Rehearsal-based Continual Learning with Deep Neural Networks for Image Classification

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

Felix Wiewel

aus Münster

Hauptberichter:	Prof. Dr.-Ing. Bin Yang
Mitberichter:	Prof. Dr. Wolfram Burgard

Tag der mündlichen Prüfung:	03.04.2024

Institut für Signalverarbeitung und Systemtheorie
der Universität Stuttgart

2024

Danksagung

Diese Arbeit entstand während meiner Zeit als Doktorand am Institut für Signalverarbeitung und Systemtheorie (ISS) an der Universität Stuttgart unter der Leitung von Prof. Dr. Bin Yang. In dieser Zeit hatte ich nicht nur die Möglichkeit die wissenschaftliche Arbeit und Lehre am Institut sondern auch viele nette und hilfsbereite Menschen aus dem In- und Ausland kennenzulernen.

Besonders bedanken möchte ich mich bei meinem Doktorvater Prof. Dr. Bin Yang für die Betreuung dieser Doktorarbeit. Hervorheben möchte ich hierbei vor allem die Freiheiten in der Durchführung, die hilfreichen Gespräche während der gesamten Zeit, die Unterstützung durch sein umfangreiches Fachwissen sowie das Betreben seine Doktoranden bestmöglich zu unterstützen. Ein weiterer herzlicher Dank gilt Prof. Dr. Wolfram Burgard von der Technischen Universität Erlangen für die Übernahme des Mitberichts.

Den internen und externen Doktoranden sowie allen Mitarbeitern des ISS danke ich für die interessanten und hilfreichen Gespräche sowie ihre generelle Unterstützung. Besonderer Dank gilt hierbei Alexander Bartler, Mario Döbler und Robert Marsden für die etlichen Gespräche und Kooperationen in der wissenschaftlichen Arbeit sowie der Lehre am ISS. Weiterhin möchte ich mich bei all meinen Studenten für die Unterstützung durch ihre Arbeiten danken.

Abschließend möchte ich einen ganz besonderen Dank an meine Familie richten, meine Frau Xiao Guo, meinen Bruder Florian Wiewel, meine Eltern Petra Miech und Reinhard Wiewel, meine Großeltern und alle weiteren Familienmitgleider die mich stets auf meinem Weg unterstüzt haben. Ich widme diese Arbeit ihnen.

Stuttgart, 28. April 2024 Felix Wiewel

Contents

Abstract

Despite recent successes in many fields of machine intelligence, there are still some important aspects of learning where a Deep Neural Network (DNN) lacks in capability when compared with humans. One of these is the ability to learn on a sequence of tasks while retaining previously learned knowledge and transferring it to new unseen ones. Humans excel at this and learn in a continual way throughout most of their life, which not only enables them to learn fast but also efficient. Currently used DNNs on the other hand suffer from the phenomenon of catastrophic forgetting, a rapid decrease in performance on a previously learned task almost instantly after training of a new task begins. Overcoming this limitation and enabling continual learning with DNNs is therefore of great interest and would not only make their training potentially faster and more efficient. It would also enable applications where they can be trained incrementally whenever new training data is available without requiring to store all previously learned data.

The focus of this thesis are rehearsal-based methods for continual learning with DNNs on resource constrained systems where it is not possible to store large amounts of training data. Despite their simplicity these methods show strong baseline performance and can be applied to all types of continual learning problems. Image classification on publicly available benchmark datasets is used as an exemplary problem type throughout the remainder of this work due to its popularity in the literature and a therefore simplified comparison with related work.

A summary of the applicability to different continual learning scenarios forms the basis of a comprehensive overview for continual learning methods. It is complemented by a more detailed presentation of canonical examples from every type of approach to continual learning, metrics, commonly used datasets and neural architectures.

A novel method for analyzing catastrophic forgetting by attributing a change in loss to individual parameters of a DNN is introduced. In contrast to the common practice of measuring catastrophic forgetting using scalar metrics, this method allows for identifying which parts of a DNN suffer to what extend from forgetting. Using this approach, three different continual learning scenarios are studied on several datasets. The experimental results align with related literature and establish a deeper understanding of the continual learning scenarios and the most vulnerable parts in the architecture of a DNN.

