposed an adaptation network to generate the FiLM's parameters and quickly adapt to new tasks. CNAPs has showed promising results when having access to a large amount of tasks to pre-train the common features. However, this setting is different from continual learning where the learner has to obtain new knowledge on the fly. Therefore, CNAPs are principally differs from CTN in that CNAPs assume having access to a well-pretrained knowledge source and uses FiLM to quickly adapt this knowledge to a new task. On the other hand, CTNs use FiLM to accelerate the knowledge acquisition when learning progressively. Lastly, we emphasize that the CTN's design is general. If more budget is allowed, the proposed CTN is readily compatible with the aforementioned feature transformation methods such as [83] by adjusting the controller's output dimension.

4.3.2 Meta Learning

Meta learning [7], also learning to learn, refers to a learning paradigm where an algorithm learns to improve the performance of another algorithm. Our CTN design is related to such learning to learn architectures where the controller is trained to improve the base model's performance. Importantly, we note that there exist other continual learning variants that intersect with meta learning, such as meta-continual learning [138] and continual-meta learning [136, 171]. However, they consider *different* goals and problem settings, such as meta pre-training [138] or rapid recovering the performance at test time given a finetuning step before inference is allowed [136], which is *not* the conventional online continual learning problem [80] we focus in this dissertation.

Meta learning has been an appealing solution to learn a good initialization from a large amount of tasks [73], even in an online manner: Online Meta Learning (OML) [134]. However, we emphasize that OML fundamentally *differs* from our CTN in two aspects. First, OML requires all data of previous tasks and aims to improve the performance of future tasks, which is different from continual learning. Second, OML learns an initialization and requires finetuning at test time, which is not practical, especially when testing on learned tasks. In contrast, CTN is a continual learning method that maximizes the performance of the current task as well as all previous tasks. Moreover, CTN can make a prediction at any time without requiring an additional finetuning step.

TABLE 4.1: Evaluation metrics on continual learning benchmarks considered. All methods use the same backbone network for all benchmarks, episodic memory size is M=50 samples per task

Method	pMNIST			CORe50		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
Finetune	61.66±1.50	20.67±1.64	80.89±0.45	4.38±0.10	49.66±1.14	49.08±1.20
LwF	63.31±3.56	14.29±3.05	75.76±1.43	31.20±0.66	20.44±1.37	49.20±1.10
EWC	67.34±3.00	11.00±2.36	76.59±1.49	31.86±3.90	14.34±3.08	42.98±2.50
GEM	74.84±0.95	8.57±0.33	81.74±0.77	42.56±0.86	7.36±0.90	46.84±2.22
KDR	72.97±0.58	9.20±0.44	81.40±0.41	OOM	OOM	OOM
AGEM	68.67±0.71	13.98±0.68	81.54±0.25	40.28±3.15	11.08±4.01	46.68±1.51
MER	76.59±0.74	6.88±0.59	82.09±0.33	39.28±1.25	9.08±1.25	45.52±0.96
ER-Ring	76.02±0.59	8.57±0.33	83.69±0.44	41.72±1.30	9.10±0.80	48.18±0.81
MIR	76.58±0.10	8.34±0.11	83.57±0.07	43.50±1.92	6.14±0.91	45.98±1.14
CTN (ours)	**79.01±0.65**	**6.69±0.51**	**85.11±0.45**	**54.17±0.85**	**5.50±1.01**	**55.32±0.34**
Independent*	81.05±0.29	0.00	81.05±0.29	53.54±1.10	0.00	53.54±1.10
Offline	84.95±0.95	-	-	89.73±0.91	-	-

Method	Split CIFAR			Split miniIMN		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
Finetune	33.52±3.13	33.88±2.78	65.15±1.18	31.51±2.00	26.00±2.12	55.83±1.42
EWC	39.46±3.75	24.69±3.84	64.54±1.20	32.52±0.53	25.74±2.78	56.39±2.45
ICARL	50.27±0.84	16.55±0.82	65.83±1.53	44.95±0.08	17.59±0.40	61.46±0.50
GEM	57.77±0.86	10.93±1.03	66.45±0.06	55.04±1.88	7.81±1.70	60.13±1.36
KDR	62.75±0.80	5.01±0.79	66.11±0.70	56.89±2.45	4.83±1.23	59.29±1.31
AGEM	58.27±0.86	8.76±0.67	66.12±1.17	51.14±2.16	6.99±1.96	55.11±0.76
MER	61.32±0.86	11.90±0.86	72.51±0.41	57.94±1.08	8.98±0.79	66.11±0.76
ER-Ring	61.36±1.01	7.20±0.72	67.05±1.08	53.43±1.18	11.21±1.35	63.46±1.05
MIR	63.37±1.99	10.53±1.63	73.27±0.77	51.97±1.58	10.37±2.72	60.63±3.43
CTN (ours)	**67.65±0.43**	6.33±0.70	**73.43±0.45**	**65.82±0.59**	**3.02±1.13**	**67.43±1.37**
Independent*	67.21±0.51	0.00	67.21±0.51	65.85±0.98	0.00	65.85±0.98
Offline	74.11±0.66	-	-	71.15±2.95	-	-

4.4 Experiments

4.4.1 Benchmark Datasets and Baselines

We consider four continual learning benchmarks in our experiments. **Permuted MNIST (pMNIST)** [80]: each task is a random but fixed permutation of the original MNIST. We generate 23 tasks with 1,000 images for training and the testing set has the same amount of images as in the original MNIST data. **Split CIFAR-100 (Split CIFAR)** [80] is constructed by splitting the CIFAR100 [28] dataset into 20 tasks, each of which contains 5 different classes sampled without replacement from the total of 100 classes. **Split Mini ImageNet (Split miniIMN)** [129], similarly, we split the miniIMN dataset [68] into 20 disjoint tasks. Finally, we consider the **CORe50** benchmark by constructing a sequence of 10 tasks using the original CORe50 dataset [78].

Throughout the experiments, we compare CTN with a suite of baselines: GEM [80], AGEM [129], MER [156], ER-Ring [130], and MIR [125]. We also consider the *independent* model [80], a dynamic architecture method that maintains a separate network for each task, and each has the **same number of parameters** as other baselines. While the independent model is unrealistic, it is highly competitive and was used as an upper bound of a state-of-the-art dynamic architecture method in [137]. Finally, we include the *Offline* model, which does not follow the continual learning setting and performs multitask training on all tasks' data.

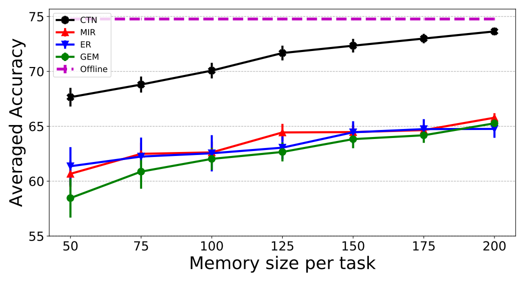
We use a multilayer perceptron with two hidden layers of size 256 for pMNIST, a reduced ResNet18 with three times fewer filters [80] for Split CIFAR and Split miniIMN, and a full ResNet18 on CORE50. Following [80], we use a Ring buffer as the memory structure for all methods and random sampling to select data from memory, including the episodic and semantic memories of CTN.

The exceptions are MER [156], which uses reservoir sampling, and MIR [125], which use their sampling strategies as proposed by the authors. For CTN, the episodic memory and semantic memory are implemented as two Ring buffers with sizes equal to 80% and 20% of the total budget. This configuration is also cross-validated from the validation tasks. For each incoming batch of data, we randomly push 80% samples to the current task's episodic memory and the other 20% are for the current task's semantic memory.

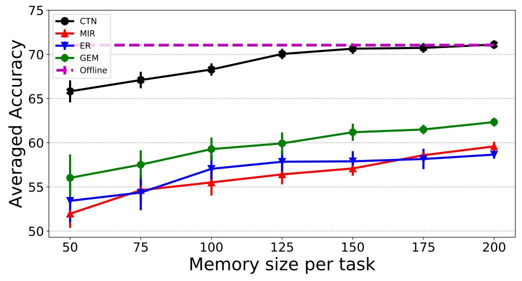
We follow the procedure proposed in [129] to cross-validate all hyperparameters using the first three tasks. Then, the best configuration is selected to perform continual learning on the remaining tasks. During continual learning, the task identifier is given to all methods. We optimize all models using SGD with a mini-batch of size ten over *one* epoch. We run each experiment five times, each has the same task order but different initialization seed, and report the following metrics as discussed in Chapter 2: Averaged Accuracy [80]: ACC($\uparrow$) (higher is better) , Forgetting Measure [96]: FM($\downarrow$) (lower is better), and Learning Accuracy [156]: LA($\uparrow$) (higher is better).

4.4.2 Results of Continual Learning Benchmarks

Table 4.1 reports the evaluation metrics of the models on four continual learning benchmarks considered with 50 samples per task. We observe that CTN is even comparable with the independent method and outperforms other baselines by a large margin. We remind that the independent method has T times more parameters than the remaining methods, where T is the total number of tasks. Moreover, CTN can exploit the relationship across tasks via the task identifiers to improve its performance. For example, learning to classify "man" and "woman" may be helpful to classify "boy" and "girl" because they belong to the same superclass "people". Finally, CTN significantly outperforms the baselines by achieving a better trade-off between alleviating catastrophic forgetting and facilitating knowledge transfer, as shown by lower FM(↓) and higher LA(↑) . Overall, CTN achieves state-of-the-art results, even comparable with arge scale dynamic architecture method, while enjoying neglectable model complexity overhead compared to fixed architecture methods.



(A) Split CIFAR



(B) Split miniIMN

FIGURE 4.2: ACC(↑) as a function of the episodic memory size on the Split CIFAR-100 and Split miniIMN benchmarks. Best viewed in colors.

ACC(↑) as a function of the episodic memory size We study the models' performances as the memory size increases. We consider the Split CIFAR 100 and Split miniIMN benchmarks and train the models of CTN, ER, MIR, and GEM with the total memory size per task increasing from 50 to 200. Figure 4.2 plots the ACC(↑) curves as a function of the memory size. Generally, the performances of all methods increase with larger memory sizes. Overall, CTN consistently outperforms the competitors across all memory sizes. Notably, in both benchmarks, CTN can achieve comparable performances to the Offline model even when the memory size per task is only 175. The results show that CTN not only excels in the low memory regime but also scales remarkably well when more memory budget is allowed.

4.4.3 Results on Learning with Limited Training Data

TABLE 4.2: Evaluation metrics on the Small Split CIFAR benchmarks, M denotes the memory per task

Method	Reduced Split CIFAR 25%, M = 50			Reduced Split CIFAR 25%, M = 25		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
GEM	51.01±0.95	5.65±1.09	53.39±0.98	47.33±0.89	8.77±1.58	53.79±1.35
ER-Ring	52.02±0.90	4.31±0.94	53.97±0.65	48.15±0.87	8.22±1.17	53.88±0.93
MIR	50.82±0.83	5.22±0.68	53.27±1.05	47.19±0.54	8.41±0.94	53.51±0.74
CTN	**61.27±0.93**	**4.19±0.78**	**61.92±1.15**	**56.17±1.63**	**7.71±1.22**	**61.40±0.64**
Method	Reduced Split CIFAR 10%, M = 50			Reduced Split CIFAR 10%, M = 25		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
GEM	44.06±1.31	6.96±0.87	48.67±0.84	42.67±1.62	8.39±1.35	49.46±0.40
ER-Ring	44.60±1.65	6.07±1.77	48.36±0.46	43.09±1.22	7.68±1.94	49.40±1.40
MIR	46.63±0.56	4.38±0.45	48.35±0.52	44.12±0.94	6.84±1.05	48.48±0.76
CTN	**56.61±0.74**	**4.33±0.48**	**58.77±0.99**	**52.64±0.63**	**6.74±0.73**	**57.27±1.02**

One important goal of continual learning is to be able to learn with a limited amount of training data per task. This setting is much more challenging because it tests the learner's ability to quickly acquire knowledge only with limited training samples by utilizing its past experiences. In this experiment, we explore how different memory-based methods perform with only limited *training samples* per task and memory size. We consider the Split CIFAR benchmark; however, we reduce the amount of training data per task significantly. Particularly, we only consider 25% and 10% of the original data per task while the test data remains the same. We name the new benchmarks Reduced Split CIFAR 25% and Reduced Split CIFAR 10%, respectively. Notably, the Reduced Split CIFAR 10% only has **five samples** per class, which is extremely challenging. We compare CTN with GEM, ER, and MIR on these benchmarks with the memory size of 50 and 25 samples per task.

Table 4.2 shows the results of this experiment. When the training data is scarce, the baselines performances drop significantly, even below 50% ACC(↑) in three settings. CTN, on the other hand, consistently outperforms the baselines by a large margin, from **8% to 10%** across benchmarks, even in the challenging Reduced Split Cifar 10%. Moreover, the three baselines have similarly low LA, showing that they struggle in acquiring new knowledge when the training data of each task are limited. On the other hand, CTN can leverage information about the task-specific features to improve knowledge transfer and the learning outcomes. It is worth noting that even with 25% training data and 50 memory slots per task, CTN already outperforms several baselines that are trained with full data by cross-referencing the results with Table 4.1.

TABLE 4.3: ACC(↑) of each component in CTN on Split CIFAR and Split mini Imagenet with 50 memory slots per task. BC: behavioral cloning (Equation 4.6), C: controller, BO: Bilevel optimization (Equation 4.3)

	BC	C	BO	Split CIFAR			Split miniIMN		
				ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
CTN	✓	✓	✓	**67.65±0.43**	**6.33±0.70**	73.43±0.45	**65.82±0.59**	**3.02±1.13**	67.73±1.73
	✓	✓		66.37±0.53	9.64±0.98	**75.40±0.60**	60.04±1.37	10.48±0.99	**69.87±0.60**
		✓	✓	64.46±1.16	8.51±1.53	72.23±0.54	61.01±1.09	5.31±0.94	64.35±0.83
		✓		62.76±0.49	10.10±0.78	72.12±0.41	58.95±1.76	9.08±1.61	66.94±0.83
ER				61.36±1.01	7.20±0.72	67.05±1.08	53.43±1.18	11.21±1.35	63.46±1.05

4.4.4 Ablation Study

We study the contribution of each component in CTN in its overall performance and consider the Split CIFAR and Split miniIMN benchmarks with an episodic memory of 50 samples per task. Particularly, we are interested in how (1) the controller, (2) the bi-level optimization, and (3) the behavioral cloning strategy contribute to the base model. We implement variants of CTN with different combinations of these components and report the results in Table 4.3. Notably, CTN with only the controller (C) is equivalent to training the base network and the controller using the vanilla experience replay approach. Despite this, the controller can offer significant improvements over ER: over 5% ACC(↑) in Split miniIMN. When the controller is optimized by our proposed bilevel optimization (C + BO), the performances are further improved, showing that our proposed bilevel objective achieves a better trade-off between alleviating forgetting and facilitating knowledge transfer. Lastly, the behavioral cloning strategy can help alleviate forgetting and further strengthen the results. Overall, each of the proposed components adds positive contributions to the base model, and they work collectively as a holistic method and achieved state-of-the-art results in continual learning.

4.4.5 Complexity Analysis

TABLE 4.4: Model complexity of CTN with various backbone architectures

Backbone		Controller		Total	Increase
Structure	# Params	Structure	# Params		
MLP [784-256-256-10]	269,322	Linear model	17,728	287,050	6.58%
Reduced ResNet18	1,095,555	Linear model	20,992	1,116,547	1.92%
Full ResNet18	11,202,162	Linear model	59,200	11,261,362	0.53%

In this section, we study the CTN's complexity with the backbones used in our experiments and report the results in Table 4.4. In all cases, the controller only adds minimal additional parameters, almost neglectable in complex deep architectures such as ResNets [60, 80]. Therefore, we can safely compare CTN

TABLE 4.5: Averaged running time (in seconds) of compared methods on the task-aware continual learning benchmarks. All methods use M=50 memory slots per task, Ring buffer, and up to four gradient updates per samples

Benchmark \ Method	ER-Ring	MIR	AGEM	CTN	GEM
pMNIST	61	92	90	110	103
Split CIFAR100	632	1030	680	910	1700
Split miniIMN	1320	2130	1700	1890	2850

with other fixed architecture methods using the same backbone because they have nearly the same number of parameters.

Table 4.5 reports the averaged running time (in seconds) of considered methods. All methods are implemented using Pytorch [151] version 1.5 and CUDA 10.2. Experiments are conducted using a single K80 GPU and all methods are allowed up to four gradients steps per sample. Clearly, ER-Ring has the most efficient time complexity thanks to its simplicity. On the other hand, GEM has high computational costs because of its quadratic constraints. MIR also exhibits high running time because of its virtual update, which doubles the total gradient updates. CTN, in general, is slightly faster MIR and more efficient than GEM. Overall, CTN achieves a great trade-off between model/computational complexity and performance: CTN's performances are significantly higher than considered baselines with only minimal memory and computational overhead.

4.4.6 Effect of The semantic memory Size

We study how the semantic memory size effects CTN performance. For this experiment, we consider the validation tasks in the Split CIFAR-100 benchmark (the first three tasks) and vary the semantic memory size and episodic memory size such that their total sizes equals to 50 samples per task.

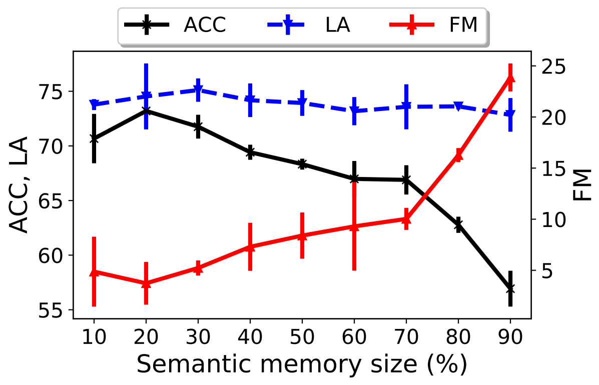


FIGURE 4.3: Effect of memory size on CTN's performance. For every semantic memory size $m \times 50$, the corresponding episodic memory size is $(1 - m) \times 50$.

Figure 4.3 reports the results of this experiment. We can see that when the semantic memory size is 10 (20% of the total memory), CTN achieves the highest ACC, FM(↓) and lowest FM(↓) and these evaluation metrics degrades when the semantic memory sizes increases. Generally, we have to balance the amount of memory for controller and the base network. Since the controller is only a simple model, it only requires a small amount of data in the semantic memory.

4.5 Variants of CTN

TABLE 4.6: Alternative strategies to reduce forgetting in CTN's inner optimization. BC: behavioural cloning strategy in Equation 4.6

Method	Split CIFAR			Split miniIMN		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
CTN-BC	**67.65±0.43**	**6.33±0.70**	**73.43±0.45**	**65.82±0.59**	**3.02±1.13**	**67.73±1.73**
CTN-EWC	60.33±1.44	9.33±1.55	68.78±0.24	57.69±0.96	5.59±0.45	61.53±1.38
CTN-GEM	64.40±2.52	8.06±1.92	71.49±0.46	60.65±0.80	5.83±0.84	64.42±0.46

In this section, we explore alternative strategies for alleviating catastrophic forgetting in CTN's inner optimization problem, which is experience replay (ER) to train the base model ϕ. Particularly, instead of the behavioural cloning strategy in Equation 4.6, we consider two strategy to alleviate forgetting in ER by combining ER with EWC [76] and GEM [80]. Table 4.6 show the results of this experiment on the Split CIFAR100 and Split miniIMN benchmarks. We can see that the behavioural cloning strategy significantly outperforms its competitors, EWC and GEM. Notably, using CTN with EWC requires larger episodic memory to store the previous tasks' parameters and their importance. Moreover, using CTN with GEM results in slower running time since GEM has the slowest training time as shown in Table 4.5. The results show that the behavioural cloning strategy is more suitable for alleviating forgetting in ER, while enjoying less memory overhead or faster running time compared to other alternatives.

4.6 Conclusion

In this Chapter, we propose Contextual Transformation Networks (CTN), a fixed architecture network that can model both the common features and specific features of each task. Through extensive experiments, our results demonstrate that CTN consistently outperforms fixed architecture methods and achieves state-of-the-art results. Moreover, CTN is even comparable with a large scale dynamic architecture network, while enjoying almost no additional model complexity.

Despite promising results, there are still *two* major drawbacks in CTN. First, it is strictly a task-aware method because the controller requires the identifiers as input. Second, CTN uses ER as a proxy to obtain general representation via multitask learning. Such an multi-task training approach usually suffers when tasks are not closely related. In Chapter 5 we seek to develop a more general strategy that is applicable to general continual learning settings and is robust to the negative transfer from unrelated tasks.

4.7 Implementing CTN

4.7.1 CTN with Common Architectures

In this section, we provide the implementation details of CTN on two feedforward network bases that we use in our experiments. We implement the context model as a single regression layer. Moreover, we share the parameter of the scale and shift models γ, β, resulting in one set of parameters that takes a task embedding as input and outputs both scale and shift values for a particular layer of the base network. Next, we will describe our implementation of CTN with the base network as MLP and ResNet [60]. For CTN, we will use $\hat{h}$ as the original features, $\tilde{h}$ as the task-specific features, and h as the combine features.

CTN with Multilayer Perceptron. Consider an $L-$layers MLP with the form:

$$\begin{aligned} h_0 =& \boldsymbol{x} \\ h_l =& \mathrm{ReLU}(\boldsymbol{W}_l^\top h_{l-1}), \forall l = 1, \dots, L-1, \\ h_L =& g_t = \mathrm{Softmax}(\boldsymbol{W}_{L,t}^\top h_{L-1}) \end{aligned}$$

where the last layer is the softmax classifier h_t. Since the last classification layer is already conditioned on the task information, here we are interested in conditioning the intermediate layers $h_{l<L}$. The CTN with MLP is implemented as:

$$\begin{aligned}
\hat{h}_0 =& \boldsymbol{x} \\
\hat{h}_l =& \mathrm{ReLU}(\boldsymbol{W}_l^\top h_{l-1}), \forall l = 1, \dots, L-1, \\
\tilde{h}_l =& \mathrm{ReLU}(\gamma_t \otimes \boldsymbol{W}_l^\top h_{l-1} + \beta_t), \forall l = 1, \dots, L-1, \\
h_l =& \hat{h}_l + \tilde{h}_l \\
h_L =& g_t = \mathrm{Softmax}(\boldsymbol{W}_{L,t}^\top h_{L-1})
\end{aligned}$$

We condition each hidden layer of a MLP by using one context network for each layer. Each context network does not share parameters, however, the scale and shift models for one layer is shared.