Deciding which data to add and which to potentially replace is an important step in the context of rehearsal-based methods. It is especially challenging in the online continual learning setup where data arrives in a stream or small batches without knowledge about task boundaries. A novel method for sample selection based on an information theoretic perspective on sampling from a rehearsal memory is presented. Experimental results show that it outperforms direct competitors and is competitive with the latest related work.

The use of synthetically generated data for rehearsal-based continual learning shows great potential as it allows for generating data that maximizes the rehearsal performance. But synthesizing examples individually entails the risk of generating and storing redundant information since it prevents sharing components between them. In order to avoid this and improve storage efficiency, a novel method that learns data in a form that allows for sharing common components between them is introduced. The increased memory efficiency and the accompanying performance increase through being able to store more data in the same sized memory is demonstrated on commonly used benchmark datasets and compared with related methods.

Despite its simplicity rehearsal shows a strong baseline performance in continual learning but is still sensitive to the exact implementation of the rehearsing process itself. Depending on how rehearsal and new data are combined in a mini batch, vastly different continual learning performance and computational efficiencies are achieved. Based on a formal study of the underlying effects, a novel method that incorporates the prior knowledge about the rehearsal mechanism into the prediction of a DNN is introduced. This reduces the sensitivity with respect to the underlying implementation of the rehearsal mechanism and ultimately allows for using a more computationally efficient implementation while maintaining a performance similar to the most demanding rehearsal scheme.

Finally, the main contributions of this thesis to analyzing catastrophic forgetting, rehearsal memory management for online continual learning, efficient generation and storage of synthetic data and the incorporation of prior knowledge about the rehearsal mechanism itself into the prediction of a DNN are summarized and an outlook is discussed.

Kurzfassung

Trotz der jüngsten Erfolge in vielen Bereichen der maschinellen Intelligenz gibt es immer noch einige wichtige Aspekte des Lernens, bei denen ein DNN im Vergleich zum Menschen Defizite aufweist. Einer davon ist die Fähigkeit, eine Reihe von Aufgaben zu erlernen und dabei bereits gelerntes Wissen beizubehalten sowie auf neue, ungesehene Aufgaben zu übertragen. Der Mensch ist darin besonders gut und lernt fast sein ganzes Leben lang kontinuierlich, was ihm nicht nur ein schnelles, sondern auch ein effizientes Lernen ermöglicht. Die derzeit verwendeten DNNs leiden hingegen unter dem Phänomen des katastrophalen Vergessens, d. h. einem rapiden Leistungsabfall bei zuvor gelernten Aufgaben fast unmittelbar nach Beginn des Trainings einer neuen Aufgabe. Die Überwindung dieser Beeinträchtigung und die Ermöglichung kontinuierlichen Lernens mit DNNs ist daher von großem Interesse und würde nicht nur ihr Training potenziell schneller und effizienter machen. Es würde auch Anwendungen ermöglichen, bei denen sie inkrementell trainiert werden können, sobald neue Trainingsdaten verfügbar sind, ohne dass alle zuvor eingelernten Daten aufbewahrt werden müssen.

Der Schwerpunkt dieser Arbeit liegt auf Rehearsal-basierten Methoden des kontinuierlichen Lernens mit DNNs für Systeme auf denen es nicht möglich ist große Mengen an Trainingsdaten zu speichern. Trotz ihrer Einfachheit bieten diese Methoden eine solide Basisleistung und können auf alle Arten von kontinuierlichen Lernproblemen angewendet werden. Die Bildklassifikation auf öffentlich zugänglichen Referenzdatensätzen wird im weiteren Verlauf dieser Arbeit als exemplarischer Problemtyp verwendet, da sie in der Literatur weit verbreitet ist und daher einen vereinfachten Vergleich mit verwandten Arbeiten ermöglicht.