CTN with Deep Residual Network. Unlike MLP, we apply the task conditioning after the residual blocks instead of each convolution layer. Particularly, given a residual block defined as:

$$\begin{aligned}
\hat{h}_1 =& \mathrm{ReLU}(\mathrm{BN}(\mathrm{conv}(\boldsymbol{x}))) \quad & \hat{h}_2 = \mathrm{BN}(\mathrm{conv}(h_1)) \\
\hat{h}_3 =& \mathrm{BN}(\mathrm{conv}(\boldsymbol{x})) \quad & \hat{h}_4 = \mathrm{conv}(x) \\
\bar{h} =& \hat{h}_3 + \hat{h}_4 &
\end{aligned}$$

The task-conditioned residual block is computed as:

$$\tilde{h} = \mathrm{ReLU}(\bar{h}) + \mathrm{ReLU}(\gamma_t \otimes \bar{h} + \beta_t)$$

While in principle, it is possible to have a context network for each of the residual block, we empirically found that this does not offer significant improvements over using only one controller on the last residual block. Therefore, we only use one controller on the last residual block in all experiments that use a ResNet.

4.7.2 Pseudo-code

We provide the details algorithm of our CTN and its subroutines in Alg. 1. For simplicity, we drop the dependency of the losses on the parameters and use

$\mathcal{L}^{tr}(\mathcal{B}_n)$ to denote $\mathcal{L}^{tr}(\boldsymbol{\phi}, \boldsymbol{\varphi}, \mathcal{B}_n; \boldsymbol{\theta})$ and $\mathcal{L}(\mathcal{B}_n)$ to denote $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\varphi}, \mathcal{B}_n; \boldsymbol{\theta})$

Algorithm 1: Contextual Transformation Networks (CTN)

```
1  Algorithm TrainCTN($\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{tr}_{1:T}$)
       Require: base model $\boldsymbol{\phi}$, controller $\boldsymbol{\theta}$, classifier $\boldsymbol{\varphi}$
       Init: $\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\varphi}, \mathcal{M}^{em}_t \leftarrow \varnothing, \mathcal{M}^{sm}_t \leftarrow \varnothing$
2      for $t \leftarrow 1$ to $T$ do
3          for $j \leftarrow 1$ to $n_{batches}$ do      // Receive the dataset $D^{tr}_t$ sequentially
4              Receive a mini batch of data $\mathcal{B}_j$ from $\mathcal{D}^{tr}_t$
5              $\boldsymbol{x}^*, y^* \leftarrow$ Random sampling from $\mathcal{B}_j$
6              $\mathcal{M}^{sm}_t \leftarrow$ MemoryUpdate($\mathcal{M}^{sm}_t, \{\boldsymbol{x}^*, y^*\}$)
7              $\mathcal{M}^{em}_t \leftarrow$ MemoryUpdate($\mathcal{M}^{em}_t, \mathcal{B}_j$)
8              for $i \leftarrow 1$ to $n_{outer}$ do
9                  for $n \leftarrow 1$ to $n_{inner}$ do
10                     $\mathcal{B}^{em} \leftarrow$ Sample($\mathcal{M}^{em}_{<t}$)
11                     $\mathcal{B}_n \leftarrow \mathcal{B}^{em} \cup \mathcal{B}_j$
12                     $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \nabla_{\boldsymbol{\phi}}\mathcal{L}^{tr}(\mathcal{B}_n)$ // Inner update the base model $\boldsymbol{\phi}$
                       $\boldsymbol{\varphi} \leftarrow \boldsymbol{\varphi} - \nabla_{\boldsymbol{\varphi}}\mathcal{L}^{tr}(\mathcal{B}_n)$    // Inner update the classifier $\boldsymbol{\phi}$
13                 $\mathcal{B}^{sm} \leftarrow$ Sample ($\mathcal{M}^{sm}_{\leq t}$)
14                 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{B}^{sm})$        // Outer update the controller $\boldsymbol{\theta}$
15         $\mathcal{M}^{em}_t \leftarrow \mathcal{M}^{em}_t \cup \{\pi(\hat{y}/\tau)\}$     // Calculate the behavioural cloning
           outputs
16         $\mathcal{M}^{em} \leftarrow \mathcal{M}^{em} \cup \mathcal{M}^{em}_t$         // Update the total episodic memory
17     return $\boldsymbol{\theta}, \boldsymbol{\phi}$
```

Algorithm 2: CTN subroutines.

```
1 Procedure Forward($\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\varphi}, \boldsymbol{x}, t$)
2     $\gamma_t, \beta_t \leftarrow c_{\boldsymbol{\theta}}(t)$    // Calculate the transforming coefficients
3     $\tilde{h}(\boldsymbol{x}; t) \leftarrow \frac{\gamma_t}{\|\gamma_t\|_2} \otimes \hat{h}(\boldsymbol{x}) + \frac{\beta_t}{\|\beta_t\|_2}$    // Calculate the task-specific features
4     return $g_{\boldsymbol{\varphi}_t}(h(\boldsymbol{x}, t))$

1 Procedure MemoryUpdate($\mathcal{M}, \mathcal{B}$)
      Require: Implement $\mathcal{M}$ as a queue (FIFO) data structure
2     for $(\boldsymbol{x}, y)$ in $\boldsymbol{B}$ do
3         $\mathcal{M}$.append$(\boldsymbol{x}, y)$
4     return $\mathcal{M}$
```

Chapter 5

Fast and Slow Continual Learning

In Chapter 4, we discussed CTN, a promising online continual learning strategy motivated by the CLS theory. This Chapter further refines the fast-and-slow learning idea to DualNets, a general continual learning method. As we will demonstrate, DualNets not only address CTN's weaknesses but also show remarkable successes on various continual learning scenarios.

5.1 Introduction

In Chapter 4, we reviewed the CLS theory and discussed how it motivated the development of CTN. In literature, several continual learning strategies are also inspired from the CLS theory, from using the episodic memory [80] to improving the representation [138, 154]. However, most existing studies were developed on the standard, controlled benchmarks and showed limited success on general scenarios where the data consists limited training samples or distribution shifts [199]. Therefore, the main focus of this Chapter is the development of a novel and general framework inspired by the CLS theory to address not only the conventional continual learning problems but also the real-world challenges of distribution shifts or limited training data.

To this end, we first argue that most existing studies do not closely model the CLS theory because they use a single backbone to model both the hippocampus and neocortex, which binds two representation types into the same network. Therefore, they cannot guarantee to decouple general representation and task-specific representation. Moreover, since such networks are trained to minimize only the supervised loss, they lack a separate and specific slow learning component that supports a general representation. During continual learning, the representation obtained by repeatedly performing supervised learning on a small amount of memory data can be prone to overfitting and may not generalize well across tasks. On the other hand. recent studies in continual learning show that other forms of representation such as unsupervised representation [59, 111] are often more resisting to forgetting compared to the supervised learning ones, which yields little improvements [106]. This result motivates us to conceptualize a novel fast-and-slow learning framework for continual learning, which comprises a two separate learning systems. The fast learner focuses on supervised learning while the slow learner focuses on accumulating better representations. As a result, the fast learner can take advantage of the slow representation to learn new tasks more efficiently, while retaining the old tasks' knowledge.

From the fast-and-slow learning framework, we propose *DualNets* (for Dual Networks), a novel and practical continual learning paradigm consisting of two separate learning systems. Particularly, DualNets consist of two *complementary* and *parallel* training processes. First, the representation learning phase involves only the *slow network*, which continuously optimizes a Self-Supervised Learning (SSL) loss to model the generic, task-agnostic features [59, 111]. Separating

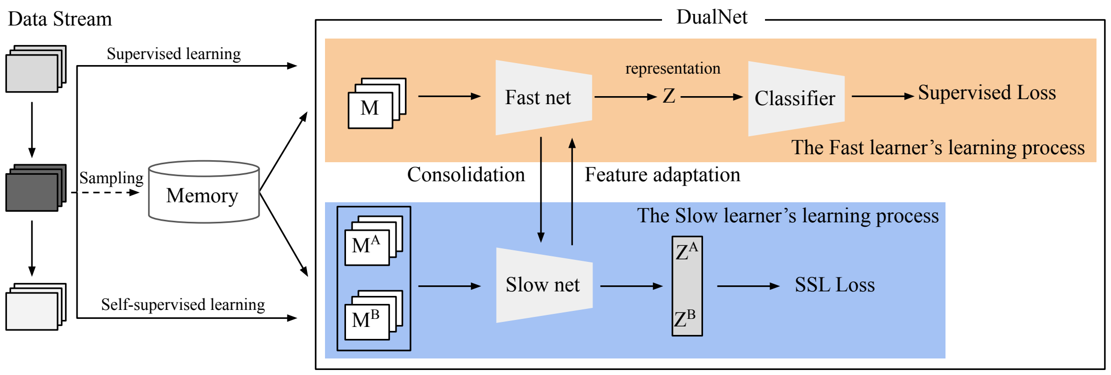


FIGURE 5.1: Overview of the DualNet architecture, which consists of (i) a slow learner (blue) that learns representation by optimizing an SSL loss using samples from the memory, and (ii) a fast learner (orange) that adapts the slow net's representation for quick knowledge acquisition of labeled data. Both learners can be trained synchronously. $\boldsymbol{M}$ denotes a randomly sampled mini-batch and $\boldsymbol{M}^A, \boldsymbol{M}^B$ denote two views of $\boldsymbol{M}$ obtained by applying two different data transformations. Best viewed in colors.

the slow learning phase allows DualNets continuously improve representation even when there are no labeled samples, which is ubiquitous in real-world deployment scenarios where labeled data can be delayed [132] or even limited, which we will demonstrate in Section 5.4.1. Secondly and simultaneously, the supervised learning phase involves both learners. In this phase, the goal is to train the fast learner, a more lightweight model that can do supervised learning more efficiently on data streams. We also propose a simple feature adaptation mechanism so that the fast learner can incorporate the slow representations into its predictions to ensure good results. Figure 5.1 depicts an overview of our DualNets.

Lastly, by design, the original DualNet utilizes all slow features to learn the current sample. We note that this strategy may hinder the performance when the continuum contains unrelated tasks. In such scenarios, there might be negative transfer among tasks, resulting in a performance drop when using all slow features. In continual learning literature, this challenge is commonly addressed by a dynamic architecture design, which uses different subnetworks for the unrelated tasks [199] and can perform well where tasks are not related. However, despite strong results, such strategies are often expensive to train, incurs additional complexities overhead, and often perform poorly in the online learning scenario [129]. Therefore, we propose to enrich DualNet with the ability to prevent negative knowledge transfer from unrelated features by preventing the co-adaptation between the fast and slow learners, allowing the fast learner to learn more useful and robust features for the supervised learning. To this end, we propose DualNet++, which equips DualNet with a simple, yet elegant regularization strategy to alleviate the negative knowledge transfer in continual learning. Particularly, DualNet++ inserts a dropout layer between the fast and slow learners' interaction, which prevents its fast learner co-adapt to the slow features. As a result, DualNet++ is robust to the negative knowledge transfer under the presence of unrelated or interference tasks in continual learning. Notably, as we will empirically verify in Section 5.4.2, DualNet++ achieve promising performance on the CTrL benchmark [199], which was specifically designed to test the model's ability to transfer knowledge in different complex scenarios.

In summary, this Chapter makes the following contributions:

1. We propose DualNet, a novel and generalized continual learning framework comprising two key components of fast and slow learning systems, which is motivated by the CLS theory.
2. We develop to practical algorithms of DualNet and DualNet++, which implements the fast and slow learning approaches for continual learning. Notably, DualNet++ is also robust to the negative knowledge transfer.

3. We conduct extensive experiments to demonstrate DualNet's competitive performance compared to state-of-the-art (SOTA) methods. We also provide comprehensive studies of DualNet's efficacy, robustness to the slow learner's objectives, and scalability to the computational resources.

5.2 Related Work

Representation Learning for Continual Learning Representation learning has been an important research field in machine learning and deep learning [31, 39]. Recent works demonstrated that a general representation could transfer well to many downstream tasks [110], or generalize well under limited training samples [73]. For continual learning, extensive efforts have been devoted to learning a generic representation that can alleviate forgetting while facilitating knowledge transfer. The representation can be learned either by supervised learning [82], unsupervised learning [59, 111, 154], or meta (pre-)training [138, 136]. While unsupervised and meta training have shown promising results on simple datasets such as MNIST and Omniglot, they lack the scalability to real-world benchmarks. In contrast, our DualNets decouple the representation learning into the slow learner, which is scalable in practice by training synchronously with the supervised learning phase. Moreover, our work incorporates self-supervised learning into the continual learning process and does not require any pre-training steps.

5.3 Method

We denote $\mathcal{M}$ as the episodic memory that stores a subset of observed data and interleave them when learning the current samples [80, 130]. From $\mathcal{M}$, we use $\boldsymbol{M}$ to denote a randomly sampled mini-batch, and $\boldsymbol{M}^A$, $\boldsymbol{M}^B$ to denote two views of $\boldsymbol{M}$ obtained by applying two different data transformations. We also denote $\boldsymbol{\phi}$ as the parameter of the slow network that learns general representation from the input data and $\boldsymbol{\theta}$ as the parameter of the fast network that learns the transformation coefficients.

5.3.1 The DualNets Paradigm

DualNet learns generic representations to support better generalization capabilities across both old and new tasks in continual learning. The model consists two main learning modules (Figure 5.1): (i) the slow learner is responsible for learning a general representation; and (ii) the fast learner learns with labeled data from the continuum to quickly capture the new information and then consolidate the knowledge to the slow learner.

DualNets' learning can be broken down into two *synchronous* phases. First, the self-supervised learning phase in which the slow learner optimizes an SSL objective on incoming samples and samples from the episodic memory. Second, the supervised learning phase, which involves the fast learner using the representation from the slow learner and adapting it for supervised learning. The incurred loss will be backpropagated through both learners for supervised knowledge consolidation. Additionally, the fast learner's adaptation is per-sample-based and does not require additional information such as the task identifiers. While the SSL can make the slow learner's representation generic, backprogating the supervised learning loss end-to-end ensures that the slow learner can learn representations that are useful for the supervised learning. Lastly, DualNet uses the same episodic memory's budget as other methods to store the samples and their labels, but the slow learner only requires the samples while the fast learner uses both samples and their labels.

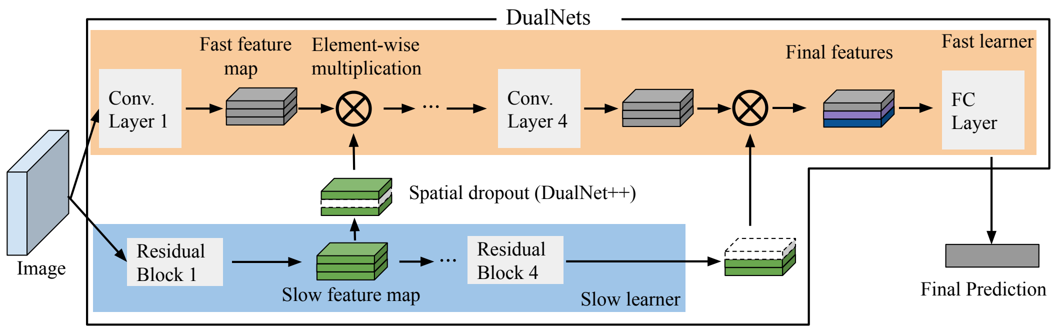


FIGURE 5.2: An illustration of DualNets forward calculation during the supervised learning or inference phase on a standard ResNet [60] backbone. Given an input image, the slow learner (blue) first performs the forward pass to obtain the feature maps. Then, the fast learner (orange) perform its forward pass using both the slow and fast features. Best viewed in colors.

The Slow Learner

The slow learner is a standard backbone network ϕ trained to optimize an SSL loss, denoted by $\mathcal{L}_{SSL}$. As a result, any SSL objectives can be applied in this step. However, to minimize the additional computational resources while ensuring a general representation, we only consider the SSL loss that (i) does not require additional memory units (such as the negative queue in MoCo [180]), (ii) does not always maintain an additional copy of the network (such as BYOL [177]), and (iii) does not use handcrafted pretext losses (such as RotNet [31] or JiGEN [128]). Therefore, we consider contrastive SSL losses [110] and implement DualNets using Barlow Twins [221], a common SSL method that achieved promising results with minimal computational overheads. Formally, Barlow Twins requires two views $\boldsymbol{M}^A$ and $\boldsymbol{M}^B$ by applying two different data transformations to a batch of images $\boldsymbol{M}$. By default, $\boldsymbol{M}$ contains incoming samples from the environment and samples in the episodic memory to maximize the number of samples for SSL. The augmented data is then passed to the slow net ϕ to obtain two representations $\boldsymbol{Z}^A$ and $\boldsymbol{Z}^B$. The Barlow Twins loss is defined as:

$$\mathcal{L}_{\mathcal{BT}} \triangleq \sum_i (1-\mathcal{C}_{ii})^2 + \lambda_{\mathcal{BT}} \sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2, \tag{5.1}$$

where $\lambda_{\mathcal{BT}}$ is a trade-off factor, and $\mathcal{C}$ is the cross-correlation matrix between $\boldsymbol{Z}^A$ and $\boldsymbol{Z}^B$:

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z_{b,i}^A z_{b,j}^B}{\sqrt{\sum_B (z_{b,i}^A)^2} \sqrt{\sum_B (z_{b,j}^B)^2}} \tag{5.2}$$

with b denotes the mini-batch index and i, j are the vector dimension indices. Intuitively, by optimizing the cross-correlation matrix to be identity, Barlow Twins enforces the network to learn essential information that is invariant to the distortions (unit elements on the diagonal) while eliminating the redundancy information in the data (zero element elsewhere). In our implementation, we follow the standard practice in SSL to employ a projector on top of the slow network's last layer to obtain the representations $\boldsymbol{Z}^A, \boldsymbol{Z}^B$. For supervised learning with the fast network, which will be described in Section 5.3.1, we use the slow network's last layer as the representation $\boldsymbol{Z}$.

Optimization in Online Continual Learning In most SSL training, the LARS optimizer [88] is employed for distributed training across many devices, which takes advantage of a large amount of unlabeled data. However, in **online continual learning**, the episodic memory only stores a small number of samples, which are always changing because of the memory updating mechanism. As a result, the data distribution in the episodic memory always drifts after each iteration, and the SSL loss in DualNet presents different challenges compared to the traditional SSL optimization. Particularly, although the SSL objective in continual learning can be easily optimized using one device, we need to quickly capture the knowledge of the currently stored samples before the newer ones replace them. In this work, we propose to optimize the slow learner using the *Look-ahead* optimizer [162], which performs the following updates:

$$\tilde{\phi}_k \leftarrow \tilde{\phi}_{k-1} - \epsilon \nabla_{\tilde{\phi}_{k-1}} \mathcal{L}_{\mathcal{BT}}, \text{ with } \tilde{\phi}_0 \leftarrow \phi \text{ and } k = 1, \ldots, K \tag{5.3}$$

$$\phi \leftarrow \phi + \beta(\tilde{\phi}_K - \phi), \tag{5.4}$$

where β is the Look-ahead's learning rate and ϵ is the Look-ahead's SGD learning rate. As a special case of $K = 1$, the optimization reduces to the traditional optimization of $\mathcal{L}_{\mathcal{BT}}$ using SGD. By performing $K > 1$ updates using a standard SGD optimizer, the look-ahead weight $\tilde{\phi}_K$ is used to perform a momentum update for the original slow learner ϕ. As a result, the slow learner optimization can explore regions that are undiscovered by the traditional optimizer and enjoys faster training convergence [162]. Note that SSL focuses on minimizing the training loss rather than generalizing this loss to unseen samples, and the learned representation requires to be adapted to perform well on a downstream task. Therefore, such properties make the Look-ahead optimizer a more suitable choice over the standard SGD to train the slow learner. For the batch continual learning setting [76], because the model can learn the current task for many epochs, it is sufficient to train the slow learner with the standard SGD in this scenario.

Lastly, we emphasize that although we choose to use Barlow Twins as the SSL objective, DualNets are compatible with any existing methods in the literature, which we will explore empirically in Section 5.4.1. Moreover, we can always train the slow learner in the background by optimizing Equation 5.1 synchronously with the continual learning of the fast learner, which we will detail in the following section.

The Fast Learner

Given a labeled sample $\{\boldsymbol{x}, y\}$, the fast learner's goal is utilizing the slow learner's representation to learn this sample via an adaptation mechanism. We propose a general context-free adaptation mechanism by extending and improving the channel-wise transformation [112, 214] to the general continual learning setting. Particularly, such strategies relies on low dimensional context vectors, such as task identifiers, to learn a channel-wise transformation coefficients. In the task-free setting, the model needs to learn such information from the raw, high dimensional images. To compensate for the increased learning complexity, we propose to implement the fast net as a smaller neural network instead of a simple linear layer [214]. Moreover, the transformation is pixel-wise instead of channel-wise to allow for a more fine-grained usage of the slow feature given the current image.

Formally, let $\{\boldsymbol{h}_i\}_{i=1}^{L}$ be the feature maps from the slow learner's layers on the image $\boldsymbol{x}$, e.g. $\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4$ are outputs from four residual blocks in ResNets [60], our goal is to obtain the adapted feature h'_L conditioned on the image $\boldsymbol{x}$. Therefore, we design the fast learner as a simple CNN with L layers, and the adapted feature $\boldsymbol{h}'_L$ is obtained as

$$\begin{aligned} \boldsymbol{m}_l =& g_{\boldsymbol{\theta},l}(\boldsymbol{h}'_{l-1}), \text{ with } \boldsymbol{h}'_0 = \boldsymbol{x} \text{ and } l = 1, \dots, L \\ \boldsymbol{h}'_l =& \boldsymbol{h}_l \odot \boldsymbol{m}_l, \quad \forall l = 1, \dots, L, \end{aligned} \tag{5.5}$$

where $\odot$ denotes the element-wise multiplication, $g_{\boldsymbol{\theta},l}$ denotes the l-th layer's output from the fast network $\boldsymbol{\theta}$ and has the same dimension as the corresponding slow feature $\boldsymbol{h}_l$. The fast net final layer's transformed feature $\boldsymbol{h}'_L$ will be fed into a classifier for prediction.

Thanks to the simplicity of the transformation, the fast learner is light-weight but still can take advantage of the slow learner's rich representation for better supervised learning. Meanwhile, the slow learner is mostly trained by the SSL loss to obtain a generic representation that is resistant to catastrophic forgetting. Figure 5.2 illustrates the fast and slow learners' interaction during the supervised learning or inference phase.

The Fast Learner's Objective To further facilitate the fast learner's knowledge acquisition during supervised learning, we also mix the current sample with previous data in the episodic memory, which is a form of experience replay (ER). Particularly, given the incoming labeled sample $\{\boldsymbol{x}, y\}$ and a mini-batch of memory data $\boldsymbol{M}$ belonging to a past task k, we consider the ER with a soft label loss [120] for the supervised learning phase as:

$$\begin{aligned}\mathcal{L}_{tr} =& \mathrm{CE}(\pi(\mathrm{DualNet}(\boldsymbol{x}), y) + \frac{1}{|\boldsymbol{M}|}\sum_{i=1}^{|\boldsymbol{M}|} \mathrm{CE}(\pi(\hat{y}_i), y_i) + \\ &+ \lambda_{tr} D_{\mathrm{KL}}\left(\pi\left(\frac{\hat{y}_i}{\tau}\right)\middle\|\, \pi\left(\frac{\hat{y}_k}{\tau}\right)\right),\end{aligned} \tag{5.6}$$

where CE is the cross-entropy loss , D_{KL} is the KL-divergence, $\hat{y}$ is the DualNet's prediction, $\hat{y}_k$ is snapshot of the model's logits (the fast learner's prediction) of the corresponding sample at the end of task k, $\pi(\cdot)$ is the softmax function with temperature τ, and λ_{tr} is the trade-off factor between the soft and hard labels in the training loss. Similar to [214, 170], Equation 5.6 requires minimal additional memory to store the soft label $\hat{y}$ in conjunction with the image $\boldsymbol{x}$ and the hard label y.