Eine Zusammenfassung der Anwendbarkeit auf verschiedene Szenarien bildet die Grundlage für einen umfassenden Überblick über die Methoden des kontinuierlichen Lernens. Sie wird ergänzt durch eine detailliertere Darstellung von kanonischen Beispielen für alle Arten von Ansätzen zum kontinuierlichen Lernen, Metriken, häufig verwendeten Datensätzen und neuronalen Architekturen.

Eine neuartige Methode zur Analyse des katastrophalen Vergessens durch die Zuordnung einer Verluständerung zu einzelnen Parametern eines DNN wird vorgestellt. Im Gegensatz zu der üblichen Praxis, katastrophales Vergessen mit Hilfe von skalaren Metriken zu messen, ermöglicht diese Methode die Bestimmung, welche Teile eines DNN in welchem Ausmaß unter dem Vergessen leiden. Mit diesem Ansatz werden drei verschiedene Szenarien des kontinuierlichen Lernens auf mehreren Datensätzen untersucht. Die experimentellen Ergebnisse stimmen mit der einschlägigen Literatur überein und tragen zu einem tieferen Verständnis der kontinuierlichen Lernszenarien und der am stärksten gefährdeten Teile der Architektur eines DNNs bei.

Die Entscheidung, welche Daten dem Rehearsal-Speicher hinzuzufügen und welche möglicherweise zu ersetzen sind, ist ein wichtiger Schritt bei Rehearsal-basierten Methoden. Dies ist eine besondere Herausforderung beim kontinuierlichen Online-Lernen, bei dem die Daten in einem Strom von Einzeldaten oder kleinen Gruppen eingehen, ohne dass die Grenzen der Aufgaben bekannt sind. Es wird ein neuartiges Verfahren für die Auswahl von Daten aus einem Rehearsal-Speicher vorgestellt, das auf einer informationstheoretischen Perspektive der Datenentnahme basiert. Experimentelle Ergebnisse zeigen, dass diese Methode die direkten Konkurrenten übertrifft und mit den neuesten verwandten Methoden konkurrenzfähig ist.

Die Verwendung synthetisch erzeugter Daten für das kontinuierliche Lernen auf der Grundlage von Rehearsal weist ein großes Potenzial auf, da sie es ermöglicht, Daten zu erzeugen, welche die Wiederholungsleistung maximieren. Die Synthese einzelner Beispiele birgt jedoch das Risiko, redundante Informationen zu erzeugen und zu speichern, da die gemeinsame Nutzung von Komponenten zwischen den Beispielen verhindert wird. Um dies zu vermeiden und die Speichereffizienz zu verbessern, wird eine neuartige Methode eingeführt, die Daten in einer Form lernt, die es erlaubt, gemeinsame Komponenten zwischen ihnen zu teilen. Die gesteigerte Speichereffizienz und die damit einhergehende Leistungssteigerung durch die Möglichkeit, mehr Daten in der gleichen Speichergröße zu speichern, wird anhand häufig verwendeter Datensätze demonstriert und mit ähnlichen Methoden verglichen.

Trotz seiner Einfachheit zeigt Rehearsal eine starke Grundleistung beim kontinuierlichen Lernen, ist aber dennoch empfindlich gegenüber der genauen Implementierung des Rehearsal-Prozesses selbst. Je nachdem, wie Rehearsal-Daten und neue Daten in einem Mini-Batch kombiniert werden, werden sehr unterschiedliche kontinuierliche Lernleistungen und Recheneffizienzen erzielt. Auf der Grundlage einer formalen Untersuchung der zugrundeliegenden Effekte wird eine neuartige Methode eingeführt, die das Vorabwissen über den Rehearsal-Mechanismus in die Vorhersage eines DNNs einbezieht. Dadurch wird die Empfindlichkeit gegenüber der zugrundeliegenden Implementierung des Rehearsal-Mechanismus reduziert. Weiterhin ermöglicht dies letztendlich die Verwendung einer wesentlich recheneffizienteren Implementierung, wobei eine ähnliche Leistung wie bei den anspruchsvollsten Rehearsal-Schemata erzielt wird.