5.3.2 DualNet++

This section details DualNet++, an improved version of DualNet to tackle the challenge of preventing negative knowledge transfer and learning modular knowledge in continual learning [199]. In many real-world applications, data in continual learning may contain vastly different visual features, which could even be adversarial to one another [217]. Common approaches to tackle such challenges [199, 213] are mainly based on the dynamic architectures, which compartmentalize knowledge into different modules. Then, the model only uses the relevant subnetwork to learn a current task, which only transfers useful knowledge while alleviating the negative effects of unrelated features. To make DualNets applicable to real-world problems, we believe that selective knowledge transfer is an important component to enrich DualNet with.

To this end, we analyze the DualNet's drawback in achieving a good transfer when learning from complex continual learning streams. We argue that since the slow features in DualNets are obtained from all tasks' data, the original DualNet will learn the fast features dependent on the slow features. While this is helpful in the controlled environments with no negative transfers, it will hinder the performance when there are tasks unrelated to one another. Thus, we propose DualNet++ that alleviates the co-adaptation between the fast and slow features, allowing it to learn more robust features that are useful for the current task. DualNet++ introduces a simple dropout layer [35] between the fast and slow learners' interactions. As a result, under the presence of negative transfer, the fast learner will not become dependent on the slow feature [35, 53] and can focus on learning features useful for the current inputs.

To implement DualNet++, we insert a spatial dropout layer [55] between the fast and slow learner interaction. Formally, DualNet++ replace the interaction in Eq. 5.5 as:

$$\begin{aligned} \boldsymbol{m}_l =& g_{\boldsymbol{\theta},l}(\boldsymbol{h}'_{l-1}), \text{ with } \boldsymbol{h}'_0 = \boldsymbol{x} \text{ and } l = 1, \ldots, L \\ \boldsymbol{h}'_l =& \boldsymbol{D}_l \odot \boldsymbol{h}_l \odot \boldsymbol{m}_l, \quad \forall l = 1, \ldots, L, \end{aligned} \tag{5.7}$$

where $\boldsymbol{D}_l \in \mathbb{R}^{n \times w \times h}$ is a spatial dropout mask obtained as

$$d_{l,i} \sim \text{Bernoulli}(p), \forall i = 1, \ldots, n \tag{5.8}$$

$$\boldsymbol{D}_l \leftarrow \text{Repeat}(\boldsymbol{d}_l, n \times w \times h), \boldsymbol{d}_l = \{d_{l,1}, \ldots, d_{l,n}\}. \tag{5.9}$$

In Eq. 5.8, $\text{Bernoulli}(\text{p})$ denotes a sample randomly drawn from a Bernoulli distribution with probability p, and Repeat in Eq. 5.9 denotes reshaping a vector to a particular dimensions by repeating its values along the required axes. Specifically, the dropout mask $\boldsymbol{D}$ is obtained by performing n independent dropout trial on a feature map of size $n \times w \times h$, and each trial will zero an entire channel. We also note that it is possible to apply the traditional dropout [35] on the pixels independently, or inserting dropout in the backbone networks. However, preliminary results of such strategies are not promising due to the incompatibility between dropout and batch normalization [51, 142]. Therefore, we decided to not explore these configurations further. In contrast, the spatial dropout is more suitable for convolutional neural networks, and is only inserted in the fast and slow networks' interaction, not between the hidden layers. Lastly, a recent work [191] also show promising results of applying dropout in continual learning. However, their studies only focus on the simple feed-forward architectures and left the convolution networks unexplored.

5.4 Experiments

We compare DualNets against competitive continual learning approaches in both the online [80] and offline [76, 199] scenarios. Our goal of the experiments is to investigate the following hypotheses: (i) DualNets can work well across different continual learning scenarios; (ii) DualNets are robust to the choice of the SSL loss; (iii) DualNets are scalable with the number of SSL training iterations; (iv) DualNet++ can efficiently learn under the presence of unrelated tasks and distribution shifts. In all experiments, DualNet++'s dropout ratio is set as $p = 0.1$ for the online setting, and $p = 0.2$ for the batch setting, unless otherwise stated.

5.4.1 Online Continual Learning Experiments

We first consider the *Online Continual Learning setting* [80] where both the tasks and samples within each task arrive sequentially. This setting presents a unique challenge where the catastrophic forgetting and facilitating knowledge transfer problems are entangled [170]. Thus, successful online continual learning solutions must achieve a good trade-off of these conflicting objectives.

Setup

Benchmarks We consider the “Split" continual learning benchmarks constructed from the miniImageNet [68] and CORE50 dataset [78] with three validation tasks and 17, 10 continual learning tasks, respectively. Each task is created by randomly sampling without replacement five classes from the original dataset. We also consider both the task-aware and task-free protocols. For the task-aware (TA) protocol, the task identifier is available, and only the corresponding classifier is selected for training and evaluation. In contrast, the task-identifiers are not given in the task-free (TF) protocol, and the models have to predict all classes observed so far.

Baselines We compare DualNet and DualNet++ against a suite of state-of-the-art continual learning methods. First, we consider ER [130], a simple experience replay method that works consistently well across benchmarks. Then we include DER++ [170], an ER variant that augments ER with a ℓ_2 loss on the soft labels. We also compare with CTN [214] a recent state-of-the-art method on the online task-aware setting. For all methods, the hyper-parameters are selected by performing grid-search on the cross-validation tasks.

Architecture We use a full ResNet18 [60] as the backbone for all methods. In addition, we construct the DualNet's fast learner as follows: the fast learner has the same number of convolutional layers as the number of residual blocks in the slow learners. A residual block and its corresponding fast learner's layer will have the same output dimensions. With this configuration, the fast learner's architecture is uniquely determined by the slow learner's network and only increased the number of parameters by about 20%. Lastly, all networks in our experiments are trained from scratch.

Training In the supervised learning phase, all methods are optimized by the (SGD) optimizer **over one epoch** with mini-batch size 10 and 32 on the Split miniImageNet and CORE50 benchmarks respectively [80, 214]. In the representation learning phase, we use the Look-ahead optimizer [162] to train the DualNets' slow learner as described in Section 5.3.1. We employ an episodic memory with *50 samples per task* and the Ring-buffer management strategy [80] in the task-aware setting. In the task-free setting, the memory is implemented as a reservoir buffer [5] with 100 samples per class. We simulate the synchronous training property in DualNet by training the slow learner with n iterations using the episodic memory data before observing a mini-batch of labeled data.

Data pre-processing DualNet's slow learner follows the data transformations used in BarlowTwins [221]. For the supervised learning phase, we consider two options. First, the standard data pre-processing of no data augmentation during both training and evaluation, which is commonly implemented in existing studies [80, 130]. Second, we also train the baselines with data augmentation for a fair comparison. However, we observe the data transformation in [221] is too aggressive; therefore, we only implement the random cropping and flipping for the supervised training phase of these baselines. In all scenarios, the inference phase does not any use data augmentations.

Results of Online Continual Learning Benchmarks

Table 5.1 reports the evaluation metrics on the CORE50 and Split miniImageNet benchmarks, where we omit CTN's performance on the task-free setting since it is strictly a task-aware method. Our DualNet's slow learner optimizes the Barlow Twins objective for $n = 3$ iterations between every incoming mini-batch of labeled data. We will explore the impact of the SSL iteration in Section 5.4.1.

Generally, data augmentation creates more samples to train the models and provides improvements in all cases. Consistent with previous studies, we observe that DER++ performs slightly better than ER thanks to its soft-label loss. Similarly, CTN can perform better than both ER and DER++ because of its ability to model task-specific features. Overall, our DualNets consistently outperform other baselines by a large margin, even with the data augmentation propagated to their training. Specifically, DualNets are more resistant to catastrophic forgetting (lower FM) while greatly facilitating knowledge transfer (higher LA), which results in better overall performance, indicated by higher ACC. We also observe that DualNet++ performs marginally better than DualNet in all cases, suggesting the benefits of the spatial dropout regularization. Lastly, since

TABLE 5.1: Evaluation metrics on the Split miniImageNet and CORE50 benchmarks. All methods use an episodic memory of 50 samples per task in the TA setting, and 100 samples per class in the TF setting. The "Aug" suffix denotes using data augmentation. We highlight the methods with best mean metrics in bold, and underline the second best methods

Method	Split miniImageNet-TA			Split miniImageNet-TF		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
ER	58.24±0.78	9.22±0.78	65.36±0.71	25.12±0.99	28.56±1.10	49.04±1.56
ER-Aug	59.80±1.51	4.68±1.21	58.94±0.69	27.94±2.44	29.36±3.23	54.02±1.02
DER++	62.32±0.78	7.00±0.81	67.30±0.57	27.16±1.99	34.56±2.48	59.54±1.53
DER++-Aug	63.48±0.98	4.01±1.21	62.17±0.52	28.26±1.81	36.70±1.85	62.70±0.41
CTN	65.82±0.59	**3.02±1.13**	67.43±1.37	N/A	N/A	N/A
CTN-Aug	68.04±1.23	3.94±0.98	69.84±0.78	N/A	N/A	N/A
DualNet	73.20±0.68	3.86±1.01	74.12±0.12	36.86±1.36	28.63±2.26	63.46±1.97
DualNet++	**74.24±0.95**	**2.83±0.71**	**74.11±0.30**	**37.56±1.12**	**27.13±1.16**	**63.96±1.02**

Method	CORE50-TA			CORE50-TF		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
ER	41.72±1.30	9.10±0.80	48.18±0.81	21.80±0.70	14.42±1.10	33.94±1.49
ER-Aug	44.16±2.05	5.72±0.02	47.83±1.61	25.34±0.74	15.28±0.63	37.94±0.91
DER++	46.62±0.46	4.66±0.46	48.32±0.69	22.84±0.84	13.10±0.40	34.50±0.81
DER++-Aug	45.12±0.68	5.02±0.98	47.67±0.08	28.10±0.80	10.43±2.10	36.16±0.19
CTN	54.17±0.85	5.50±1.10	55.32±0.34	N/A	N/A	N/A
CTN-Aug	53.40±1.37	6.18±1.61	55.40±1.47	N/A	N/A	N/A
DualNet	57.64±1.36	4.43±0.82	58.86±0.66	38.76±1.52	8.06±0.43	**40.00±1.67**
DualNet++	**59.07±1.30**	**2.86±0.92**	**59.23±1.03**	**39.42±1.80**	**7.08±2.25**	39.52±1.09

our DualNets have a similar supervised procedure as DER++, this result shows that the DualNets' representation learning and fast adaptation mechanism are beneficial to continual learning.

Ablation Study of the Slow Learner Objectives and Optimizers

We now study the effects of the slow learner's objective and optimizer on the final performance of DualNets by considering several objectives to train the

TABLE 5.2: DualNet's performance with different slow learner objectives and optimizers on the Split miniImageNet-TA benchmark

DualNet	SGD			Look-ahead		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
Barlow Twins	64.20$\pm$2.37	4.79$\pm$1.19	64.83$\pm$1.67	**73.20$\pm$0.68**	**3.86$\pm$1.01**	**74.12$\pm$0.12**
SimCLR	**71.49$\pm$1.01**	**4.23$\pm$0.46**	**72.64$\pm$1.20**	72.13$\pm$0.44	4.13$\pm$0.52	73.09$\pm$0.16
SimSiam	70.55$\pm$0.98	4.93$\pm$1.31	71.90$\pm$0.65	71.94$\pm$0.64	4.21$\pm$0.28	72.93$\pm$0.38
BYOL	69.76$\pm$2.12	**4.23$\pm$1.41**	70.33$\pm$0.87	71.73$\pm$0.47	**3.96$\pm$0.62**	72.06$\pm$0.28
Classification	68.50$\pm$1.67	5.53$\pm$1.67	**72.93$\pm$1.10**	70.96$\pm$1.08	6.33$\pm$0.28	73.92$\pm$1.14
DualNet++	SGD			Look-ahead		
	ACC(↑)	FM(↓)	LA(↑)	ACC(↑)	FM(↓)	LA(↑)
Barlow Twins	69.76$\pm$1.43	**2.93$\pm$0.67**	68.96$\pm$1.03	**74.24$\pm$0.95**	**2.83$\pm$0.71**	**74.11$\pm$0.30**
SimCLR	**71.16$\pm$0.72**	3.13$\pm$1.33	71.96$\pm$0.77	72.14$\pm$0.81	4.12$\pm$0.28	73.33$\pm$0.58
SimSiam	69.56$\pm$0.11	4.53$\pm$0.54	71.66$\pm$1.01	71.99$\pm$1.33	4.13$\pm$0.45	73.01$\pm$0.57
BYOL	69.16$\pm$1.12	4.24$\pm$1.21	69.93$\pm$1.46	71.63$\pm$1.12	4.96$\pm$1.88	72.73$\pm$0.77
Classification	69.83$\pm$0.36	5.26$\pm$0.54	**72.20$\pm$0.66**	70.96$\pm$0.52	5.10$\pm$0.57	72.96$\pm$0.86

slow learner. First, we consider the *classification loss* to train the slow net, which reduces DualNet's representation learning to only supervised learning. Second, we consider various contrastive SSL losses, including SimCLR [173], SimSiam [206], and BYOL [177]. In this setting, DualNets' slow representation involves a direct optimization of a SSL loss and an indirect classification loss backpropagated via the fast learner.

We consider the Split miniImageNet-TA and TF benchmark with 50 memory slots per task and optimize each objective using the SGD and Look-ahead optimizers. Table 5.2 reports the result of this experiment. In general, we observe that SSL objectives achieve a better performance than the classification loss. Moreover, the Look-ahead optimizer consistently improves the performances on all objectives compared to the SGD optimizer. This result shows that the DualNets design is general and can work well with different slow learner's objectives. Interestingly, when using the Look-ahead optimizer, we observe a correlation between the SSL losses in DualNets with their performances in the standard SSL scenario [221]. This result suggests that DualNets can take advantage of future SOTA SSL losses to further improve the performance.

Ablation Study of Self-Supervised Learning Iterations

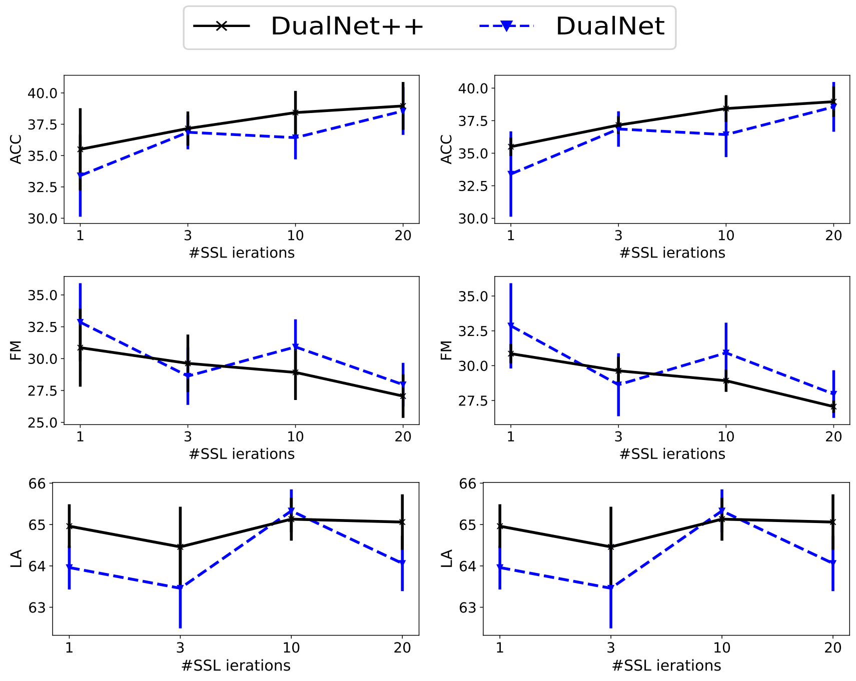


FIGURE 5.3: Performance of DualNet and DualNet++ with different self-supervised learning iterations n.

We now investigate DualNet's performances with different SSL optimization iterations n. Small values of n indicate there is little to no delay of labeled data from the continuum, and the fast learner has to query the slow learner's representation continuously. On the other hand, larger n simulates the situations where labeled data is delayed, which allows the slow learner to train its SSL objective for more iterations between each query from the fast learner. In this experiment, we gradually increase the SSL training iterations between each supervised update by varying from $n = 1$ to $n = 20$.

We run the experiments on both the Split miniImageNet benchmarks under the TA and TF settings. Figure 5.3 reports the result of DualNet and DualNet++ in this scenario. In general, we observe that in all cases, the average accuracy ACC(↑) increases as more SSL iterations are allowed. The same conclusion also holds for the FM(↓) and LA(↑) metrics, although there are small fluctuations at $n = 3$ and $n = 10$. Moreover, DualNet++ is more stable than DualNet in this experiment by having smaller variance across different runs. We can conclude that both DualNet and DualNet++ are highly scalable with the number of SSL training iterations. This promising result demonstrates the DualNets' potential to be deployed in real-world continual learning scenarios where labeled data is delayed [132], which allows the slow learner to learn in the background.

Ablation Study of DualNets Components

TABLE 5.3: Evaluation of DualNet's slow learner on the Split miniImageNet TA and TF benchmarks

DualNet	Split miniImageNet-TA		
	ACC(↑)	FM(↓)	LA(↑)
Slow + Fast Nets	**73.20**$\pm$0.68	**3.86**$\pm$1.01	**74.12**$\pm$0.12
Slow Net	68.33$\pm$0.57	5.12$\pm$0.78	69.20$\pm$0.32
DualNet	Split miniImageNet-TF		
	ACC(↑)	FM(↓)	LA(↑)
Slow + Fast Nets	**36.86**$\pm$1.36	**28.63**$\pm$2.26	**63.46**$\pm$1.97
Slow Net	27.30$\pm$0.25	34.60$\pm$1.12	59.70$\pm$1.26

Compared to the standard ER strategy with soft labels [120, 170], DuelNets introduce an additional fast learner and a representation learning phase. In this experiment, we investigate the contribution of the fast learner on the Split mini-ImageNet benchmark, both the TA and TF settings. We create a variant, *Slow Learner*, that uses only a ResNet backbone to optimize both the supervised and SSL losses. Table 5.3 report the result of this experiment. We can see that the slow learner variant binds both representation types into the same backbone and performs significantly worse than the original DualNet in both scenarios. This result corroborates with our motivation in Section 5.1 that it is more beneficial to separate the two representations into two distinct systems.

Semi-Supervised Continual Learning Setting

TABLE 5.4: Evaluation metrics on the Split miniImageNet-TA benchmarks under the semi-supervised setting, where ρ denotes the fraction of data that is labeled

Method	$\rho = 10\%$		
	ACC(↑)	FM(↓)	LA(↑)
ER	41.66$_{\pm 2.72}$	6.80$_{\pm 2.07}$	42.33$_{\pm 1.51}$
DER++	44.56$_{\pm 1.41}$	4.55$_{\pm 0.66}$	43.03$_{\pm 0.71}$
CTN	49.80$_{\pm 2.66}$	3.96$_{\pm 1.16}$	47.76$_{\pm 0.99}$
DualNet	54.03$_{\pm 2.88}$	3.46$_{\pm 1.17}$	49.96$_{\pm 0.17}$
DualNet++	**58.03**$_{\pm \mathbf{0.99}}$	**2.16**$_{\pm \mathbf{0.59}}$	**53.56**$_{\pm \mathbf{0.11}}$
Method	$\rho = 25\%$		
	ACC(↑)	FM(↓)	LA(↑)
ER	50.13$_{\pm 2.19}$	6.76$_{\pm 1.51}$	51.90$_{\pm 2.16}$
DER++	51.63$_{\pm 1.11}$	6.03$_{\pm 1.46}$	52.36$_{\pm 0.55}$
CTN	55.90$_{\pm 0.86}$	3.84$_{\pm 0.32}$	55.69$_{\pm 0.98}$
DualNet	62.80$_{\pm 2.40}$	3.13$_{\pm 0.99}$	59.60$_{\pm 1.87}$
DualNet++	**63.96**$_{\pm 1.22}$	**2.20**$_{\pm \mathbf{0.32}}$	**60.66**$_{\pm \mathbf{1.02}}$

In real-world continual learning scenarios, there exist abundant unlabeled data, which are costly and even unnecessary to label entirely. Therefore, a practical continual learning system should be able to improve its representation using unlabeled samples while waiting for the labeled data. To test the performance of existing methods in such scenarios, we create a *semi-supervised continual learning* benchmark, where the data stream contains both labeled and unlabeled data. For this, we consider the Split miniImageNet-TA benchmark but provide labels randomly to a fraction (ρ) of the total samples, which we set to be $\rho = 10\%$ and $\rho = 25\%$. The remaining samples are unlabeled and cannot be processed by the baselines we have considered so far. In contrast, such samples can go directly to the DualNet's slow learner to improve its representation while the fast learner stays inactive. Other configurations remain the same as the experiment in Section 5.4.1.

Table 5.4 shows the results of this experiment. Under the limited labeled data regimes, the results of ER and DER++ drop significantly. Meanwhile, CTN can still maintain competitive performances thanks to additional information from the task identifiers, which remains untouched. On the other hand, both DualNet and DualNet++ can efficiently leverage the unlabeled data to improve its performance and outperform other baselines, even CTN. We also observe larger gaps between DualNet++ and DualNet, especially with $\rho = 10\%$, compared to the fully supervised scenario. This gap is attributed to the dropout layers in DualNet++, which improve the fast learner's ability to use the slow features and to better perform supervised learning. Overall, the result demonstrates DualNets potential to work in a real-world environment, where data is partially labeled.

DualNet's Upper Bound

In our work, there are three factors affecting the DualNets' upper bound: (i) model architecture: slow net (standard backbone) versus fast and slow nets

TABLE 5.5: Performance of the Offline model under different configuration on the Split miniImageNet-TA benchmark, * denotes the method is trained in the continual learning setting

Architecture	Method	Loss	Data Aug	ACC
Fast+Slow nets	DualNet*	SL+SSL	Yes	73.20$\pm$0.68
	Offline	SL	No	75.83$\pm$1.07
	Offline	SL	Yes	77.63$\pm$0.48
	Offline	SL+SSL	Yes	**77.98$\pm$0.16**
Slow net	Offline	SL	No	71.15$\pm$2.95
	Offline	SL	Yes	75.46$\pm$0.97

(DualNets); (ii) training loss: supervised learning loss (SL) or supervised and self-supervised learning losses (SL+SSL); and (iii) data augmentation. As a result, we believe that an upper bound of DualNet is a model having all three factors as DualNet (has fast and slow learners, optimized both SL and SSL losses with data augmentation) and is trained offline. The offline model has access to all tasks' data to simultaneously optimizes both the SL and SSL losses, which are backpropagated through both learners. Here we consider the offline model trained up to five epochs.