Abschließend werden die wichtigsten Beiträge dieser Arbeit zur Analyse des katastrophalen Vergessens, des Rehearsal-Speicher-Managements für kontinuierliches Online-Lernen, der effizienten Erzeugung und Speicherung synthetischer Daten und der Einbeziehung von Vorwissen über den Rehearsal-Mechanismus selbst in die Vorhersagen eines DNNs zusammengefasst und ein Ausblick diskutiert.

List of Symbols

Notation

$\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$	Set of natural, integer, real and complex numbers
$x \in \mathbb{R}$	Scalar
$\mathbf{x} \in \mathbb{R}^M$	M-dimensional column vector
$\mathbf{X} \in \mathbb{R}^{M \times N}$	$M \times N$-dimensional Matrix
$f : \mathcal{X} \rightarrow \mathcal{Y}$	Function mapping from vector space $\mathcal{X}$ to $\mathcal{Y}$
$\mathcal{T} = \{x_i, y_i\}_{i=1}^N$	Set of N pairs x, y
$[\mathbf{x}]_i$	i-th element of vector $\mathbf{x}$
$[\mathbf{X}]_{i,j}$	i, j-th element of matrix $\mathbf{X}$
$p(x)$	Probability density function of the random variable X
$p(x\|y)$	Conditional probability density function of the random variable X conditioned on y
$p(x, y)$	Joint probability density function of the random variables X and Y
$\mathrm{Dir}(\boldsymbol{\mu}\|\boldsymbol{\alpha})$	Dirichlet distribution over $\boldsymbol{\mu}$ with concentration parameters $\boldsymbol{\alpha}$

Mathematical operations

$\|\mathcal{T}\|$	Cardinality of set $\mathcal{T}$
$\min_x f(x)$	Minimum of function f w.r.t x
$\arg\min_x f(x)$	Argument for minimum of function f w.r.t x
$\mathrm{dom}(f)$	Domain of function f

$\mathrm{supp}(f)$	Support of function f
e^x	Exponential function of x
$\ln(x)$	Natural logarithm of x
$\mathbb{E}_{p(\mathbf{x}\|\boldsymbol{\theta})}[\mathbf{x}]$	Expectation of random variable $\mathbf{x}$ following the conditional distribution $p(\mathbf{x}\|\boldsymbol{\theta})$
$\frac{\partial}{\partial x}$	Partial derivative w.r.t x
$\frac{\partial^2}{\partial^2 x}$	Second partial derivative w.r.t x
$\mathrm{diag}(\mathbf{X})$	Diagonal of square matrix $\mathbf{X}$
$\nabla_{\mathbf{x}} f(\mathbf{x})$	Gradient of scalar valued function f w.r.t. to vector $\mathbf{x}$
$\langle \mathbf{x}, \mathbf{y} \rangle$, $\mathbf{xy}$	Inner product between vectors $\mathbf{x}$ and $\mathbf{y}$
$\\|\mathbf{x}\\|_p = \sqrt[p]{\sum_i \|[\mathbf{x}]_i\|^p}$	p-norm of vector $\mathbf{x}$
$\mathrm{KL}(p\\|q)$	KL divergence between p and q
$\odot$	Hadamard product

Symbols

$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
$\mathcal{T}$	Task
$\mathcal{D}$	Domain
$\mathcal{A}$	Assignment
$\mathcal{R}$	Regularization function
$\mathcal{L}$	Loss function
$\mathcal{Q}$	Set of posterior distributions
$\mathcal{N}$	Normal distribution
$\mathcal{C}$	Trajectory through parameter space
$\mathcal{M}$	Rehearsal memory
$\mathcal{S}$	Set of synthetic data
$\mathcal{S}^\star$	Set of optimal synthetic data
$\mathcal{B}$	Mini batch
$\boldsymbol{\theta}$	Vector of trainable parameters