We explore different combinations of the aforementioned factors to train an Offline model on the Split miniImageNet-TA benchmark and report the result in Table 5.5. Note that the configuration of Slow Net + Offline + SL + no data augmentation is the previous result reported in [214]. Our argued upper bound for DualNet has the following configuration: Fast + Slow nets + Offline + SL + SSL + data augmentation. The result confirms the upper bound of DualNet. Moreover, in the offline training with all data, the SSL only contributes a minor improvement to the SL. However, in continual learning, SSL is more beneficial because its representation does not depend on the class label, and therefore more resistant to catastrophic forgetting when old task data is limited.

TABLE 5.6: Details of the CTrL benchmark streams built from five common datasets: CIFAR-10 [28], MNIST [19], DTD [43], F-MNIST (Fashion MNIST) [87], and SVHN [33]

Stream	Configuration	T1	T2	T3	T4	T5	T6
$\mathcal{S}^{+}$	**Dataset**	CIFAR-10	MNIST	DTD	F-MNIST	SVHN	CIFAR-10
	# Train samples	4000	400	400	400	400	400
	# Val. samples	2000	200	200	200	200	200
$\mathcal{S}^{-}$	**Dataset**	CIFAR-10	MNIST	DTD	F-MNIST	SVHN	CIFAR-10
	# Train samples	400	400	400	400	400	4000
	# Val. samples	200	200	200	200	200	2000
$\mathcal{S}^{\text{in}}$	**Dataset**	R-MNIST	CIFAR-10	DTD	F-MNIST	SVHN	R-MNIST
	# Train samples	4000	400	400	400	400	50
	# Val. samples	2000	200	200	200	200	30
$\mathcal{S}^{\text{out}}$	**Dataset**	CIFAR-10	MNIST	DTD	F-MNIST	SVHN	CIFAR-10
	# Train samples	4000	400	400	400	400	400
	# Val. samples	2000	200	200	200	200	200
$\mathcal{S}^{\text{pl}}$	**Dataset**	MNIST	DTDd	F-MNIST	SVHN	CIFAR-10	-
	# Train samples	400	400	400	400	4000	-
	# Val. samples	200	200	200	200	2000	-

5.4.2 Batch Continual Learning Experiments

We now consider the *Batch Continual Learning setting* [76], where all data samples of a task arrive at each continual learning step. As a result, the model is allowed to train on these samples for multiple epochs before moving on to the next task. Thus, most methods do not suffer from the difficulties of training deep neural networks online and focus only on preventing catastrophic forgetting [170].

Setups

The CTrL Benchmark In the batch learning setting, we focus on exploring DualNet's ability to facilitate knowledge transfer in complex continual learning scenarios. To this end, we consider the CTrL benchmark [199], which was carefully designed to access the model's ability to selectively transfer knowledge while avoiding catastrophic forgetting. Before introducing the CTrL streams, we briefly summarize the concept of related tasks used in CTrL.

We denote a continual learning task as $\mathcal{T}$. Then, CTrL introduces four variants of $\mathcal{T}$ as: (1) $\mathcal{T}^-$: a task whose data is sampled from the same distribution as $\mathcal{T}$ but has a much smaller number of training samples; (2) $\mathcal{T}^+$ is similar to $\mathcal{T}^-$ but has much more training samples than $\mathcal{T}$; (3) $\mathcal{T}'$ is similar to $\mathcal{T}$ but has a different input distribution, i.e., different background colors; and (4) $\mathcal{T}''$ is similar to $\mathcal{T}$ but has a different output distribution, i.e., the label order is randomly permuted. In addition, there are no relationships between two tasks that have different subscripts. With this notation, the CTrL benchmarks introduce five continual learning streams to evaluate five basic knowledge transfer abilities comprehensively.

In the $\mathcal{S}^- = \{\mathcal{T}_1^+, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_1^-\}$ stream, the last task is similar to the first one but it has much smaller training samples. Therefore, successful methods must remember and transfer the knowledge after learning four unrelated tasks.

In the $\mathcal{S}^+ = \{\mathcal{T}_1^-, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_1^+\}$ stream, the first task is similar but has much smaller data than the last one, which requires the model to remember and update the knowledge after learning irrelevant tasks.

The $\mathcal{S}^{\text{in}} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_1'\}$ and $\mathcal{S}^{\text{out}} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_1''\}$ streams require the model to learn representations that are useful to either input or output distribution shifts.

Lastly, the $\mathcal{S}^{\text{pl}} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4, \mathcal{T}_5\}$ stream test the model's ability to learn unrelated tasks with the potential interference from unrelated features. Table 5.6 provides the details of each stream in the CTrL benchmark. Interestingly, the $\mathcal{S}^-, \mathcal{S}^+, \mathcal{S}^{\text{in}}$ and $\mathcal{S}^{\text{out}}$ streams contain one unrelated task from the DTD dataset [43], which has very different visual features from the remaining tasks (see Table 5.6). Therefore, we believe the CTrL benchmark can access DualNets' ability to selective transfer useful knowledge under the presence of negative transfer.

Baselines Existing works have shown that standard static architecture methods struggle to solve the CTrL benchmark while dynamic architecture approaches show more promising results [199, 213] thanks to their ability to compartmentalize knowledge into modules. We follow the experimental setting in [213] and compare DualNets with a suite of competitive baselines. First, we consider static architecture approaches of **EWC** [76] and **O-EWC** [104] (online EWC), which use a quadratic regularizer to penalize changes to important parameters of previous tasks according to the Fisher information. The original EWC [76] maintains an estimate of the Fisher information matrix for each task, while O-EWC maintains a moving average of the parameters importance. Next, we include experience replay **ER**, dark experience replay **DER++**, and the naive **Finetune** strategy that trains a single model without any continual learning strategies. We also consider the task-free and task-aware variants of these baselines.

Second, we consider a suite of dynamic architecture approaches. The **Independent** [80] baseline trains a separate model for each task. HAT [117] proposes to learn hard attention masks to gate the backbone network, which prevents catastrophic forgetting. **SG-F** [189] proposes a task-specific structural network that learns to update existing modules, combine modules, and add new modules. **MNTDP** [199] organizes the backbone network into modules and efficiently searches the path configuration to connect the modules to solve a given task. Lastly, **LMC** [213], a recent dynamic architecture method that proposes to equip each module with a local structural component to predict its relevance for a given input, which does not require task identifier at test time [1] and provide a more systematic strategy to expand and search the layout over modules. Although these methods are not task-free, [213] found that using a shared classifier might be helpful for the CTrL benchmark. Thus, we also considered, within the task-aware category, a shared-head variant of these baselines, denoted by the (H) suffix. The task-identifiers are only used for selecting the subnetworks in the backbone network.

Training We follow the training procedure provided in [213] for a fair comparison. Particularly, for each task, we train each method over 100 epochs using the Adam optimizer [52]. We use a weak data augmentation of random flipping and random cropping for both the supervised learning phase of all methods and the self-supervised learning phase of DualNet. Furthermore, since DualNets are allowed to train for 100 epochs, it is sufficient to use the standard SGD to optimize the SSL loss. Our preliminary experiments show that in this setting, the Look-ahead and SGD optimizers achieve the same results. Therefore, we train DualNets' supervised and SSL losses using the standard SGD optimizer.

Regarding the model complexity, we anchor on the final model of LMC, the state-of-the-art method on this benchmark, to calculate the total parameters used. Then, we select the replay buffer size of each method so that their total parameters [2] equals to LMC. We repeat each experiment five times and report the average ACC(↑) and BWT(↑) at the end of learning.

Evaluation Metrics on CTrL

Table 5.7 reports the evaluation metrics at the end of training on the CTrL benchmarks. We organize the results into two blocks: (i) task-aware setting that

[1]LMC still requires task identifiers during training, which we consider as a task-aware method.

[2]total parameters = model parameters + memory parameters in floating point numbers.

TABLE 5.7: Evaluation metrics on the CTrL benchmark, we report the average accuracy ACC(↑) and backward transfer BWT(↑) at the end of training. We organize the methods into task-free (first block) and task-aware (last-block). Task-aware variants are denoted with the (A) suffix, (H) suffix denotes the shared-head variant, * denotes methods that use more parameters. We highlight the methods with best mean metrics in bold, and underline the second best methods

Method	$\mathcal{S}^{-}$		$\mathcal{S}^{+}$		$\mathcal{S}^{\text{in}}$		$\mathcal{S}^{\text{out}}$		$\mathcal{S}^{\text{pl}}$	
	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)
					Task-free					
Finetune	47.5± 1.5	-14.9±1.4	31.4±3.7	-29.3±3.8	39.7±5.0	-23.9±5.7	45.4±4.0	-15.5±3.7	29.1±3.1	-29.2±3.2
Finetune-L	52.1±1.4	-15.7±1.7	38.2±3.2	-25.08±3.3	49.3±2.0	-18.4±2.0	49.3±2.1	-18.4±2.0	37.1±2.1	-26.0±2.2
EWC	62.7±0.7	-3.6±0.9	53.4±1.8	-2.3±0.4	56.3±2.5	-9.1±3.3	**62.5±0.9**	-3.6±0.9	52.3±1.4	-5.7±1.3
O-EWC	62.0±0.7	-3.2±0.7	54.6±0.7	-1.3±1.0	54.2±3.1	-10.8±3.1	62.4±0.4	-3.0±0.9	52.3±1.4	-5.7±1.3
ER	61.5±0.5	-3.1±1.2	59.9±0.5	0.2±0.9	50.0±1.4	-12.6±0.2	51.0±0.5	-6.3±1.2	54.6±2.2	-5.7±2.3
DER++	63.0±0.7	-2.6±0.4	59.9±0.9	0.1±0.9	55.8±2.6	-10.5±1.9	55.4±0.6	-0.9±0.4	58.1±1.7	-3.3±0.6
DualNet	65.5±0.6	-1.8±0.4	61.1±0.7	1.3±0.6	59.1±0.5	-6.8±0.5	55.7±0.4	-0.7±0.6	58.9±2.2	-1.8±0.1
DualNet++	**68.7±0.7**	**-0.1±0.1**	**62.9±0.8**	**2.9±1.4**	**61.6±1.1**	**-5.1±0.6**	57.3±0.9	**-0.6±0.3**	**59.7±1.2**	**0.2±0.4**
					Task-aware					
Independent*	62.7±0.9	0.0	63.2±0.8	0.0	63.1±0.7	0.0	63.1±0.7	0.0	63.9±0.5	0.0
HAT(A)	63.7±0.7	-1.3±0.6	61.4±0.5	-0.2±0.2	50.1±0.8	0.0±0.1	61.9±1.3	-3.2±1.3	61.2±0.7	-0.1±0.2
SG-F(A)	63.6±1.5	0.0	61.5±0.6	0.0	65.5±1.8	0.0	64.1±0.0	0.0	62.0±1.3	0.0
SG-F(A)	29.5±3.5	-35.3±4.0	20.4±4.4	-39.3±6.7	24.4±5.6	-38.7±4.0	30.5±4.5	-34.0±5.5	19.4±1.0	-41.8±1.6
LMC(A)	66.6±1.5	-0.0±0.1	60.1±2.7	-1.4±2.4	**69.5±1.0**	0.0±0.1	66.7±2.2	-0.1±0.1	61.6±4.8	-3.5±3.1
MNTDP(A,H)	66.3±0.8	0.0	62.6±0.8	0.0	63.1±0.7	0.0	63.1±0.7	0.0	63.9±0.5	0.0
MNTDP(A)	41.9±2.5	-2.8±0.6	43.2±1.3	-10.8±2.0	32.7±13.6	-15.2±13.2	37.9±2.7	-5.8±3.5	35.1±3.6	-16.4±4.6
LMC(A,H)	67.2±1.5	-0.5±0.4	62.2±4.5	2.3±1.6	68.5±1.7	-0.1±0.1	55.1±3.4	-7.4±4.0	**63.5±1.9**	-1.0±1.5
LMC(A)	64.9±1.9	-0.2±0.2	55.8±2.5	-0.3±1.2	67.6±2.7	-0.8±1.0	54.2±3.6	-2.9±2.0	53.8±5.7	3.1±5.5
ER(A,H)	62.9±0.4	-0.7±1.1	55.9±1.2	1.7±0.9	54.8±3.2	-4.2±3.7	47.6±1.5	-7.6±1.6	55.6±1.3	-1.2±1.5
DER++(A)	65.3±0.5	0.2±0.7	60.2±1.4	0.9±0.5	59.0±3.8	-3.8±2.0	64.7±1.4	0.4±0.2	58.5±1.2	0.0±1.7
DualNet(A)	68.1±0.7	0.2±0.4	62.2±1.0	**4.6±1.3**	63.8±1.6	-3.4±0.8	67.3±0.7	**0.7±1.5**	59.6±0.9	-1.2±0.1
DualNet++(A)	**69.6±1.1**	**1.3±0.8**	**63.3±0.9**	4.3±0.5	64.1±2.2	**-3.4±1.1**	**68.1±0.5**	**0.7±1.8**	62.2±0.6	**0.1±1.1**

provides task identifiers during both training and evaluation; and (ii) task-free setting where task-identifier are not provided during evaluation. In general, the results show that dynamic architecture methods, especially recent works such as the shared-head variants of MNTDP [199] and LMC [213], can perform competitively on this benchmark and outperforms the static architecture approaches. Among task-free methods, we observe that DualNet and DualNet++ outperform all the baselines considered with only one exception in the

$\mathcal{S}^{\text{out}}$stream where they were outperformed by EWC and O-EWC. Moreover, in most cases, DualNet++ achieves improvements on ACC(↑) and BWT(↑) over DualNet, indicating the benefits of its spatial dropout layers in preventing negative transfer in continual learning. In the task-aware setting, we observe many competitive baselines with high performance. Notably, most baselines in this category are dynamic architecture approaches, which have long been dominating the CTrL benchmark. Nevertheless, DualNet++ can perform favorably and achieve top-2 performances in several streams. To the best of our knowledge, this is the first result showing a static architecture method consistently performing comparable with the dynamic architecture ones on the CTrL benchmarks. Lastly, we highlight that, despite the strong results, dynamic architecture methods in the Task-aware category exhibit high variance on different runs, which arises from them training larger models on a small amount of samples. On the other hand, DualNets perform consistently and have small variance in all cases.

TABLE 5.8: Transfer results on the CTrL benchmark. We report the accuracy of the first and last task of each stream. We also report Δ, the difference between the last task accuracy from the model compared to the reference **Independent** model. We highlight the methods with best mean metrics in bold, and underline the second best methods

Method	$\mathcal{S}^-$			$\mathcal{S}^+$			$\mathcal{S}^{\text{in}}$			$\mathcal{S}^{\text{out}}$		
	ACC T1	ACC T6	Δ	ACC T1	ACC T6	Delta	ACC T1	ACC T6	Δ	ACC T1	ACC T6	Δ
Independent	65.5±0.7	41.8±1.0	0	41.3±2.9	$\underline{65.6\pm0.5}$	0	98.5±0.2	76.9±4.9	0	65.9±0.6	43.5±1.6	0
MNTDP(A)	63.0±3.6	56.9±5.1	15.1	**43.2±0.7**	**65.9±0.8**	**0.3**	**98.9±0.1**	$\underline{93.3\pm1.6}$	16.4	65.0±1.2	57.7±1.7	14.2
LMC	65.2±0.4	60.0±1.1	18.2	42.9±0.9	60.6±1.9	-4.7	$\underline{98.7\pm0.1}$	92.5±7.6	15.6	65.2±0.2	59.8±1.1	16.3
LMC(A,H)	62.2±0.4	63.0±1.7	21.2	$\underline{43.1\pm0.6}$	62.2±0.7	-3.4	$\underline{98.7\pm0.1}$	88.3±1.6	11.4	65.5±0.6	42.0±21.9	-1.5
SG-F(A)	64.9±0.4	49.1±7.3	7.3	$\underline{43.1\pm0.4}$	61.7±1.7	-3.9	98.8±0.1	80.4±6.8	3.5	65.0±0.4	51.5±6.5	8
DER++(A)	68.5±1.5	69.9±0.8	28.1	36.7±2.4	58.8±1.7	-6.8	98.1±0.2	93.4±2.3	16.5	67.4±2.3	66.9±2.0	23.4
DualNet(A)	$\underline{71.8\pm0.8}$	$\underline{71.9\pm0.8}$	**27.2**	41.2±0.4	64.5±0.8	-1.1	**98.9±0.1**	**94.8±2.4**	**17.9**	$\underline{69.6\pm1.8}$	$\underline{68.4\pm1.7}$	$\underline{24.9}$
DualNet++(A)	**72.6±0.9**	**72.8±0.6**	**31.0**	40.0±0.1	64.8±0.8	-0.8	**98.9±0.1**	**94.8±1.5**	**17.9**	**71.0±1.4**	**71.4±1.2**	**27.9**

TABLE 5.9: DualNet++ with different dropout ratio p on the CTrL benchmark using the task-aware evaluation. With the ratio $p = 0.0$, DualNet++ reduces to the standard DualNet. Best mean results are highlighted in bold

DualNet++(A)	$\mathcal{S}^-$		$\mathcal{S}^+$		$\mathcal{S}^{\text{in}}$		$\mathcal{S}^{\text{out}}$		$\mathcal{S}^{\text{pl}}$	
	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)	ACC(↑)	BWT(↑)
$p = 0.0$	68.1±0.7	0.2±0.4	62.2±1.0	4.6±1.3	63.8±1.6	-3.4±0.8	67.3±0.7	0.7±1.5	58.6±0.9	-1.2±0.1
$p = 0.1$	68.8±0.8	0.7±0.3	61.6±1.1	1.3±0.7	63.6±2.0	**4.7±0.6**	**69.1±1.4**	**1.4±0.6**	61.5±1.8	1.4±0.6
$p = 0.2$	**69.6±1.1**	**1.3±0.8**	**63.3±0.9**	**4.3±0.5**	**64.1±2.2**	-3.4±1.1	68.1±0.5	0.7±1.8	**62.2±0.6**	0.1±1.1
$p = 0.3$	66.8±1.2	-1.0±0.7	61.2±0.6	3.5±0.2	63.5±1.6	-2.6±0.7	67.6±0.8	0.4±0.5	61.3±1.4	0.6±0.5
$p = 0.5$	67.1±0.6	-0.7±1.1	60.7±1.0	3.6±0.3	63.2±0.2	-1.8±1.1	67.2±1.2	-0.2±0.7	61.2±0.9	0.7±0.1

Transfer Results on CTrL

We now take a closer look at the transferring capabilities of different methods on the CTrL benchmark. Recall that in the $\mathcal{S}^-$, $\mathcal{S}^+$, $\mathcal{S}^{\text{in}}$, $\mathcal{S}^{\text{out}}$streams, only the first and last tasks are closely related, while the intermediate ones are distractors. Therefore, the ACC(↑) metric alone might not be sufficient to evaluate the model's ability to transfer because it also measures the performance of unrelated tasks. Thus, in these streams, it is helpful to look at the accuracy of the first and last task explicitly [213], and compare them with a reference model, Independent, that trains a separate model for each task. We report the results of this experiment in Table 5.8. We can see that except for the $\mathcal{S}^+$stream, both DualNet and DualNet++ achieve significantly better accuracy on the last task and have a higher differences (Δ) compared to the reference model. On the $\mathcal{S}^+$stream, we observe that DualNets are marginally worse than a few baselines, suggesting that remembering and continuing to learn from a limited representation is challenging for DualNets. This phenomenon is easy to understand since learning good representations from limited data with distractors is a challenging problem. Overall, the results suggest that DualNets can remember long-term knowledge in several cases and learn robust representations to distribution shifts, which facilitates successful continual learning in complex scenarios. Moreover, retaining and continuing learning from limited experiences (the $\mathcal{S}^{+}$stream) presents a promising future research direction.

Robustness to The Dropout Ratio

In literature, the dropout ratio for fully connected layers are commonly set as $p = 0.5$ while this value is set to lower values, e.g. $p = 0.1$ or $p = 0.15$, for convolutional layers [55]. We now investigate how this hyper-parameter affects DualNet++ performance. We consider the CTrL benchmark and report DualNet++(A) with different dropout ratios in Table 5.9.

It is worth noting that by removing the dropout layer (setting $p = 0.0$), DualNet++ reduces to DualNet. The results show that DualNet++ is robust to and beneficial from the small dropout ratios. Specifically, DualNet++ achieves similar performances when $p = 0.1$ and $p = 0.2$, and both can outperform the standard DualNet ($p = 0.0$). When using larger dropout ratio, e.g. $p = 0.3$ and $p = 0.5$, we observe a performance drop on all streams in the CTrL benchmark, which is consistent with the conventional usage of dropout layers in convolutional networks.

5.4.3 Summary of Results

We now provide a summary of the experimental results.

We have conducted experiments on various continual learning settings and examined different scenarios, ranging from the traditional settings to the complex scenarios of semi-supervised learning or complex transfer scenarios. In most cases, DualNet and DualNet++ outperform the baselines, often quite significantly. There is an exception of the $\mathcal{S}^{+}$and $\mathcal{S}^{\text{in}}$streams where the model needs to learn from limited data with different input distributions, which presents an interesting future work. We also found DualNets to be robust to the choice of SSL loss, an benefited from the LA optimizer and more SSL training iterations in the online continual learning setting. Between DualNet and DualNet++, DualNet++ achieves similar performances to DualNet in the controlled environment where labeled data is plentiful. On the other hand, DualNet++ offers significant improvements in scenarios where labeled data is limited (semi-supervised setting) or there exists negative knowledge transfer from unrelated tasks or distribution shifts (CTrL benchmark).

5.5 Conclusion

Inspired by the Complementary Learning System theory, we propose a novel fast-and-slow learning framework for continual learning and conceptualize it into the DualNets paradigm. DualNets (DualNet and DualNet++) comprise two key learning components: (i) a slow learner that focuses on learning a general and task-agnostic representation using the memory data; and (ii) a fast learner focuses on capturing new supervised learning knowledge via a novel adaptation mechanism. Moreover, the fast and slow learners complement each other while working synchronously, resulting in a holistic continual learning method. Our experiments on challenging benchmarks demonstrate the efficacy of DualNets. Lastly, extensive and carefully designed ablation studies show that DualNets are robust the hyper-parameter configurations, scalable with more resources, and can work well in several challenging continual learning scenarios.

Limitations and Future Work Because the DualNet's slow learner can always be trained in the background, it incurs computational costs that need to be properly managed. For large-scale systems, such additional computations can increase infrastructural costs substantially. Therefore, it is important to manage the slow learner to balance between performance and computational overheads. In addition, implementing DualNet to specific applications requires additional considerations to address its inherent challenges. For example, medical image analysis applications may require paying attention to specific regions in the image or considering the data imbalance. However, since we demonstrate the efficacy of DualNet in general settings, such properties are not considered. In practice, it would be more beneficial to capture such domain-specific information to achieve better results. For general applications of DualNet, we also expect that a more suitable objective to train the slow learner can further improve the results. Lastly, through extensive experiments, we identified the scenario of $\mathcal{S}^{\text{in}}$and $\mathcal{S}^{+}$, continual learning from limited data under distribution shifts, to be challenging for DualNets. This suggests a promising future research direction to develop a better, more robust representation learning from limited training samples and can be robust to distribution shifts.