$\boldsymbol{\theta}^\star$	Vector of optimal parameters
$\boldsymbol{\theta}_k^\star$	Vector of optimal parameters for k-th task
$\boldsymbol{\delta}_{\boldsymbol{\theta}}$	Infinitesimal change in parameters $\boldsymbol{\theta}$
$\boldsymbol{\phi}$	Embedding
$\boldsymbol{\mu}$	Vector of class probabilities
$\boldsymbol{\alpha}$	Precision parameters of Dirichlet distribution
ω	Contribution to change in loss
Ω	Regularization weights of SI
β	Weighting parameter
σ	Activation function
$\hat{\alpha}$	Weighting parameter
r	Ratio of new data in mini batch
s	Ratio of rehearsal data in mini batch
f, g, h	DNNs
Z	Normalization constant
H	Shannon Entropy
K	Kernel function
d	Distance
C	Number of classes
$\mathbf{F}$	Fisher Information matrix
$\mathbf{e}$	Anchor point
$\mathbf{W}$	Weight matrix or tensor
$\mathbf{R}$	Matrix containing accuracy on sequence of tasks
$\mathbf{c}$	Loss change contribution vector
$\mathbf{g}$	Gradient vector

Acronyms

A Accuracy

AGEM Averaged Gradient Episodic Memory

AGI Artificial General Intelligence

AI Artificial Intelligence

AIA Average Incremental Accuracy

ASER Adversarial Shapely value Experience Replay

AUCROC Area Under Curve of the Receiver Operating Characteristic Curve

BiC Bias Correction

BOS Buffer Oversampling

BP Backpropagation

BSIL Balanced Softmax Incremental Learning

BWT Backward Transfer

CCMCL Condensed Composite Memory Continual Learning

CE Computational Efficiency

CL-LROS Continual Learning in Low-Rank Orthogonal Subspaces

CLS Complementary Learning Systems

COS Class Oversampling

CVPR Computer Vision and Pattern Recognition Conference

DCGM	Dataset Condensation with Gradient Matching
DFA	Direct Feedback Alignment
DGR	Deep Generative Replay
DMC	Deep Model Consolidation
DNN	Deep Neural Network
DPN	Dirichlet Prior Network
DPNCL	Dirichlet Prior Networks for Continual Learning
EbLL	Encoder-based Lifelong Learning
ELBO	Evidence Lower Bound
ESS	Entropy-based Sample Selection
EWC	Elastic Weight Consolidation
FIM	Fisher Information Matrix
FWT	Forward Transfer
GAN	Generative Adversarial Network
GEM	Gradient Episodic Memory
GSS	Gradient based Sample Selection
GVCL	Generalized Variational Continual Learning
HAL	Hindsight Anchor Learning
HAT	Hard Attention to the Task
i.i.d.	independent and identically distributed
ICaRL	Incremental Classifier and Representation Learning
ICL	Incremental Class Learning

ID In-Distribution

IDL Incremental Domain Learning

ILDA Incremental Learning through Deep Adaptation

IMM Incremental Moment Matching

IQP Interger Quadratic Program

ITL Incremental Task Learning

KDE Kernel Density Estimator

KL Kullback-Leibler

KNN k-Nearest Neighbours

LUCIR Learning a Unified Classifier Incrementally via Rebalancing

LwF Learning without Forgetting

MAC Multiply-Accumulate

MAS Memory Aware Synapses

MBSIL Meta Balanced Softmax Incremental Learning

MC Monte Carlo

MIR Maximally Interfered Retrieval

MLP Multi-Layer Preceptron

MS Model Size

MSE Mean Squared Error

NAS Neural Architecture Search

NCL Natural Continual Learning

NeurIPS Conference on Neural Information Processing Systems

NoOS	No Oversampling
NTK	Neural Tangent Kernel
OGD	Orthogonal Gradient Descent
OOD	Out-Of-Distribution
OSLA	Online Structured Laplace Approximation
PC	Progress and Compress
PN	Pack Net
PNN	Progressive Neural Network
PODNet	Pooled Output Distillation Network
R-EWC	Rotated Elastic Weight Consolidation
RCL	Reinforced Continual Learning
ReLU	Rectified Linear Unit
RWALK	Riemannian Walk
SGD	Stochastic Gradient Descent
SI	Synaptic Intelligence
SLAM	Simultaneous Localization and Mapping
SSS	Sample Storage Size
SVM	Support Vector Machine
TPCIL	Topology-Preserving Class-Incremental Learning
UB	Upper Bound
VAE	Variational Auto Encoder

VCL	Variational Continual Learning
VIO	Visual-Inertial Odometry
VQ	Vector Quantization

Chapter 1.