5.6 Implementing DualNets

We provide the pseudo-code of DualNets in Algorithm 3.

Algorithm 3: Psuedo-code to train DualNets.

```
Algorithm TrainDualNet ($\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{D}^{tr}_{1:T}$)
    Require: slow learner $\boldsymbol{\phi}$, fast learner $\boldsymbol{\theta}$, episodic memory $\mathcal{M}$, inner
             updates $N$, Look-ahead inner updates $K$ (for Look-ahead)
    Init: $\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{M} \leftarrow \varnothing$
    for $t \leftarrow 1$ to $T$ do
        for $j \leftarrow 1$ to $n_{batches}$ do    // Receive the dataset $D^{tr}_t$ sequentially
            Receive a mini batch of data $\mathcal{B}_j$ from $\mathcal{D}^{tr}_t$
            $\mathcal{M} \leftarrow$ MemoryUpdate($\mathcal{M}, \mathcal{B}_j$)    // Update the episodic memory
            for $i \leftarrow 1$ to $\infty$ do    // Train the slow learner synchronously
                Train the slow learner using the Look-ahead procedure
            for $n \leftarrow 1$ to $N$ do    // Train the fast learner synchronously
                $\boldsymbol{M}_n \leftarrow$ Sample($\mathcal{M}$)
                $\mathcal{B}_n \leftarrow \boldsymbol{M}_n \cup \mathcal{B}_j$
                SGD update the slow learner: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \nabla_{\boldsymbol{\phi}} \mathcal{L}_{tr}(\mathcal{B}_n)$
                SGD update the fast learner $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}} \mathcal{L}_{tr}(\mathcal{B}_n)$
        $\mathcal{M}^{em}_t \leftarrow \mathcal{M}^{em}_t \cup \{\pi(\hat{y}/\tau)\}$
    return $\boldsymbol{\theta}, \boldsymbol{\phi}$

Procedure Look-ahead($\boldsymbol{\phi}, \mathcal{M}$)
    $\tilde{\boldsymbol{\phi}}_0 \leftarrow \boldsymbol{\phi}$
    for $k \leftarrow 1$ to $K-1$ do
        $\boldsymbol{M}_k \leftarrow$ Sample($\mathcal{M}$) $\cup \mathcal{B}_j$
        Obtains two views of $\boldsymbol{M}_k$ : $\boldsymbol{M}^A_k, \boldsymbol{M}^B_k$
        Calculate the Barlow Twins loss: $\mathcal{L}_{\mathcal{BT}}(\tilde{\phi}_k, \boldsymbol{M}^A_k, \boldsymbol{M}^B_k)$
        SGD update the slow learner: $\tilde{\boldsymbol{\phi}}_{k+1} \leftarrow \boldsymbol{\phi}_k - \epsilon \nabla_{\boldsymbol{\phi}_k} \mathcal{L}_{\mathcal{BT}}$
    Look-ahead update the slow learner: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \beta(\tilde{\boldsymbol{\phi}}_K - \boldsymbol{\phi})$
    return $\boldsymbol{\phi}$
```

Chapter 6

Learning Fast and Slow for Online Time Series Forecasting

In Chapter 4 and 5, we demonstrated the fast-and-slow learning framework for continual learning. Despite encouraging results, previous experiments are mostly conducted on benchmarks derived from standard datasets for batch training. This common practice results in a gap between continual learning researches and how continual learning can actually be applied in other disciplines. To address this limitation, this Chapter investigates a radical approach to online time series forecasting from a continual learning and the CLS theory perspectives. Particularly, we aim to address two important challenges of fast adaptation to concept drifts and facilitate learning of recurring concepts in time series as a continual learning problem. Further improving upon the fast-and-slow learning discussed so far in this dissertation, we propose FSNet as a novel method to effectively forecast time series on the fly. Extensive experiments on real and synthetic datasets suggested that FSNet can quickly learn (require less samples) new concepts on data streams, remember and recall recurring patterns to facilitate their learning in the future.

6.1 Introduction

Time series forecasting plays an important role in both research and industries. Correctly forecast time series can greatly benefit various business sectors such as traffic management and electricity consumption [105]. As a result, tremendous efforts have been devoted to develop better forecasting models [195, 202, 216], with a recent success on deep neural networks [141, 218, 220, 222] thanks to their impressive capabilities to discover hierarchical latent representations and complex dependencies. However, such studies focus on the batch learning setting which requires the whole training dataset to be made available a priori and implies the relationship between the input and outputs remains static over time. This assumption is restrictive in real-world applications such as finance, where data arrives in a stream and the input-output relationship can change quickly [44]. In such cases, re-training the model from scratch could be time consuming. Therefore, it is desirable to train the forecaster online [38, 63] using only new samples to capture the changing dynamic in the environment.

As we have discussed so far in this dissertation, training deep neural networks on data streams remains challenging for two reasons. First, naively train deep neural networks on data streams converges slowly [115, 125] because the offline training benefits such as mini-batches or training for multiple epochs are not available. Moreover, when a distribution shift happens [44], such cumbersome models would require many more training samples to be able to learn such new concepts. Second, time series data often exhibits recurrent patterns where one pattern could become inactive and re-emerge in the future. Since deep networks suffer from the catastrophic forgetting phenomenon [8], they cannot retain prior knowledge and result in inefficient learning of recurring patterns, which further hinders the overall performance. In many time series forecasting applications, the distributions often evolve (change) over time, which makes the previous models outdated and requires us to provide new models with meaningful forecasts in a timely manner. Therefore, deep neural networks, although possess strong representation learning capabilities, lack a mechanism to facilitate successful learning on online time series data.

To address the above limitations, we innovatively formulate online time series forecasting as an *online, task-free continual learning* problem [125, 126]. Particularly, continual learning requires balancing two objectives: (i) utilizing past knowledge to facilitate fast learning of current patterns; and (ii) maintaining and updating the already acquired knowledge. These two objectives closely match the aforementioned challenges and are usually referred to as the *stability-plasticity* dilemma [4]. With this connection, we develop an effective online time series forecasting framework motivated by the *Complementary Learning Systems (CLS) theory* [14, 62], a neuroscience framework for human continual learning. Specifically, the CLS theory suggests that humans can continually learn thanks to the interactions between the *hippocampus* and the *neocortex*, which supports the consolidation, recall, and update such experiences to form a more general representation, which supports generalization to new experiences.

This Chapter develops FSNet (Fast-and-Slow learning Network) to enhance the sample efficiency of deep networks when learning with time series data streams, especially when dealing with distribution shifts or recurring concepts. FSNet's key idea for fast learning is that it does not explicitly detect distribution shifts but instead always improving the learning of current samples. To do so, FSNet employs a per-layer adapter to model the temporal consistency in time series and adjust each intermediate layer to learn better, which in turn improve the learning of the whole deep network. In addition, FSNet further employs an associative memory [75] to store important, recurring patterns observed during training. When encountering repeating events, the adapter interacts with its memory to retrieve and update the previous actions to facilitate fast learning of such patterns. Consequently, the adapter can model the temporal smoothness in time series to facilitate learning while its interactions with the associative memory allows each layer to remember and quickly improve the learning of recurring patterns.

In summary, our work makes the following contributions. First, we innovatively formulate learning fast in online time series forecasting with deep models as a continual learning problem. Second, motivated by the CLS theory for continual learning, we propose a fast-and-slow learning paradigm of FSNet to handle both the fast changing and long-term knowledge in time series. Lastly, we conduct extensive experiments with both real and synthetic datasets to demonstrate FSNet's efficacy and robustness.

6.2 Proposed Framework

This Section summarizes the background of time series forecasting and presents our FSNet framework.

6.2.1 Time Series Forecasting Settings

Let $\mathcal{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \in \mathbb{R}^{T \times n}$ be a time series of T observations, each has n dimensions. The goal of time series forecasting is that given a look-back window of length e, ending at time i: $\mathcal{X}_{i,e} = (\boldsymbol{x}_{i-e+1}, \ldots, \boldsymbol{x}_i)$, predict the next H steps of the time series as $f_\omega(\mathcal{X}_{i,H}) = (\boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_{i+H})$, where ω denotes the parameter of the forecasting model. We refer to a pair of look-back and forecast windows as a sample. For multiple-step forecasting ($H > 1$) we follow the standard approach of employing a linear regressor to forecast all H steps in the horizon simultaneously [222]. Particularly, the linear regressor is a mapping $\mathbb{R}^d \to \mathbb{R}^{n\times H}$ that maps a $d-$dimensional feature vector (i.e. output from the penultimate layer of a deep neural network) to the values of all dimensions in the time series H steps in the future.

Online Time Series Forecasting Online time series forecasting is ubiquitous is many real-world scenarios [38, 63, 102, 127] due to the sequential nature of data. In this setting, there is no separation of training and evaluation. Instead, learning occurs over a sequence of rounds. At each round, the model receives a look-back window and predicts the forecast window. Then, the true answer is revealed to improve the model's predictions of the incoming rounds [135]. The model is commonly evaluated by its accumulated errors throughout learning [115]. Due to its challenging nature, online time series forecasting exhibits several challenging sub-problems, ranging from learning under concept drifts [44], to dealing with missing values because of the irregularly-sampled data [186, 211]. In this work, we focus on the problem of fast learning (in terms of sample efficiency) under concept drifts by improving the deep network's architecture and recalling relevant past knowledge.

There is also a rich literature of Bayesian continual learning to address regression problems [24, 140, 211]. However, such formulation follow the Bayesian framework, which allows for forgetting of past knowledge and does not have an explicit mechanism for fast learning [104, 107]. Moreover, such methods were not developed with deep neural networks and it is non-trivial to extend such methods to the setting of our study.

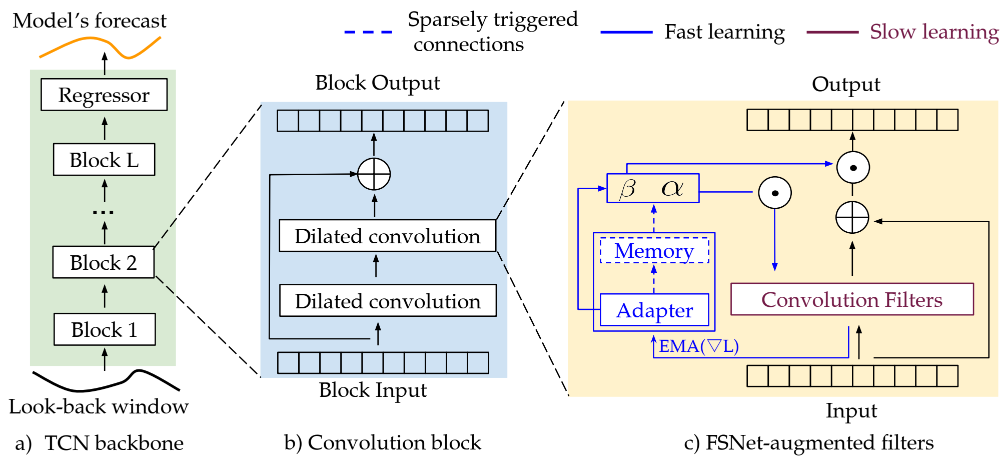


FIGURE 6.1: An overview of FSNet. a) A standard TCN backbone (green-shared) of L dilated convolution stacks (-shaded). b) A stack of convolution filters (yellow-shaded). c) Each convolution filter in FSNet is equipped with an adapter and and associative memory to facilitate fast adaptation to both old and new patterns by monitoring the backbone's gradient EMA. Best viewed in colors.

6.2.2 Online Time Series Forecasting as a Continual Learning Problem

Our formulation is motivated by the locally stationary stochastic processes observation, where a time series can be split into a sequence of stationary segments [37, 34, 58]. Since the same underlying process generates samples from a stationary segment, we refer to forecasting each stationary segment as a learning task for continual learning. We note that this formulation is general and encompasses existing learning paradigms. For example, splitting into only one segment indicates no concept drifts, and learning reduces to online learning in stationary environments [135]. Online continual learning [125] corresponds to the case of there are at least two segments. Moreover, we also do not assume that the time points of task switch are given to the model, which is a common setting in many continual learning studies [76, 80]. Manually obtaining such information in real-world time series can be expensive because of the missing or irregularly sampled data [186, 210]. Therefore, while we assume that the data comprises several tasks, the task-changing points are not provided by the environment, which corresponds to the online, task-free continual learning formulation [125, 126, 181, 205].

We now discuss the differences between our formulation with existing studies. First, most existing task-free continual learning frameworks [126, 215] are developed for image data, which vastly differs from time series. The input and label spaces of images are different (continuous vs discrete) while time series' input and output share the same real-valued space. Additionally, the image's label changes significantly across tasks while time series data changes gradually over time with no clear boundary. Moreover, time series exhibits strong temporal information among consecutive samples, which does not exist in image data. Therefore, it is non-trivial to simply apply existing continual learning methods to time series and successful solutions requires carefully handling unique characteristics from time series data.

Second, time series evolves and old patterns may not reappear exactly in the future. Thus, we are not interested in remembering old patterns precisely but predicting **how they will evolve**. *For example, we do not need to predict the electricity consumption over the last winter. But it is more important to predict the electricity consumption this winter, assuming that it is likely to have a similar pattern as the last one.* Therefore, we do not need a separate test set for evaluation, but training follows the online learning setting where a model is evaluated by its accumulated errors throughout learning.

6.2.3 Fast and Slow Learning Networks (FSNet)

FSNet always leverages past knowledge to improve the learning in the future (Section 6.2.3), which is akin to facilitating forward transfer in continual learning [80]. Additionally, FSNet remembers repeating events and continue to learn them when they reappear (Section 6.2.3, which is akin to preventing catastrophic forgetting [76].

We consider Temporal Convolutional Network (TCN) [93] as the backbone deep neural network to extract a time series feature representation due to the simple forward architecture and promising results [220]. The backbone has L layer with parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_l\}_{l=1}^L$. FSNet improves the TCN backbone with *two* complementary components: a per-layer adapter $\boldsymbol{\phi}_l$ and a per-layer associative memory $\mathcal{M}_l$. Thus, the total *trainable* parameters is $\boldsymbol{\omega} = \{\boldsymbol{\theta}_l, \boldsymbol{\phi}_l\}_{l=1}^L$ and the total associative memory is $\mathcal{M} = \{\mathcal{M}_l\}_{l=1}^L$. We also use $\boldsymbol{h}_l$ and $\tilde{\boldsymbol{h}}_l$ to denote the original feature and adapter feature map of the $l-$layer. Figure 6.1 provides an illustration of FSNet.

Fast-adaptation Mechanism

The key observation allowing for a fast learning is to facilitate the learning of each intermediate layer via the following observation: *the partial derivative $\nabla_{\boldsymbol{\theta}_l}\ell$ characterizes the contribution of layer $\boldsymbol{\theta}_l$ to the forecasting loss ℓ.* Traditional training schemes simply move the parameters along this gradient direction, which results in ineffective online learning [115, 152]. Moreover, time series data exhibits strong temporal consistency across consecutive samples, which is not captured by existing training frameworks. Putting these observations together, we argue that an exponential moving average (EMA) of the partial derivative can provide meaningful information about the temporal smoothness in time series. Consequently, leveraging this knowledge can improve the learning of each layer, which in turn improves the whole network's performance.

To utilize the gradient EMA, we propose to treat it as a context to support fast learning via the feature-wise transformation framework [112, 98, 214, 219]. Particularly, we propose to equip each layer with an adapter to map the layer's gradient EMA to a set of smaller, more compact transformation coefficients. These coefficients are applied on the corresponding layer's parameters and feature so that they can leverage the temporal consistency to learn better. We first define the EMA of the $l-$layer's partial derivative as:

$$\hat{\boldsymbol{g}}_l \leftarrow \gamma \hat{\boldsymbol{g}}_l + (1-\gamma)\boldsymbol{g}_l^t, \tag{6.1}$$

where $\boldsymbol{g}_l^t$ denotes the gradient of the $l-$th layer at time t and $\hat{\boldsymbol{g}}_l$ denotes the EMA. The adapter takes $\hat{\boldsymbol{g}}_l$ as input and maps it to the adaptation coefficients $\boldsymbol{u}_l$. Then, an adapter for the $l-$th layer is a linear layer that maps the context $\hat{\boldsymbol{g}}_l$ to a set of transformation coefficients $\boldsymbol{u}_l = [\boldsymbol{\alpha}_l; \boldsymbol{\beta}_l]$. In this work, we consider a two-stage transformations [219] which involve a weight and bias transformation coefficients $\boldsymbol{\alpha}_l$ and a feature transformation coefficients $\boldsymbol{\beta}_l$.

The adaptation process for a layer $\boldsymbol{\theta}_l$ is summarized as:

$$\begin{aligned}
[\boldsymbol{\alpha}_l, \boldsymbol{\beta}_l] =& \boldsymbol{u}_l, \text{ where } \boldsymbol{u}_l = \boldsymbol{\Omega}(\hat{\boldsymbol{g}}_l; \boldsymbol{\phi}_l) && (6.2)\\
\text{Weight adaptation: } \tilde{\boldsymbol{\theta}}_l =& \text{tile}(\boldsymbol{\alpha}_l) \odot \boldsymbol{\theta}_l, \text{ and} && (6.3)\\
\text{Feature adaptation: } \tilde{\boldsymbol{h}}_l =& \text{tile}(\boldsymbol{\beta}_l) \odot \boldsymbol{h}_l, \text{ where } \boldsymbol{h}_l = \tilde{\boldsymbol{\theta}}_l \circledast \tilde{\boldsymbol{h}}_{l-1}. && (6.4)
\end{aligned}$$

Here, $\boldsymbol{h}_l$ is a stack of I features maps with C channels and length Z, $\tilde{\boldsymbol{h}}_l$ is the adapted feature, $\tilde{\boldsymbol{\theta}}_l$ denotes the adapted weight, $\odot$ denotes the element-wise multiplication, and tile($\boldsymbol{\alpha}_l$) denotes that the weight adaptor is applied per-channel on all filters via a tile function that repeats a vector along the new axes.

Chunking Operation we now describe the chunking adapter's chunking operation to efficiently compute the adaptation coefficients. For convenient, we denote $\mathrm{vec}(\cdot)$ as a vectorizing operation that flattens a tensor into a vector; we use $\mathrm{split}(\boldsymbol{e}, B)$ to denote splitting a vector $\boldsymbol{e}$ into B segments, each has size $\dim(\boldsymbol{e})/B$. An adapter maps its backbone's layer EMA gradient to an adaptation coefficient $\boldsymbol{u} \in \mathbb{R}^d$ via the chunking process as:

$$\begin{aligned} \hat{\boldsymbol{g}}_l \leftarrow & \mathrm{vec}(\hat{\boldsymbol{g}}_l) \\ [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots \boldsymbol{b}_d] \leftarrow & \mathrm{reshape}(\hat{\boldsymbol{g}}_l; d) \\ [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_d] \leftarrow & [\boldsymbol{W}_{\boldsymbol{\phi}}^{(1)} \boldsymbol{b}_1, \boldsymbol{W}_{\boldsymbol{\phi}}^{(1)} \boldsymbol{b}_2, \ldots, \boldsymbol{W}_{\boldsymbol{\phi}}^{(1)} \boldsymbol{b}_d] \\ [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_d] \leftarrow & [\boldsymbol{W}_{\boldsymbol{\phi}}^{(2)} \boldsymbol{h}_1, \boldsymbol{W}_{\boldsymbol{\phi}}^{(2)} \boldsymbol{h}_2, \ldots, \boldsymbol{W}_{\boldsymbol{\phi}}^{(2)} \boldsymbol{h}_d]. \end{aligned}$$

Where we denote $\boldsymbol{W}_{\boldsymbol{\phi}}^{(1)}$ and $\boldsymbol{W}_{\boldsymbol{\phi}}^{(2)}$ as the first and second weight matrix of the adapter. In summary, the chunking process can be summarized by the following steps: (1) flatten the gradient EMA into a vector; (2) split the gradient vector into d chunks; (3) map each chunk to a hidden representation; and (4) map each hidden representation to a coordinate of the target adaptation parameter $\boldsymbol{u}$.

Remembering Recurring Events with an Associative Memory

In time series, old patterns may reappear in the future and it is imperative to leverage our past actions improve the learning outcomes. In FSNet, an adaptation to a pattern is represented by the coefficients $\boldsymbol{u}$, which we argue to be useful to learn repeating events. Specifically, $\boldsymbol{u}$ represents how we adapted to a particular pattern in the past; thus, storing and retrieving the appropriate $\boldsymbol{u}$

may facilitate learning the corresponding pattern when they reappear in the future. Therefore, as the second key element in FSNet, we implement an associative memory to store the adaptation coefficients of repeating events encountered during learning. Consequently, the adapter alone can handle fast changes over a short time scale, while the associative memory can facilitate learning of repeating patterns. In summary, we equip each adapter with an associative memory $\mathcal{M}_l \in \mathbb{R}^{N \times d}$ where d denotes the dimensionality of $\boldsymbol{u}_l$, and N denotes the number of elements, which we fix as $N = 32$ by default.

Sparse Adapter-Memory Interactions Interacting with the memory at every step is expensive and susceptible to noises. Thus, we propose to trigger this interaction only when a substantial representation change happens. Interference between the current and past representations can be characterized in terms of a dot product between the gradients [80, 156]. As a result, we propose to trigger the memory interaction only when the representation has changed significantly, which can be detected by monitoring the cosine similarity between the recent and long-term gradients. To this end, in addition to the gradient EMA in Equation 6.2, we deploy another gradient EMA $\hat{\boldsymbol{g}}_l'$ with a smaller coefficient $\gamma' < \gamma$ and measure their cosine similarity to trigger the memory interaction as:

$$\text{Trigger if} : \cos(\hat{\boldsymbol{g}}_l, \hat{\boldsymbol{g}}_l') = \frac{\hat{\boldsymbol{g}}_l \cdot \hat{\boldsymbol{g}}_l'}{||\hat{\boldsymbol{g}}_l||\,||\hat{\boldsymbol{g}}_l||} < -\tau, \tag{6.5}$$

where $\tau > 0$ is a hyper-parameter determining the significant degree of interference. Moreover, we want to set τ to a relatively high value (e.g. 0.7) so that the memory only remembers significant changing patterns, which could be important and may reappear.

The Adapter-Memory Interacting Mechanism Since the current adaptation coefficients may not capture the whole event, which could span over a few samples, we perform the memory read and write operations using the adaptation coefficients's EMA (with coefficient γ') to fully capture the current pattern. The EMA of $\boldsymbol{u}_l$ is calculated in the same manner as Equation 6.1. When a memory interaction is triggered, the adapter queries and retrieves the most similar transformations in the past via an attention read operation, which is a weighted sum over the memory items:

1. Attention calculation: $\boldsymbol{r}_l = \mathrm{softmax}(\mathcal{M}_l \hat{\boldsymbol{u}}_l)$;

2. Top-k selection: $\boldsymbol{r}_l^{(k)} = \mathrm{TopK}(\boldsymbol{r}_l)$;

3. Retrieval: $\tilde{\boldsymbol{u}}_l = \sum_{i=1}^{K} \boldsymbol{r}_l^{(k)}[i] \mathcal{M}_l[i]$,

where $\boldsymbol{r}^{(k)}[i]$ denotes the i-th element of $\boldsymbol{r}_l^{(k)}$ and $\mathcal{M}_l[i]$ denotes the i-th row of $\mathcal{M}_l$. Since the memory could store conflicting patterns, we employ a sparse attention by retrieving the top-k most relevant memory items, which we fix as $k = 2$. The retrieved adaptation coefficient characterizes old experiences in adapting to the current pattern in the past and can improve learning at the present time by weighted summing with the current parameters as

$$\boldsymbol{u}_l \leftarrow \tau \boldsymbol{u}_l + (1-\tau)\tilde{\boldsymbol{u}}_t, \tag{6.6}$$

where we use the same threshold value τ to determine the sparse memory interaction and the weighted sum of the adaptation coefficients. Then we perform a write operation to update and accumulate the knowledge stored in $\mathcal{M}_l$:

$$\mathcal{M}_l \leftarrow \tau \mathcal{M}_l + (1-\tau)\hat{\boldsymbol{u}}_l \otimes \boldsymbol{r}_l^{(k)} \text{ and } \mathcal{M}_l \leftarrow \frac{\mathcal{M}_l}{\max(1, ||\mathcal{M}_l||_2)}, \tag{6.7}$$

where $\otimes$ denotes the outer-product operator, which allows us to efficiently write the new knowledge to the most relevant locations indicated by $\boldsymbol{r}_l^{(k)}$ [65, 75]. The memory is then normalized to avoid its values scaling exponentially.

Remarks on FSNet's fast learning mechanism FSNet facilitate online time series forecasting by always learning fast at the current time step, regardless of a distribution shift happens or not. To do so, FSNet employs the adapters to model the temporal smoothness in time series and an associative memory to remember recurring events. Since time series always evolves, it is more desirable for the memory to support the read and write operations so that the model could leverage its experiences in the past to further improve the current learning. The episodic memory used in Chapters 3, 4, and 5 does not support these operations and requires storing the original data, which might be prohibited in many applications due to privacy concerns. The associative memory used in this Chapter is one of a simple data structures that satisfy these constraints, and has been successfully applied in other applications such as RL [163] and associative recall [46]. Exploring more sophisticated relational memory structures [185] would be an interesting future research direction.

6.3 Experiments

Our experiments aim at investigating the following hypotheses: (i) FSNet facilitates faster adaptation to both new and recurring concepts compared to existing strategies; (ii) FSNet achieves faster and better convergence than other methods; and (iii) modeling the partial derivative is the key ingredients for fast adaptation.

6.3.1 Experimental Settings

Datasets We explore a wide range of time series forecasting datasets. **ETT**[1] [222] records the target value of "oil temperature" and 6 power load features over a period of two years. We consider the ETTh2 and ETTm1 benchmarks where the observations are recorded hourly and in 15-minutes intervals respectively. **ECL** (Electricty Consuming Load)[2] collects the electricity consumption of 321 clients from 2012 to 2014. **Traffic**[3] records the road occupancy rates at San Francisco Bay area freeways. **Weather**[4] records 11 climate features from nearly 1,600 locations in the U.S in an hour intervals from 2010 to 2013.

We also construct two synthetic datasets to explicitly test the model's ability to deal with new and recurring concept drifts. We synthesize a task by sampling $1,000$ samples from a first-order autoregressive process with coefficient φ: $\mathrm{AR}_{\varphi}(1)$, where different tasks correspond to different φ values. The first synthetic data, **S-Abrupt (S-A)**, contains abrupt, and recurrent concepts where the samples abruptly switch from one AR process to another by the following order: $\mathrm{AR}_{0.1}(1)$, $\mathrm{AR}_{0.4}(1)$, $\mathrm{AR}_{0.6}(1)$, $\mathrm{AR}_{0.1}(1)$, $\mathrm{AR}_{0.3}(1)$, $\mathrm{AR}_{0.6}(1)$. The second data, **S-Gradual (S-G)** contains gradual, incremental shifts, where the shift starts at the last 20% of each task. In this scenario, the last 20% samples of a task is an averaged of two AR process with the order as above. Note that we randomly chose the values of φ so that these datasets do not give unfair advantages to any methods.

Baselines We consider a suite of baselines from continual learning, time series forecasting, and online learning. First, the **OnlineTCN** strategy that simply trains continuously [25]. Second, we consider the Experience Replay (**ER**) [12,

[1] `https://github.com/zhouhaoyi/ETDataset`
[2] `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`
[3] `https://pems.dot.ca.gov/`
[4] `https://www.ncei.noaa.gov/data/local-climatological-data/`

130] strategy where a buffer is employed to store previous data and interleave old samples during the learning of newer ones. We also include three recent advanced variants of ER. First, **TFCL** [126] introduces a task-boundaries detection mechanism and a knowledge consolidation strategy by regularizing the networks' outputs [91]. Second, **MIR** [125] replace the random sampling in ER by selecting samples that cause the most forgetting. Lastly, **DER++** [170] augments the standard ER with a knowledge distillation strategy [50]. We emphasize that ER and and its variants are strong baselines in the online setting since they enjoy the benefits of training on mini-batches, which greatly reduce noises from singe samples and offer faster, better convergence [18]. While the aforementioned baselines use a TCN backbone, we also include **Informer** [222], a recent time series forecasting method based on the transformer architecture [86]. Lastly, we include the historical inertia (**HI**) baseline [207] that simply uses the last observations in the look-back window as its prediction of the forecast window. Very recently, HI was found to perform competitive in the batch training of time series forecasting. We remind the readers that online time series forecasting have not been widely studied with deep models, therefore, we include general strategies from related fields that we inspired from. Such baselines are competitive and yet general enough to extend to our problem.

Implementation Details We split the data into warm-up and online training phases by the ratio of 25:75 and consider the TCN backbone [220] for experiments, except the Informer baseline. We follow the optimization details in [222] by optimizing the ℓ_2 (MSE) loss with the AdamW optimizer [81]. Both the epoch and batch size are set to **one** to follow the online learning setting. We implement a fair comparison by making sure that all baselines use the same total memory budget as our FSNet, which includes *three-times* the network sizes: one working model and two EMA of its gradient. Thus, for ER, MIR, and DER++,

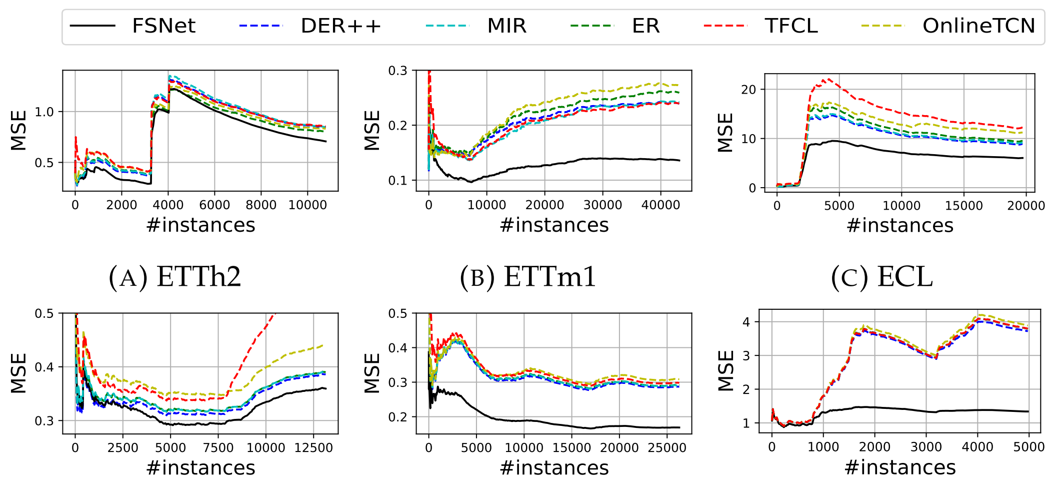
TABLE 6.1: Final cumulative MSE and MAE of algorithms at the end of learning. “†" indicates a transformer backbone, “-" indicates the model did not converge. S-A: S-Abrupt, S-G: S-Gradual. Best results are in bold.

Method		FSNet		DER++[170]		MIR[125]		ER[130]		TFCL[126]		OnlineTCN		Informer [222]		HI [207]	
	H	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh2	1	0.466	0.368	0.508	0.375	0.486	0.410	0.508	0.376	0.557	0.472	0.502	0.436	7.571	0.850	**0.404**	**0.336**
	24	**0.687**	**0.467**	0.828	0.540	0.812	0.541	0.808	0.543	0.846	0.548	0.830	0.547	4.629	0.668	2.561	0.656
	48	**0.846**	**0.515**	1.157	0.577	1.103	0.565	1.136	0.571	1.208	0.592	1.183	0.589	5.692	0.752	4.547	0.836
ETTm1	1	0.105	0.188	**0.098**	**0.183**	0.099	0.184	0.099	0.184	0.099	0.185	0.109	0.204	0.456	0.392	0.120	0.193
	24	**0.136**	**0.248**	0.239	0.329	0.242	0.335	0.259	0.346	0.239	0.335	0.272	0.361	0.827	0.551	3.027	1.010
	48	**0.129**	**0.245**	0.264	0.355	0.271	0.362	0.288	0.372	0.242	0.344	0.280	0.371	0.853	0.533	4.068	1.212
ECL	1	3.143	0.472	**2.657**	0.421	2.575	0.504	2.579	0.506	2.732	0.524	3.309	0.635	-	-	6.747	**0.376**
	24	6.051	0.997	8.996	1.035	9.265	1.066	9.327	1.057	12.094	1.256	11.339	1.196	-	-	**5.998**	**0.371**
	48	**7.034**	1.061	9.009	1.048	9.411	1.079	9.685	1.074	12.110	1.303	11.534	1.235	-	-	7.233	**0.447**
Traffic	1	**0.288**	**0.253**	0.289	0.248	0.290	0.251	0.291	0.252	0.323	0.273	0.315	0.283	0.795	0.507	0.483	0.328
	24	**0.362**	**0.288**	0.387	0.295	0.391	0.302	0.391	0.302	0.553	0.383	0.452	0.363	1.267	0.750	0.798	0.367
WTH	1	**0.162**	**0.216**	0.174	0.235	0.179	0.244	0.180	0.244	0.177	0.240	0.206	0.276	0.426	0.458	0.207	0.188
	24	**0.188**	**0.276**	0.287	0.351	0.291	0.355	0.293	0.356	0.301	0.363	0.308	0.367	0.370	0.417	0.638	0.499
	48	**0.223**	**0.301**	0.294	0.359	0.297	0.361	0.297	0.363	0.323	0.382	0.302	0.362	0.367	0.419	0.828	0.606
S-A	1	1.391	0.929	2.334	1.181	2.482	1.213	2.372	1.157	2.321	1.144	2.668	1.216	3.690	1.410	**1.033**	**0.812**
	24	**1.299**	**0.904**	3.598	1.439	3.662	1.450	3.375	1.360	3.415	1.366	3.904	1.491	3.657	1.426	7.783	2.072
S-G	1	1.760	1.038	2.335	1.181	2.482	1.213	2.476	1.212	2.428	1.199	2.927	1.304	4.024	1.501	**1.066**	**0.822**
	24	**1.299**	**0.904**	3.598	1.439	3.662	1.450	3.667	1.489	3.829	1.479	3.904	1.491	3.657	1.426	7.783	2.072

we allow an episodic memory to store previous samples to meet this budget. For the remaining baselines, we instead increased the backbone size. Lastly, in the warm-up phase, we calculate the mean and standard deviation to normalize online training samples and perform hyper-parameter cross-validation. For all benchmarks, we set the look-back window length to be 60 and vary the forecast horizon as $H \in \{1, 24, 48\}$.

6.3.2 Online Forecasting Results

Cumulative Performance Table 6.1 reports the cumulative mean-squared errors (MSE) and mean-absolute errors (MAE) at the end of training. The reported numbers are averaged over five runs. We observe that ER and its variants (MIR, DER++) are strong competitors and can significantly improve over



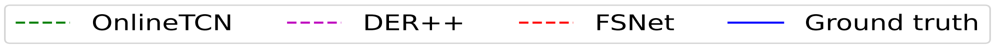
(A) ETTh2 (B) ETTm1 (C) ECL

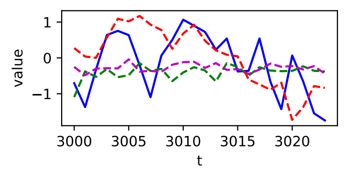
(D) Traffic (E) WTH (F) S-G

FIGURE 6.2: Evolution of the cumulative MSE loss during training with forecasting window $H = 24$.

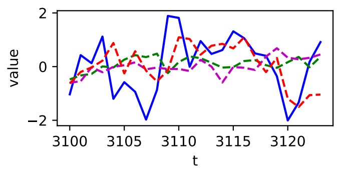
the simple TCN strategies. However, such methods still cannot work well under multiple task switches (S-Abrupt). Moreover, no clear task boundaries (S-Gradual) presents an even more challenging problem and increases most models' errors. In addition, previous work has observed that TCN can outperform Informer in the standard time series forecasting [225]. Here we also observe similar results that Informer does not perform well in the online setting, and is outperformed by other baselines. HI is also a strong competitor when the forecast window is $H = 1$. However, its performance quickly degrades on all datasets with longer forecast windows. On the other hand, our FSNet shows promising results on all datasets and outperforms most competing baselines across different settings. Moreover, FSNet's improvements on the synthetic datasets indicate its ability to quickly adapt to the non-stationary environment and recall previous knowledge, even without clear task boundaries.

Convergent behaviors of Different Learning Strategies Figure 6.2 reports the convergent behaviors on the considered methods. We omit the S-Abrupt

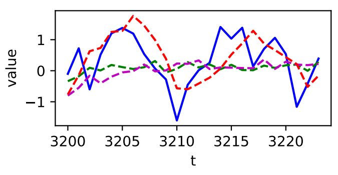




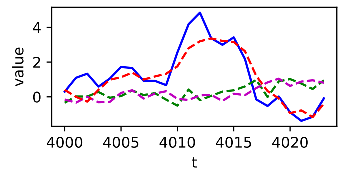
(A) Prediction from $t = 3000$



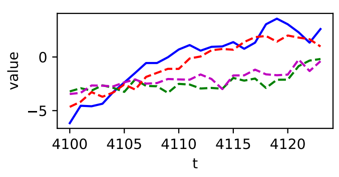
(B) Prediction from $t = 3100$



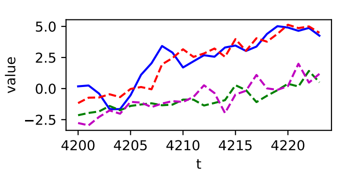
(C) Prediction from $t = 3200$



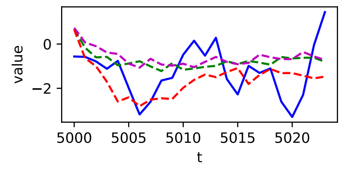
(D) Prediction from $t = 4000$



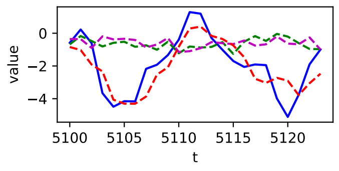
(E) Prediction from $t = 4100$



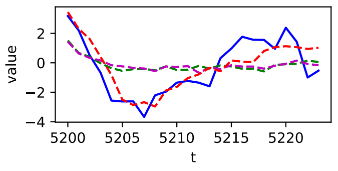
(F) Prediction from $t = 4200$



(G) Prediction from $t = 5000$



(H) Prediction from $t = 5100$



(I) Prediction from $t = 5200$

FIGURE 6.3: Visualization of the model's prediction throughout the online learning process. We focus on a short horizon of 200 time steps after a concept drift, which is critical for fast learning.

dataset for spaces because we observe the same behavior as S-Gradual. The results clearly show the benefits of ER by offering faster convergence during learning compared to OnlineTCN. However, it is important to note that storing the original data may not apply in many domains. On S-Gradual, most baselines demonstrate the inability to quickly recover from concept drifts, indicated by the increasing trend in the error curves. We also observe promising results of FSNet on most datasets, with significant improvements over the baselines on the ETT, WTH, and S-Gradual datasets. The remaining datasets are more challenging with missing values [141] and large magnitude varying *within and* *across* dimensions, which may require calculating better data normalization statistics. While FSNet achieved encouraging results, handling the above challenges can further improve its performance. Overall, the results shed light on the challenges of online time series forecasting and demonstrate promising results of FSNet.

Visualization We explore the model's prediction quality on the S-Abrupt since it is a univariate time series. The remaining real-world datasets are multivariate, thus challenging to visualize. Particularly, we are interested in the models' behaviour when an old task's reappear. Therefore, in Figure 6.3, we plot the model's forecasting at various time points after $t = 3000$. We can see the difficulties of training deep neural networks online in that the model struggles to learn at the early stages, where it only observed a few samples. We focus on the early stages of task switches (e.g. the first 200 samples), which requires the model to quickly adapt to the distribution shifts. With the limited samples per task and the presence of multiple concept drifts, the standard online optimization collapsed to a naive solution of predicting random noises around zero. However, FSNet can successfully capture the time series' patterns and provide better forecasts as learning progresses. Overall, we can clearly see FSNet can provide better quality forecasts compared to other baselines.

6.3.3 Ablation Studies of FSNet's Design

This experiment analyzes the contribution of each FSNet's component. First, we explore the benefits of using the associative memory (Section 6.2.3) by constructing a *No Memory* variant that only uses an adapter, without the memory. Second, we further remove the adapter, which results in the *Naive* variant that directly trains the adaptation coefficients $\boldsymbol{u}$ jointly with the backbone, which corresponds to the NCCL design [219]. The Naive variant demonstrates the

TABLE 6.2: Final comulative MSE and MAE of different FSNet variants. Best results are in bold.

Method		FSNet				Variant			
		M=128 (large)		M=32 (original)		No Memory		Naive	
Data	H	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh2	24	**0.616**	**0.456**	0.687	0.467	0.689	0.468	0.860	0.555
	48	**0.846**	**0.513**	**0.846**	0.515	0.924	0.526	0.973	0.570
Traffic	1	**0.285**	**0.251**	0.288	0.253	0.294	0.252	0.330	0.282
	24	0.358	0.285	0.362	0.288	**0.355**	**0.284**	0.463	0.362
S-A	1	**1.388**	**0.928**	1.391	0.929	1.734	1.024	3.318	1.416
	24	**1.213**	**0.870**	1.299	0.904	1.390	0.933	3.727	1.467
S-G	1	**1.758**	1.040	1.760	**1.038**	1.734	1.024	3.318	1.414
	24	**1.293**	**0.902**	1.299	0.904	1.415	0.940	3.748	1.478

benefits of monitoring the layer's gradients, our key idea for fast adaptation (Section 6.2.3). Lastly, we explore FSNet's scalability by increasing the associative memory size from 32 items (original) to a larger scale of 128 items.

We report the results in Table 6.2. We first observe that FSNet achieves similar results with the No Memory variant on the Traffic and S-Gradual datasets. One possible reason is the insignificant representation interference in the Traffic dataset and the slowly changing representations in the S-Gradual dataset. In such cases, the representation changes can be easily captured by the adapter alone and may not trigger the memory interactions. In contrast, on ETTh2 and S-Abrupt, which may have sudden drifts, we clearly observe the benefits of storing and recalling the model's past action to facilitate learning of repeating events. Second, the Naive variant does not achieve satisfactory results, indicating the benefits of modeling the temporal smoothness in time series via the use of gradient EMA. Lastly, the large memory variant of FSNet provides improvements in most cases, indicating FSNet's scalability with more budget. Overall, these results demonstrated the complementary of each FSNet's components to

TABLE 6.3: Results of different FSNet's hyper-parameter configurations on the ETTh2 ($H = 48$) and S-A ($H = 24$) benchmarks.

Configuration			ETTh2		S-A	
γ	γ'	τ	MSE	MAE	MSE	MAE
0.9	0.3	0.75	0.846	0.515	1.760	1.038
0.9	0.4	0.8	0.860	0.521	1.816	1.086
0.99	0.4	0.7	0.847	0.512	1.791	1.049
0.99	0.3	0.8	0.845	0.514	1.777	1.042

deal with different types of concept drift in time series.

6.3.4 Robustness of Hyper-parameter Settings

This experiment explores the robustness of FSNet to different hyper-parameter setting. Particularly, we focus on the configuration of *three* hyper-parameters: (i) the gradient EMA γ; (ii) the short-term gradient EMA γ'; and (iii) the associative memory activation threshold τ. In general, we provide two guidelines to reduce the search space of these hyper-parameters: (i) setting γ to a high value (e.g. 0.9) and γ' to a small value (e.g. 0.3 or 0.4); (ii) set τ to be relatively high (e.g. 0.75). We report the results of several hyper-parameter configurations in Table 6.3. We observe that there are not significant differences among these configurations . It is also worth noting that we use the same configuration for all experiments conducted in this work. Therefore, we can conclude that FSNet is robust to these configurations.

6.3.5 FSNet and Experience Replay

This experiment explore the complementarity between FSNet and experience replay (ER). We hypothesize that ER is a valuable component when learning on

TABLE 6.4: Performance of FSNet with and without experience replay.

Data	H	FSNet		FSNet+ER	
		MSE	MAE	MSE	MAE
ETTh2	1	0.466	0.368	**0.434**	**0.361**
	24	0.687	0.467	**0.650**	**0.462**
	48	0.846	0.515	**0.842**	**0.511**
Traffic	1	0.321	0.26	**0.243**	**0.248**
	24	0.421	0.312	**0.350**	**0.275**

data streams because it introduces the benefits of mini-batch training to online learning.

We implement a variant of FSNet with an episodic memory for experience replay and report its performance in Table 6.4. We can see that FSNet+ER outperforms FSNet in all cases, indicating the benefits of ER, even to FSNet. However, it is important that using ER will introduce additional memory complexity and that scales with the look-back window. Lastly, in many real-world applications, storing previous data samples might be prohibited due to privacy concerns.

6.3.6 Complexity Analysis

In this Section, we analyze the memory and time complexity of FSNet.

Asymptotic analysis We consider the TCN forecaster used throughout this work and analyze the *model, total memory, and time* complexities of the methods considered in our work. We let N denotes the number of parameters of the the convolutional layers, E denotes the length of the look-back window, and H denotes the length of the forecast window.

TABLE 6.5: Summary of the model and total memory complexity of different methods.N denotes the number parameters of the convolutional layers, H and E denotes the look-back and forecast windows length

Method	OnlineTCN	ER	MIR	DER++	FSNet
Model Complexity	$\mathcal{O}(N+H)$				
Memory Complexity	N/A	$\mathcal{O}(E+H)$			$\mathcal{O}(N)$
Total Complexity	$\mathcal{O}(N+H)$	$\mathcal{O}(N+E+H)$			$\mathcal{O}(N+H)$

Model and Total complexity We analyze the model and the total memory complexity, which arises from the model and additional memory units.

First, the standard TCN forecaster incur a $\mathcal{O}(N+H)$ memory complexity arising from N parameters of the convolutional layers, and an order of H parameters from the linear regressor.