Introduction

Since the first breakthrough win of a DNN with AlexNet [32] in the ILSVRC-2012 competition, their capabilities have rapidly increased to a point where DNNs now achieve or even surpass human performance on several tasks like object recognition [38] and games like Go [65].

But despite this success, DNNs and Artificial Intelligence (AI) in general still fall short in some aspects of learning when compared to humans. For this thesis, the ability to learn an ever increasing number of tasks in a sequential way is of particular interest. Humans typically undergo a life-long learning process without problems in memory retention and tend to learn sequentially while transferring knowledge between previously learned and new tasks [20]. Although rehearsing previously learned tasks may be part of this process, the incorporation of new knowledge usually does not interfere with previously learned concepts. The ability to transfer knowledge from previous experience to new concepts further supports such a sequential way of learning. Although the details of human cognition currently are not fully understood, neuroscience research suggests that an intricate balance between stability and plasticity of neurons is maintained throughout the human brain [45, 86]. As a consequence, humans can build a vast knowledge over their life-time through accumulation of new, transfer of old to new and revision of previously acquired knowledge.

In contrast to this, DNNs suffer from the phenomenon of catastrophic forgetting, also known as catastrophic interference, where the incorporation of new knowledge interferes with previously learned concepts and causes a severe degradation of performance on all but the most recently learned ones. Catastrophic forgetting in DNNs is well known since the early 1990s [10, 11] but overcoming it and maintaining the right balance between plasticity and stability has proven itself to be a long standing challenge. Recently the study of this phenomenon, methods on how to overcome it and enabling DNNs to learn in a sequential way has seen renewed interest through an advent in the field of continual learning.

1.1. What is Continual Learning?

In the context of machine learning and DNNs, continual learning, also referred to as continuous, lifelong, or incremental learning, describes the study of learning in an incremental way on a sequence of tasks with limited access to previously encountered data when a new task is trained. In principle there are many different types of tasks that can be considered in continual learning as DNNs can be applied to solve a multitude of different problems like object recognition and detection, semantic segmentation, natural language processing and regression. While early work on continual learning was limited to object recognition, i.e. image classification, an increased interest in the field has lead to continual learning being applied to many other problem settings as well. Typically all tasks in a sequence originate from the same type of problem but in general continual learning can also be applied to a sequence of tasks, that originate from different problems. But even in the former case there can be distinct scenarios of continual learning depending on what exactly differs between tasks. A formal definition of continual learning in the context of this thesis, how it relates to other fields and a brief summary of its biological background are given in chapter 2.

1.2. Why is Continual Learning of Interest?

Given even this rather short introduction to continual learning it is obvious that overcoming catastrophic forgetting and enabling continual learning is of great interest not only from a purely scientific perspective but also for enabling new applications of DNNs. The process of training a DNN that is capable of continual learning, for example, does not require simultaneous access to the complete training dataset. Instead it can be trained on small parts of the training dataset individually while accumulating knowledge over time, essentially replacing spatial through temporal aggregation of data. This can be especially useful in applications where simultaneous access to the complete training dataset is prohibited, for example due to limited storage or legal restrictions preventing the accumulation of large amounts of private data. Achieving continual learning, in general, can be seen as moving away from the current paradigm of static DNNs with parameters that remain fixed after training on to DNNs that can dynamically adapt to changes