Second, we consider the replayed-based strategies, which also incur the same $\mathcal{O}(N+H)$ model complexity as the OnlineTCN. For the total memory, they use an episodic memory to store the previous samples, which costs $\mathcal{O}(E+H)$ for both methods. Additionally, TFCL stores the importance of previous parameters while MIR makes a copy of the model for its virtual update, both of which cost $\mathcal{O}(N+H)$. Therefore, the total memory complexity of the replay strategies (ER, DER++, MIR, and TFCL) is $\mathcal{O}(N+E+H)$.

Third, in FSNet, both the per-layer adapters and the associative memory cost similar number of parameters as the convolutional layers because they are matrices with number of channels as one dimension. Therefore, asymptotically, FSNet also incurs a model and total complexity of $\mathcal{O}(N+H)$ where the constant term is small.

Table 6.5 summarizes the asymptotic memory complexity discussed so far. Table 6.6 shows the number of parameters used of different strategies on the

TABLE 6.6: Summary of the model complexity on the ETTh2 data set with forecasting window $H = 24$. We report the number of floating points incurred by the backbone and different types of memory. GI = Gradient Importance (TFCL), G-EMA = Gradient Exponential Moving Average (FSNet), AM = Associative Memory (FSNet), EM = Episodic Memory (ER).

Method	Model		Memory				Total
	Backbone	Adapter	GI	G-EMA	AM	EM	
FSNet	1,041,288	733,334	N/A	614,400	1,130,496	N/A	3,519,518
ER	1,041,288	N/A	N/A	N/A	N/A	2,822,400	3,863,688
OnlineTCN	3,667,208	N/A	N/A	N/A	N/A	N/A	3,667,208
TFCL	1,041,288	N/A	2,082,576	N/A	N/A	806,400	3,930,264

ETTh2 dataset with the forecast window of $H = 24$. We consider the total parameters (model and memory) of FSNet as the total budget and adjust other baselines to meet the budget. As we analyzed, for FSNet, its components, including the adapter, associative memory, and gradient EMA, require an order of parameter as the convolutional layers in the backbone network. For the OnlineTCN strategy, we increases the number of convolutional filters so that it has roughly the same total parameters as FSNet. For ER and TFCL, we change the number of samples stored in the episodic memory.

Time Complexity We report the throughput (samples/second) of different methods in Table 6.7 using a single V100 GPU. We can see that ER and DER++ have high throughput (low running time) compared to others thanks to their simplicity. As FSNet introduces additional mechanisms to allow the network to take less samples to adapt to the distribution shifts, its throughput is lower than ER and DER++. Nevertheless, FSNet is more efficient than and MIR comparable to TFCL, which are two common continual learning strategies.

TABLE 6.7: Throughput (sample/second) of different methods in our experiments with forecast window of $H = 1$.

Running Time	ETTh2	ETTm1	WTH	ECL	Traffic	S-A
ER	46	46	43	42	39	46
DER++	45	45	43	42	38	46
TFCL	29	28	27	27	26	27
MIR	22	22	21	21	30	23
FSNet	28	28	28	27	27	29

6.4 Conclusion

We have investigated the limitations of training deep neural networks for online time series forecasting in non-stationary environments, where they lack the capability to adapt to new or recurring patterns quickly. We then propose Fast and Slow learning Networks (FSNet) by extending the CLS theory for continual learning to online time series forecasting. FSNet augments a neural network backbone with two key components: (i) an adapter for adapting to the recent changes; and (ii) an associative memory to handle recurrent patterns. Moreover, the adapter sparsely interacts with its memory to store, update, and retrieve important recurring patterns to facilitate learning of such events in the future. Extensive experiments demonstrate the FSNet's capability to deal with various types of concept drifts to achieve promising results in both real-world and synthetic time series data.

We now discuss several aspects for further studies. First, properly normalizing data in a stream can greatly facilitate training, especially under the presence of concept drift (Section 6.3.2). In addition, while FSNet presents a general framework to forecast time series online, adopting it to a particular application requires incorporating specific domain knowledge to ensure satisfactory performances. In summary, we firmly believe that FSNet is an encouraging first step towards a general solutions for an important, yet challenging problem of online time series forecasting.

6.5 Additional Details

6.5.1 Synthetic Data

We use the following first-order auto-regressive process model $AR_{\varphi}(1)$ defined as

$$X_t = \varphi X_{t-1} + \epsilon_t, \tag{6.8}$$

where ϵ_t are random noises and X_{t-1} are randomly generated. The S-Abrupt data is described by the following equation:

$$X_t = \begin{cases} AR_{0.1} & \text{if } 1 < t \leq 1000 \\ AR_{0.4} & \text{if } 1000 < t \leq 1999 \\ AR_{0.6} & \text{if } 2000 < t \leq 2999 \\ AR_{0.1} & \text{if } 3000 < t \leq 3999 \\ AR_{0.4} & \text{if } 4000 < t \leq 4999 \\ AR_{0.6} & \text{if } 5000 < t \leq 5999. \end{cases} \tag{6.9}$$

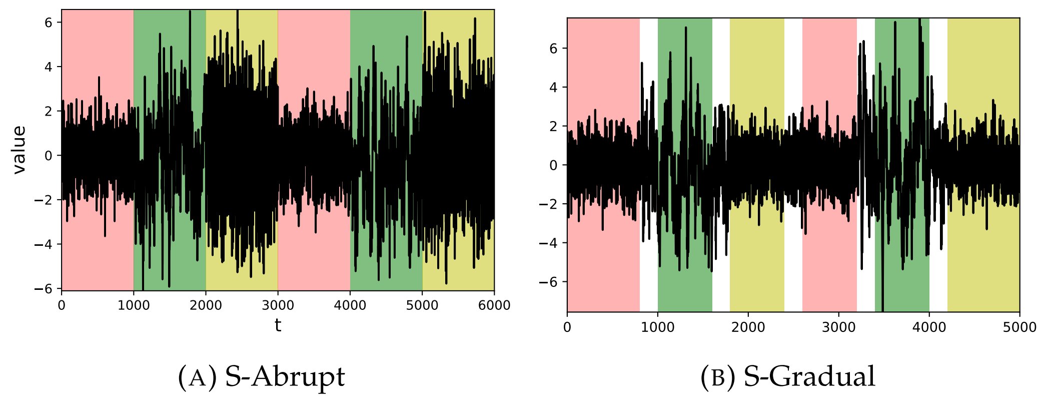


FIGURE 6.4: Visualization of the raw S-Abrupt and S-Gradual datasets before normalization. Colored regions indicate the data generating distribution where we use the same color for the same distribution. In S-Guadual, white color region indicates the gradual transition from one distribution to another.

The S-Gradual data is described as

$$X_t = \begin{cases} AR_{0.1} & \text{if } 1 < t \leq 800 \\ 0.5 \times (AR_{0.1} + AR_{0.4}) & \text{if } 800 < t \leq 1000 \\ AR_{0.4} & \text{if } 1000 < t \leq 1600 \\ 0.5 \times (AR_{0.4} + AR_{0.6}) & \text{if } 1600 < t \leq 1800 \\ AR_{0.6} & \text{if } 1800 < t < 2400 \\ 0.5 \times (AR_{0.6} + AR_{0.1}) & \text{if } 2400 < t \leq 2600 \\ AR_{0.1} & \text{if } 2600 < t \leq 3200 \\ 0.5 \times (AR_{0.1} + AR_{0.4}) & \text{if } 3200 < t \leq 3400 \\ AR_{0.4} & \text{if } 3400 < t \leq 4000 \\ 0.5 \times (AR_{0.4} + AR_{0.6} & \text{if } 4000 < t \leq 4200 \\ AR_{0.6} & \text{if } 4200 < t \leq 5000 \end{cases} \tag{6.10}$$

Figure 6.4 plots the unnormalized synthetic datasets.

6.5.2 Loss function

All methods in our experiments optimize the ℓ_2 loss function defined as follows. Let $\boldsymbol{x}$ and $\boldsymbol{y} \in \mathbb{R}^H$ be the look-back and ground-truth forecast windows, and $\hat{\boldsymbol{y}}$ be the model's prediction of the true forecast windows. The ℓ_2 loss is defined as:

$$\ell(\hat{\boldsymbol{y}}_t, \boldsymbol{y}_t) = \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}_t), \boldsymbol{y}_t) := \frac{1}{H} \sum_{j=1}^{H} ||\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i||^2 \tag{6.11}$$

Let $\mathcal{M}$ be the episodic memory storing previous samples, $\mathcal{B}_t$ be a mini-batch of samples sampled from $\mathcal{M}$. ER minimizes the following loss function:

$$\mathcal{L}_t^{\mathrm{ER}} = \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}_t), \boldsymbol{y}_t) + \lambda_{\mathrm{ER}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{B}_t} \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}), \boldsymbol{y}), \tag{6.12}$$

where $\ell(\cdot, \cdot)$ denotes the MSE loss and λ_{ER} is the trade-off parameter of current and past examples. DER++ further improves ER by adding a distillation loss [50]. For this purpose, DER++ also stores the model's forecast into the memory and minimizes the following loss:

$$\mathcal{L}_t^{\mathrm{DER++}} = \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}_t), \boldsymbol{y}_t) + \lambda_{\mathrm{ER}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{B}_t} \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}), \boldsymbol{y}) + \lambda_{\mathrm{DER++}} \sum_{(\boldsymbol{x},\hat{\boldsymbol{y}}) \in \mathcal{B}_t} \ell(f_{\boldsymbol{\omega}}(\boldsymbol{x}), \hat{\boldsymbol{y}}). \tag{6.13}$$

6.5.3 Implementing FSNet

Algorithm 4 provides the psedo-code for our FSNet.

Algorithm 4: Fast and Slow learning Networks (FSNet)

```
Require: Two EMA coefficients $\gamma' < \gamma$, memory interaction threshold $\tau$
Init: backbone $\boldsymbol{\theta}$, adapter $\boldsymbol{\phi}$, associative memory $\mathcal{M}$, regressor $\boldsymbol{R}$
1  for $t \leftarrow 1$ to $T$ do
2      Receive the $t-$ look-back window $\boldsymbol{x}_t$
3      $\boldsymbol{h}_0 = \boldsymbol{x}_t$
4      for $j \leftarrow 1$ to $L$ do                          // Forward computation over $L$ layers
5          $[\boldsymbol{\alpha}_l, \boldsymbol{\beta}_l] = \boldsymbol{u}_l, \text{ where } \boldsymbol{u}_l = \boldsymbol{\Omega}(\hat{\boldsymbol{g}}_l; \boldsymbol{\phi}_l)$          // Initial adaptation
           parameter
6          if trigger == True then
7              $\tilde{\boldsymbol{u}}_l \leftarrow \text{Read}(\hat{\boldsymbol{u}}_l, \mathcal{M}_l)$
8              $\mathcal{M}_l \leftarrow \text{Write}(\mathcal{M}_l, \hat{\boldsymbol{u}}_l)$
9              $\boldsymbol{u}_l \leftarrow \tau \boldsymbol{u}_l + (1-\tau)\tilde{\boldsymbol{u}}_l$
10         $\tilde{\boldsymbol{\theta}}_l = \text{tile}(\boldsymbol{\alpha}_l) \odot \boldsymbol{\theta}_l$                          // Weight adaptation
11         $\tilde{\boldsymbol{h}}_l = \text{tile}(\boldsymbol{\beta}_l) \odot \boldsymbol{h}_l, \text{ where } \boldsymbol{h}_l = \tilde{\boldsymbol{\theta}}_l \circledast \tilde{h}_{l-1}.$    // Feature adaptation
12     Forecast $\hat{\boldsymbol{y}}_t = \boldsymbol{R} h_T$
13     Receive the ground-truth $\boldsymbol{y}$
14     Calculate the forecast loss and backpropagate
15     Update the regressor $\boldsymbol{R}$ via SGD
16     for $j \leftarrow 1$ to $L$ do                 // Backward to update the model and EMA
17         Update the EMA of $\hat{\boldsymbol{g}}_l, \hat{\boldsymbol{g}}_l{}', \boldsymbol{u}_l$
18         Update $\phi_l, \boldsymbol{\theta}_l$ via SGD
19         if $\cos(\hat{\boldsymbol{g}}_l, \hat{\boldsymbol{g}}_l) < -\tau$ then
20             trigger $\leftarrow$ True
```

Chapter 7

Discussion and Future Work

In this Chapter, we first summarize the main contributions of the works presented in this dissertation in Section 7.1. We then outline several promising research directions towards realizing continual learning in Section 7.2. This dissertation ends with a concluding remark in Section 7.3.

7.1 Summary of Contributions

In this dissertation, we proposed novel algorithms for Continual Learning with Deep Neural Networks. Specifically, we identified the limitations of existing studies and proposed fast-and-slow continual learning, a novel and holistic framework to address those issues. Our methods are inspired by the Complementary Learning Systems (CLS) theory, which models the way humans perform continual learning. Our fast-and-slow learning framework suggests learning should consist of two levels: learning at the sample level and learning representation across tasks. We went beyond the traditional image benchmarks to test our fast-and-slow continual learning framework's capabilities in the time series forecasting domain. Lastly, we shed lights on how Batch Normalization

(BN), a classical and important component for training deep networks, could cause catastrophic forgetting. In summary, we have made the following contributions in the field of Continual Learning and time series forecasting:

- **Continual Normalization.** We conduct a meta-analysis of continual learning by studying BN to understand how it affects continual learning. We demonstrate that while BN can greatly facilitate the forward knowledge transfer, it simultaneously increases the error of learned tasks during test time, which we referred to as the cross-task normalization effect. From there, we propose a novel normalization layer named Continual Normalization (CN) that is specifically designed for online continual learning with images. CN can enjoy BN's benefits, alleviate the cross-task normalization effect, and incur minimal complexity overhead.

- **Contextual Transformation Networks** We aim at bridging the gap between two major continual learning approaches: (i) static architecture methods that are efficient but have limited performances; and (ii) dynamic architecture methods that achieve strong performances but are highly inefficient. The proposed CTN algorithm enhances a static architecture backbone with a novel controller component supporting the ability to model task-specific features while keeping the model's size almost fixed. Our findings show that CTN performs significantly better than previous strategies, even comparable with a strong baselines that has *17-times* more parameters.

- **Fast and Slow Continual Learning** This Chapter realizes the CLS theory into DualNets, a general continual learning framework. Particularly, DualNet consists of two distinct learning component: (i) the fast network plays the role of the hippocampus that focuses on fast learning of pattern-separated representation; and (ii) the slow network, which corresponds to the neocortex and learns a general, task-agnostic representation. Extensive experiments show that DualNets is general, applicable to various challenging continual learning scenarios, and robust to different hyper-parameter configurations.

- **Learning Fast and Slow for Online Time Series Forecasting** We proposed FSNet as a radical approach to online time series forecasting. FS-Net addresses the time series forecasting's challenges of fast adaptation to concept drifts and learning of recurring patterns by formulating them as a continual learning problem. FSNet further extends the fast-and-slow learning idea to derive an effective solution for these challenges. Our experimental findings demonstrated FSNet's ability to learn in nonstationary environments with concept drifts and recurring patterns.

7.2 Future Research Directions

This Section outlines the potential future directions for continual learning and beyond.

7.2.1 Continual Learning with Foundation Models

Foundation models [203] refer to a class of neural networks pre-trained on a massive amount of unlabeled data, usually via self-supervised learning. Since their first introduction in 2018 [97], foundation models have demonstrated remarkable successes in solving a wide range of problems, ranging from language [97, 196] to vision [174] domains. As a result, they present a paradigm shift in solving many AI problems. Due to the strong prior knowledge, foundation models can perform well on many different problems simply by finetuning a small number of parameters. Therefore, we firmly believe that foundation models is an important step toward general intelligent machines with continual learning capabilities.

Towards this goal, the learning agent first starts as a foundation model instead of learning from scratch, as we considered so far in this dissertation. With such pre-trained knowledge, foundation models have strong potential to learn new skills, even learning with a few training samples. However, they are also more prone to catastrophic forgetting: any changes to the model's parameters can greatly alter many learned patterns. As such, we need to revise the continual learning constraints for such foundation models. Particularly, facilitating forward transfer is almost guaranteed; however, one might be interested in a more compact and modularized way to organize knowledge in such models so that learning new skills could be further accelerated. Moreover, catastrophic forgetting is more eminent, which requires a better strategy to prevent the forgetting of old knowledge or even allow a positive backward transfer.

7.2.2 Broader Impacts of Continual Learning

The majority of existing continual learning research has been conducted on benchmarks derived from batch learning datasets. Although this strategy provides a controlled environment for fast research development, it lacks a connection to real-world applications. Recently, there have been attempts to extend continual learning strategies beyond such an environment to address other research challenges. Chapter 6 presents such an approach where we leveraged continual learning to address online time series forecasting. Beside our work, there are concurrent studies exploring continual learning in various applications domains such as: users modeling on Twitter [181], landmark recognition [205], recommendation systems [200], etc. Such studies explore how continual learning appears in natural data and propose novel solutions based on existing continual learning ideas. As a result, we believe that advances in continual learning research can result in significant improvements in those applications.

Beyond various applications, continual learning is also attractive to different research disciplines. One particular research of interest is the intersection of continual learning, multi-task/supervised learning [16], and the credit assignment problem [3]. Several studies [70, 71, 49] have explored multi-task learning models trained by gradient-based optimization (e.g. SGD) from the continual learning perspective. They discovered that in such settings, each task requires setting the model's parameter to a certain value, creating a tug-of-war dynamic [178]. In particular, training requires samples from all tasks to be present at each step to reach an equilibrium that can solve all tasks. Otherwise, the missing task does not participate in the tug-of-war game, and the model's parameters will be allocated to solve the remaining tasks. A closer look at supervised learning with a single task also reveals the same pattern [119]: the model can quickly learn easy samples while it takes much longer to learn the difficult ones. As a result, optimizations require going through the whole dataset several times until the model reaches an equilibrium of performances on both easy and difficult samples. This raises an interesting perspective of training classical paradigms more efficiently via continual learning. In this view, the continual learning models can quickly learn, remember easy samples and proceed to learn the more difficult ones, which would substantially reduce the training complexities.

7.2.3 Classical Continual Learning

We have discussed the potential of continual learning with foundation models and their broader impacts on other research disciplines. Those ambitious and long-term research goals require careful plans for meaningful progress. To this end, we review the benefits and limitations of existing studies and outline the immediate next steps for continual learning.

Summary and Benefits Most existing continual learning studies have developed strategies to enable deep neural networks to solve a sequence of tasks. These studies often employ a standard architecture (such as a MLP or a reduced version of ResNet [80]), and benchmarks constructed from standard datasets such as MNIST [19] or CIFAR [28]. As a result, such practices allow for a fast methodology development cycle, which quickly improves our understanding of continual learning.

Limitations The most critical limitation of classical continual learning studies is that such artificial benchmarks may only partially model how continual learning happens in the real world. For example, the above benchmarks do not consider the distribution shifts over time, training with limited or partially labeled samples, or negative transfer from noises, which are common practical consideration [132, 199]. As a result, general and practical continual learning remains a challenging research problem.

Outlook We firmly believe that developing practical and challenging benchmarks is an important and prominent first step for continual learning research.

The CTRL benchmark [199] used in Chapter 5 tried to incorporate other elements, such as (artificial) distribution shifts and negative transfer is a promising first step. Recently, there have been tremendous efforts in developing general, practical and challenging continual learning benchmarks. Notably, the Firehose [181] and Standard/Long Stream [212] are developed for continual learning in the natural language processing domain. Similarly, the Continual Localization [205] benchmark is a large-scale and challenging benchmark for continual learning in the vision domain. These are a few examples of recent benchmarks to facilitate continual learning in various domains, and we firmly believe they will facilitate meaningful continual learning progress in the future.

7.3 Concluding Remarks

Deep learning has been a frontier of AI research over the last decade. They presented a new tool for researchers and practitioners to achieve successes that seemed nearly impossible before. However, such promising opportunities are also accompanied by unique challenges. One of which is empowering deep neural networks with the continual learning capability to truly interact in a real-world environment. To this end, this dissertation studies the problem of continual learning with deep neural networks. We have contributed to the classical continual learning strategies and a novel fast-and-slow learning framework, which brings a new perspective to continual learning. We firmly believe our contributions present a promising and important step toward the next generation of intelligent machines with general and continual learning capabilities.

Bibliography

[1] DG Hebbs. “The organization of behavior”. In: *Wiely and Sons, New York, NY, USA* (1949).

[2] James Hannan. “Approximation to Bayes risk in repeated play”. In: *Contributions to the Theory of Games* 3 (1957), pp. 97–139.

[3] Marvin Minsky. “Steps toward artificial intelligence”. In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30.

[4] Stephen Grossberg. “How does a brain build a cognitive code?” In: *Studies of mind and brain* (1982), pp. 1–52.

[5] Jeffrey S Vitter. “Random sampling with a reservoir”. In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57.

[6] Gail A Carpenter and Stephen Grossberg. “ART 2: Self-organization of stable category recognition codes for analog input patterns”. In: *Applied optics* 26.23 (1987), pp. 4919–4930.

[7] Jürgen Schmidhuber. “Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook”. PhD thesis. Technische Universität München, 1987.

[8] 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.

[9] Roger Ratcliff. "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions." In: *Psychological review* 97.2 (1990), p. 285.

[10] Robert M French. "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks". In: *Proceedings of the 13th annual cognitive science society conference*. Vol. 1. 1991, pp. 173–178.

[11] Robert M French. "Semi-distributed representations and catastrophic forgetting in connectionist networks". In: *Connection Science* 4.3-4 (1992), pp. 365–377.

[12] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8.3-4 (1992), pp. 293–321.

[13] Ken McRae and Phil A Hetherington. "Catastrophic interference is eliminated in pretrained networks". In: *Proceedings of the 15h Annual Conference of the Cognitive Science Society*. 1993, pp. 723–728.

[14] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." In: *Psychological review* 102.3 (1995), p. 419.

[15] Anthony Robins. "Catastrophic forgetting, rehearsal and pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146.

[16] Rich Caruana. "Multitask learning". In: *Machine learning* 28.1 (1997), pp. 41–75.

[17] Mark B Ring. "CHILD: A first step towards continual learning". In: *Machine Learning* 28.1 (1997), pp. 77–104.

[18] Léon Bottou et al. “Online learning and stochastic approximations”. In: *On-line learning in neural networks* 17.9 (1998), p. 142.

[19] 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.

[20] David JC MacKay. “Choice of basis for Laplace approximation”. In: *Machine learning* 33.1 (1998), pp. 77–86.

[21] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[22] Jonathan Baxter. “A model of inductive bias learning”. In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198.

[23] Daniel L Silver and Robert E Mercer. “The task rehearsal method of life-long learning: Overcoming impoverished data”. In: *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer. 2002, pp. 90–101.

[24] Alexander J Smola, Vishy Vishwanathan, and Eleazar Eskin. “Laplace Propagation.” In: *NIPS*. Citeseer. 2003, pp. 441–448.

[25] Martin Zinkevich. “Online convex programming and generalized infinitesimal gradient ascent”. In: *Proceedings of the 20th international conference on machine learning (icml-03)*. 2003, pp. 928–936.

[26] Theodoros Evgeniou and Massimiliano Pontil. “Regularized multi–task learning”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 109–117.

[27] Benoît Colson, Patrice Marcotte, and Gilles Savard. "An overview of bilevel optimization". In: *Annals of operations research* 153.1 (2007), pp. 235–256.

[28] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[29] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. "Learning to detect unseen object classes by between-class attribute transfer". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 951–958.

[30] David Meunier, Renaud Lambiotte, Alex Fornito, Karen Ersche, and Edward T Bullmore. "Hierarchical modularity in human brain functional networks". In: *Frontiers in neuroinformatics* 3 (2009), p. 37.

[31] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. "Why does unsupervised pre-training help deep learning?" In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 201–208.

[32] Yusuf Aytar and Andrew Zisserman. "Tabula rasa: Model transfer for object category detection". In: *2011 international conference on computer vision*. IEEE. 2011, pp. 2252–2259.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning". In: (2011).

[34] Rainer Dahlhaus. "Locally stationary processes". In: *Handbook of statistics*. Vol. 30. Elsevier, 2012, pp. 351–413.

[35] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[36] Dharshan Kumaran and James L McClelland. "Generalization through the recurrent interaction of episodic memories: a model of the hippocampal system." In: *Psychological review* 119.3 (2012), p. 573.

[37] Michael Vogt. "Nonparametric regression for locally stationary time series". In: *The Annals of Statistics* 40.5 (2012), pp. 2601–2633.

[38] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. "Online learning for time series prediction". In: *Conference on learning theory*. PMLR. 2013, pp. 172–184.

[39] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[40] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1319–1327.

[41] 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).

[42] Rupesh Kumar Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino J Gomez, and Jürgen Schmidhuber. "Compete to Compute." In: *NIPS*. Citeseer. 2013, pp. 2310–2318.

[43] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. "Describing textures in the wild". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 3606–3613.

[44] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. "A survey on concept drift adaptation". In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37.

[45] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[46] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (2014).

[47] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[48] Anastasia Pentina and Christoph Lampert. "A PAC-Bayesian bound for lifelong learning". In: *International Conference on Machine Learning*. 2014, pp. 991–999.

[49] Shai Shalev-Shwartz. "Selfieboost: A boosting algorithm for deep learning". In: *arXiv preprint arXiv:1411.3436* (2014).

[50] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop*. 2015. URL: http://arxiv.org/abs/1503.02531.

[51] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[52] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR (Poster)*. 2015.

[53] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[54] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[55] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. “Efficient object localization using convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 648–656.

[56] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[57] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. “Learning feed-forward one-shot learners”. In: *Advances in neural information processing systems*. 2016, pp. 523–531.

[58] Sourav Das and Guy P Nason. “Measuring the degree of non-stationarity of a time series”. In: *Stat* 5.1 (2016), pp. 295–305.

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

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

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

[62] Dharshan Kumaran, Demis Hassabis, and James L McClelland. "What learning systems do intelligent agents need? Complementary learning systems theory updated". In: *Trends in cognitive sciences* 20.7 (2016), pp. 512–534.

[63] Chenghao Liu, Steven CH Hoi, Peilin Zhao, and Jianling Sun. "Online arima algorithms for time series prediction". In: *Thirtieth AAAI conference on artificial intelligence*. 2016.

[64] Anastasia Pentina. "Theoretical foundations of multi-task lifelong learning". PhD thesis. 2016.

[65] Jack W Rae, Jonathan J Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P Lillicrap. "Scaling memory-augmented neural networks with sparse reads and writes". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 3628–3636.

[66] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[67] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

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

[69] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[70] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. "A closer look at memorization in deep networks". In: *International conference on machine learning*. PMLR. 2017, pp. 233–242.

[71] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. "Active bias: Training more accurate neural networks by emphasizing high variance samples". In: *Advances in Neural Information Processing Systems* 30 (2017).

[72] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. "Pathnet: Evolution channels gradient descent in super neural networks". In: *arXiv preprint arXiv:1701.08734* (2017).

[73] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proceedings of the 34th*

International Conference on Machine Learning-Volume 70. JMLR. org. 2017, pp. 1126–1135.

[74] Sergey Ioffe. "Batch renormalization: towards reducing minibatch dependence in batch-normalized models". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 1942–1950.

[75] Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. "Learning to remember rare events". In: *arXiv preprint arXiv:1703.03129* (2017).

[76] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* (2017).

[77] Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[78] Vincenzo Lomonaco and Davide Maltoni. "CORe50: a New Dataset and Benchmark for Continuous Object Recognition". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Proceedings of Machine Learning Research. PMLR, 2017, pp. 17–26.

[79] David Lopez-Paz, Robert Nishihara, Soumith Chintala, Bernhard Scholkopf, and Léon Bottou. "Discovering causal signals in images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6979–6987.

[80] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6467–6476.

[81] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).

[82] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. “icarl: Incremental classifier and representation learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[83] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. “Learning multiple visual domains with residual adapters”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 506–516.

[84] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. “Continual learning with deep generative replay”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 2990–2999.

[85] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6924–6932.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[87] 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).

[88] Yang You, Igor Gitman, and Boris Ginsburg. “Large batch training of convolutional networks”. In: *arXiv preprint arXiv:1708.03888* (2017).

[89] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3987–3995.

[90] 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.

[91] 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.

[92] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches". In: *arXiv preprint arXiv:1803.01164* (2018).

[93] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

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

[95] Johan Bjorck, Carla Gomes, Bart Selman, and Kilian Q Weinberger. "Understanding batch normalization". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 7705–7716.

[96] 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.

[97] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[98] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. "Feature-wise transformations". In: *Distill* (2018). https://distill.pub/2018/feature-wise-transformations. DOI: 10.23915/distill.00011.

[99] Sebastian Farquhar and Yarin Gal. "Towards Robust Evaluations of Continual Learning". In: *arXiv preprint arXiv:1805.09733* (2018).

[100] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. "Bilevel Programming for Hyperparameter Optimization and Meta-Learning". In: *International Conference on Machine Learning*. 2018, pp. 1568–1577.

[101] Spyros Gidaris and Nikos Komodakis. "Dynamic few-shot visual learning without forgetting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4367–4375.

[102] San Gultekin and John Paisley. "Online forecasting matrix factorization". In: *IEEE Transactions on Signal Processing* 67.5 (2018), pp. 1223–1236.

[103] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. "Lifelong learning via progressive distillation and retrospection". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 437–452.

[104] Ferenc Huszár. "Note on the quadratic penalties in elastic weight consolidation". In: *Proceedings of the National Academy of Sciences* 115.11 (2018), E2496–E2497.

[105] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[106] Khurram Javed and Faisal Shafait. "Revisiting distillation and incremental classifier learning". In: *Asian conference on computer vision*. Springer. 2018, pp. 3–17.

[107] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. "Reply to Huszár: The elastic weight consolidation penalty is empirically valid". In: *Proceedings of the National Academy of Sciences* 115.11 (2018), E2498–E2498.

[108] Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. "Differentiable Learning-to-Normalize via Switchable Normalization". In: *International Conference on Learning Representations*. 2018.

[109] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. "Variational continual learning". In: *International Conference on Learning Representations (ICLR)* (2018).

[110] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[111] German I Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. "Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization". In: *Frontiers in neurorobotics* 12 (2018), p. 78.

[112] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. "Film: Visual reasoning with a general conditioning layer". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[113] B Pfülb and A Gepperth. "A comprehensive, application-oriented study of catastrophic forgetting in DNNs". In: *International Conference on Learning Representations*. 2018.

[114] Hippolyt Ritter, Aleksandar Botev, and David Barber. "Online structured laplace approximations for overcoming catastrophic forgetting". In: *Advances in Neural Information Processing Systems*. 2018, pp. 3738–3748.

[115] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. "Online Deep Learning: Learning Deep Neural Networks on the Fly". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. 2018.

[116] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. "How does batch normalization help optimization?" In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 2488–2498.

[117] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. "Overcoming Catastrophic Forgetting with Hard Attention to the Task". In: *Proceedings of the 35th International Conference on Machine Learning-Volume 80*. JMLR. org. 2018, pp. 4548–4557.

[118] Suraj Srinivas and François Fleuret. “Knowledge transfer with jacobian matching”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4723–4731.

[119] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. “An empirical study of example forgetting during deep neural network learning”. In: *arXiv preprint arXiv:1812.05159* (2018).

[120] Gido M van de Ven and Andreas S Tolias. “Generative replay with feedback connections as a general strategy for continual learning”. In: *arXiv preprint arXiv:1809.10635* (2018).

[121] Yuxin Wu and Kaiming He. “Group normalization”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[122] Ju Xu and Zhanxing Zhu. “Reinforced continual learning”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 899–908.

[123] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. “Lifelong Learning with Dynamically Expandable Networks”. In: *International Conference on Learning Representations (ICLR)* (2018).

[124] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. “Gradient based sample selection for online continual learning”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11816–11825.

[125] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. “Online continual learning with maximal interfered retrieval”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11849–11860.

[126] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. "Task-free continual learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11254–11263.

[127] Sergul Aydore, Tianhao Zhu, and Dean P Foster. "Dynamic Local Regret for Non-convex Online Forecasting". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 7982–7991.

[128] Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. "Domain generalization by solving jigsaw puzzles". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2229–2238.

[129] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. "Efficient Lifelong Learning with A-GEM". In: *International Conference on Learning Representations (ICLR)* (2019).

[130] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. "On Tiny Episodic Memories in Continual Learning". In: *arXiv preprint arXiv:1902.10486* (2019).

[131] Brian Cheung, Alexander Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno Olshausen. "Superposition of many models into one". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 10868–10877.

[132] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. "Continual learning in practice". In: *arXiv preprint arXiv:1903.05202* (2019).

[133] Sebastian Farquhar and Yarin Gal. "A unifying bayesian view of continual learning". In: *arXiv preprint arXiv:1902.06494* (2019).

[134] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. "Online meta-learning". In: *Proceedings of the 36th International Conference on Machine Learning-Volume 97*. JMLR. org. 2019, pp. 1920–1930.

[135] Elad Hazan. "Introduction to online convex optimization". In: *arXiv preprint arXiv:1909.05207* (2019).

[136] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. "Task agnostic continual learning via meta learning". In: *arXiv preprint arXiv:1906.05201* (2019).

[137] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. "Compacting, Picking and Growing for Unforgetting Continual Learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13669–13679.

[138] Khurram Javed and Martha White. "Meta-learning representations for continual learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 1818–1828.

[139] Ghassen Jerfel, Erin Grant, Tom Griffiths, and Katherine A Heller. "Reconciling meta-learning and continual learning with online mixtures of tasks". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 9122–9133.

[140] Richard Kurle, Botond Cseke, Alexej Klushyn, Patrick Van Der Smagt, and Stephan Günnemann. "Continual learning with bayesian neural networks for non-stationary data". In: *International Conference on Learning Representations*. 2019.

[141] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. "Enhancing the locality and breaking the

memory bottleneck of transformer on time series forecasting". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 5243–5253.

[142] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. "Understanding the disharmony between dropout and batch normalization by variance shift". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2682–2690.

[143] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. "Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting". In: *International Conference on Machine Learning*. 2019, pp. 3925–3934.

[144] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search". In: *International Conference on Learning Representations (ICLR)* (2019).

[145] Shikun Liu, Edward Johns, and Andrew J Davison. "End-to-end multi-task learning with attention". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1871–1880.

[146] Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. "Episodic Memory in Lifelong Language Learning". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 13143–13152.

[147] Cuong V Nguyen, Alessandro Achille, Michael Lam, Tal Hassner, Vijay Mahadevan, and Stefano Soatto. "Meta-analysis of Continual Learning". In: *NeurIPS Workshops*. Vol. 1. 2. 2019, p. 4.

[148] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. "Learning to remember: A synaptic plasticity driven framework for continual learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11321–11329.

[149] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* (2019).

[150] Dongmin Park, Seokil Hong, Bohyung Han, and Kyoung Mu Lee. "Continual learning by asymmetric loss approximation with single-side overestimation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3335–3344.

[151] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "PyTorch: An imperative style, high-performance deep learning library". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8024–8035.

[152] Mary Phuong and Christoph H Lampert. "Distillation-based training for multi-exit architectures". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1355–1364.

[153] Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, and Ling Shao. "Random path selection for incremental learning". In: *Advances in Neural Information Processing Systems* (2019).

[154] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. "Continual Unsupervised Representation Learning". In: *Advances in Neural Information Processing Systems* (2019).

[155] James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. "Fast and flexible multi-task classification using conditional neural adaptive processes". In: *Advances in Neural Information Processing Systems*. 2019, pp. 7959–7970.

[156] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. "Learning to learn without forgetting by maximizing transfer and minimizing interference". In: *International Conference on Learning Representations (ICLR)* (2019).

[157] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems*. 2019, pp. 348–358.

[158] Monika Schak and Alexander Gepperth. "A study on catastrophic forgetting in deep LSTM networks". In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 714–728.

[159] Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. "LAMOL: LAnguage MOdeling for Lifelong Language Learning". In: *International Conference on Learning Representations*. 2019.

[160] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. "Understanding and Improving Layer Normalization". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 4381–4391.

[161] Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. "Scalable and Order-robust Continual Learning with Additive Parameter Decomposition". In: *International Conference on Learning Representations*. 2019.

[162] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. "Lookahead optimizer: k steps forward, 1 step back". In: *Advances in Neural Information Processing Systems* (2019).

[163] Guangxiang Zhu, Zichuan Lin, Guangwen Yang, and Chongjie Zhang. "Episodic Reinforcement Learning with Associative Memory". In: *International Conference on Learning Representations*. 2019.

[164] Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. "Fast context adaptation via meta-learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7693–7702.

[165] Muhammad Awais, Md Tauhid Bin Iqbal, and Sung-Ho Bae. "Revisiting internal covariate shift for batch normalization". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[166] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. "Agent57: Outperforming the atari human benchmark". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 507–517.

[167] Mehdi Abbana Bennani, Thang Doan, and Masashi Sugiyama. "Generalisation guarantees for continual learning with orthogonal gradient descent". In: *arXiv preprint arXiv:2006.11942* (2020).

[168] Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-jussà. "Continual Lifelong Learning in Natural Language Processing: A Survey". In: *Proceedings of the 28th International Conference on Computational Linguistics*. 2020, pp. 6523–6541.

[169] John Bronskill, Jonathan Gordon, James Requeima, Sebastian Nowozin, and Richard Turner. "Tasknorm: Rethinking batch normalization for meta-learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1153–1164.

[170] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. “Dark Experience for General Continual Learning: a Strong, Simple Baseline”. In: *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. 2020.

[171] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexande Lacoste, David Vazquez, et al. “Online Fast Adaptation and Knowledge Accumulation: a New Approach to Continual Learning”. In: *arXiv preprint arXiv:2003.05856* (2020).

[172] Wei-Lun Chao, Han-Jia Ye, De-Chuan Zhan, Mark Campbell, and Kilian Q Weinberger. “Revisiting meta-learning as supervised learning”. In: *arXiv preprint arXiv:2002.00573* (2020).

[173] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[174] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[175] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. “Podnet: Pooled outputs distillation for small-tasks incremental learning”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer. 2020, pp. 86–102.

[176] Camila Gonzalez, Georgios Sakas, and Anirban Mukhopadhyay. "What is Wrong with Continual Learning in Medical Image Segmentation?" In: *arXiv preprint arXiv:2010.11008* (2020).

[177] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. "Bootstrap your own latent: A new approach to self-supervised learning". In: *Advances in Neural Information Processing Systems* (2020).

[178] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. "Embracing change: Continual learning in deep neural networks". In: *Trends in cognitive sciences* 24.12 (2020), pp. 1028–1040.

[179] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. "Remind your neural network to prevent catastrophic forgetting". In: *European Conference on Computer Vision*. Springer. 2020, pp. 466–483.

[180] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[181] Hexiang Hu, Ozan Sener, Fei Sha, and Vladlen Koltun. "Drinking from a firehose: Continual learning with web-scale natural language". In: *arXiv preprint arXiv:2007.09335* (2020).

[182] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. "Normalization Techniques in Training DNNs: Methodology, Analysis and Application". In: *arXiv preprint arXiv:2009.12836* (2020).

[183] Chris Dongjoo Kim, Jinseo Jeong, and Gunhee Kim. "Imbalanced continual learning with partitioning reservoir sampling". In: *European Conference on Computer Vision*. Springer. 2020, pp. 411–428.

[184] Jeremias Knoblauch, Hisham Husain, and Tom Diethe. "Optimal continual learning has perfect memory and is np-hard". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5327–5337.

[185] Hung Le, Truyen Tran, and Svetha Venkatesh. "Self-attentive associative memory". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5682–5691.

[186] Steven Cheng-Xian Li and Benjamin Marlin. "Learning from irregularly-sampled time series: A missing data perspective". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5937–5946.

[187] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. "Mnemonics training: Multi-class incremental learning without forgetting". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12245–12254.

[188] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. "Rehearsal-Free Continual Learning over Small Non-IID Batches." In: *CVPR Workshops*. 2020, pp. 989–998.

[189] Jorge A Mendez and Eric Eaton. "Lifelong learning of compositional structures". In: *arXiv preprint arXiv:2007.07732* (2020).

[190] Fei Mi, Xiaoyu Lin, and Boi Faltings. "Ader: Adaptively distilled exemplar replay towards continual learning for session-based recommendation". In: *Fourteenth ACM Conference on Recommender Systems*. 2020, pp. 408–413.

[191] Seyed Iman Mirzadeh, Mehrdad Farajtabar, and Hassan Ghasemzadeh. "Dropout as an implicit gating mechanism for continual learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 232–233.

[192] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. "Linear Mode Connectivity in Multitask and Continual Learning". In: *International Conference on Learning Representations*. 2020.

[193] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. "Understanding the Role of Training Regimes in Continual Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*. 2020.

[194] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. "Continual learning with hypernetworks". In: *International Conference on Learning Representations (ICLR)* (2020).

[195] Fotios Petropoulos, Daniele Apiletti, Vassilios Assimakopoulos, Mohamed Zied Babai, Devon K Barrow, Souhaib Ben Taieb, Christoph Bergmeir, Ricardo J Bessa, Jakub Bijak, John E Boylan, et al. "Forecasting: theory and practice". In: *arXiv preprint arXiv:2012.03854* (2020).

[196] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J. Mach. Learn. Res.* 21.140 (2020), pp. 1–67.

[197] Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. "Anatomy of Catastrophic Forgetting: Hidden Representations and Task Semantics". In: *International Conference on Learning Representations*. 2020.

[198] Hoang Thanh-Tung and Truyen Tran. "Catastrophic forgetting and mode collapse in GANs". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–10.

[199] Tom Veniat, Ludovic Denoyer, and MarcAurelio Ranzato. "Efficient Continual Learning with Modular Networks and Task-Driven Priors". In: *International Conference on Learning Representations*. 2020.

[200] Fajie Yuan, Guoxiao Zhang, Alexandros Karatzoglou, Xiangnan He, Joemon Jose, Beibei Kong, and Yudong Li. "One person, one model, one world: Learning continual user representation without forgetting". In: *arXiv preprint arXiv:2009.13724* (2020).

[201] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. "Class-incremental learning via deep model consolidation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1131–1140.

[202] Aadyot Bhatnagar, Paul Kassianik, Chenghao Liu, Tian Lan, Wenzhuo Yang, Rowan Cassius, Doyen Sahoo, Devansh Arpit, Sri Subramanian, Gerald Woo, et al. "Merlion: A machine learning library for time series". In: *arXiv preprint arXiv:2109.09265* (2021).

[203] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258* (2021).

[204] Andrew Brock, Soham De, and Samuel L Smith. "Characterizing signal propagation to close the performance gap in unnormalized ResNets". In: *arXiv preprint arXiv:2101.08692* (2021).

[205] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. "Online continual learning with natural distribution shifts: An empirical study with visual data". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8281–8290.

[206] Xinlei Chen and Kaiming He. "Exploring simple siamese representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15750–15758.

[207] Yue Cui, Jiandong Xie, and Kai Zheng. "Historical inertia: A neglected but powerful baseline for long sequence time-series forecasting". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 2965–2969.

[208] Thang Doan, Mehdi Abbana Bennani, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Alquier. "A Theoretical Analysis of Catastrophic Forgetting through the NTK Overlap Matrix". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 1072–1080.

[209] Benjamin Ehret, Christian Henning, Maria Cervera, Alexander Meulemans, Johannes Von Oswald, and Benjamin F Grewe. "Continual learning in recurrent neural networks". In: *International Conference on Learning Representations*. 2021.

[210] Amirreza Farnoosh, Bahar Azari, and Sarah Ostadabbas. "Deep Switching Auto-Regressive Factorization: Application to Time Series Forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 7394–7403.

[211] Vibhor Gupta, Jyoti Narwariya, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. "Continual Learning for Multivariate Time Series Tasks

with Variable Input Dimensions”. In: *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2021, pp. 161–170.

[212] Aman Hussain, Nithin Holla, Pushkar Mishra, Helen Yannakoudakis, and Ekaterina Shutova. “Towards a robust experimental framework and benchmark for lifelong language learning”. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*. 2021.

[213] Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. “Continual learning via local module composition”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 30298–30312.

[214] Quang Pham, Chenghao Liu, Doyen Sahoo, and Steven CH Hoi. “Contextual Transformation Networks for Online Continual Learning”. In: *International Conference on Learning Representations (ICLR)* (2021).

[215] Quang Pham, Chenghao Liu, and Steven Hoi. “Dualnet: Continual learning, fast and slow”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 16131–16144.

[216] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir, and Ram Rajagopal. “NeuralProphet: Explainable Forecasting at Scale”. In: *arXiv preprint arXiv:2111.15397* (2021).

[217] Liyuan Wang, Mingtian Zhang, Zhongfan Jia, Qian Li, Chenglong Bao, Kaisheng Ma, Jun Zhu, and Yi Zhong. “Afec: Active forgetting of negative transfer in continual learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 22379–22391.

[218] Jiehui Xu, Jianmin Wang, Mingsheng Long, et al. “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[219] Haiyan Yin, Ping Li, et al. "Mitigating Forgetting in Online Continual Learning with Neuron Calibration". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 10260–10272.

[220] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. "TS2Vec: Towards Universal Representation of Time Series". In: *arXiv preprint arXiv:2106.10466* (2021).

[221] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. "Barlow twins: Self-supervised learning via redundancy reduction". In: *arXiv preprint arXiv:2103.03230* (2021).

[222] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. "Informer: Beyond efficient transformer for long sequence time-series forecasting". In: *Proceedings of AAAI*. 2021.

[223] Quang Pham, Chenghao Liu, and HOI Steven. "Continual Normalization: Rethinking Batch Normalization for Online Continual Learning". In: *International Conference on Learning Representations*. 2022.

[224] Quang Pham, Chenghao Liu, Doyen Sahoo, and Steven CH Hoi. "Learning Fast and Slow for Online Time Series Forecasting". In: *arXiv preprint arXiv:2202.11672* (2022).

[225] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. "CoST: Contrastive Learning of Disentangled Seasonal-Trend Representations for Time Series Forecasting". In: *arXiv preprint arXiv:2202.01575* (2022).