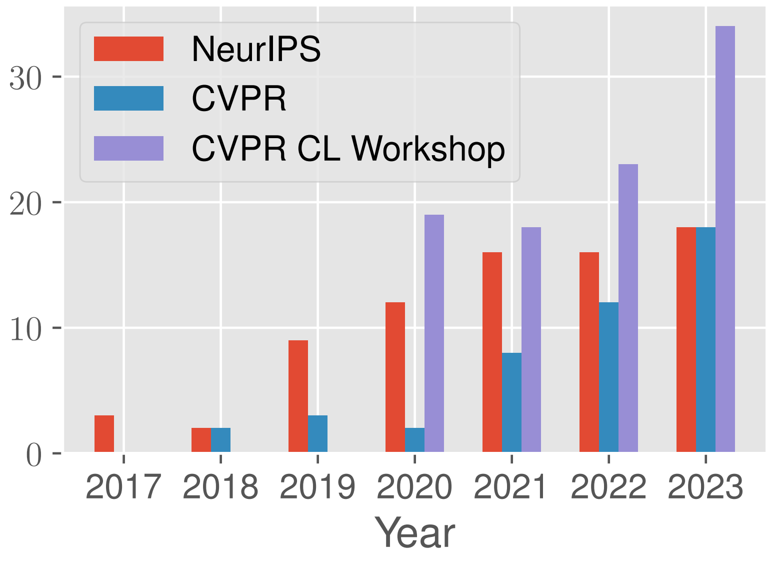


Figure 1.1.: Number of publications whose title includes the keywords "continual learning", "continuous learning" or "catastrophic forgetting". Workshop numbers are total submissions.

in the environment they are deployed in. This would bring them closer to a Artificial General Intelligence (AGI) “that is capable of solving almost all tasks that humans can solve” [90].

This increasing interest in continual learning for DNNs is also reflected in leading conferences of the field. Figure 1.1 shows the number of publications form the Conference on Neural Information Processing Systems (NeurIPS) and Computer Vision and Pattern Recognition Conference (CVPR) whose title includes at least one of the keywords "continual learning", "continuous learning" or "catastrophic forgetting". In addition to an increasing number of publications on conferences, dedicated workshops for continual learning have been introduced. As an example, the total number of submissions to the "Continual Learning in Computer Vision" Workshop of CVPR is also shown.

1.3. Contribution and Structure of this Thesis

This thesis combines the work of several years and publications with the goal of contributing to and advancing the field of continual learning with DNNs. Given the large number of settings and possible approaches to the problem, a focus is required in order to keep the overall scope of this thesis manageable. Based on a thorough review of the literature and experiments, rehearsal-based methods were identified as a promising direction and therefore a focus of the work that this thesis is based upon.

The structure of this thesis is split into the first chapters, which introduce a formal definition of continual learning and a literature overview, the main contributions of this work and a closing chapter.

Chapter 2 introduces a formal definition of continual learning as the common foundation for all following chapters. Three scenarios of continual learning that are common in the literature are introduced using examples. This is followed by a short explanation of the relation between continual learning and other related but different fields like transfer and federated learning. The chapter is then closed by a brief introduction into the biological aspects and neuroscience background of continual learning.

A literature overview that covers the related work most relevant to this thesis follows in chapter 3. The presented methods are categorized in regularization-based, structural, Bayesian or memory-based methods in order to give a more structured overview. Due to the large number of proposed algorithms and methods for avoiding catastrophic forgetting, only a subset of all works is presented. But to give the reader a wider overview and advice for further reading references to more methods and their applicability to the different scenarios are given in Table 3.1. Furthermore, metrics, datasets and architectures that are commonly used throughout the literature and this thesis are introduced in the closing sections of this chapter.

The main part of this thesis focuses on four aspects of rehearsal-based continual learning that are formulated as research questions:

- Chapter 4: *Which parts of a DNN contribute with what extend to a change in loss when it is trained on a continual learning sequence without any measures for preventing catastrophic forgetting?*
- Chapter 5: *How can the buffer of a rehearsal-based method for continual learning be managed in a simple but effective way to enable online continual learning?*
- Chapter 6: *How can synthetic data be learned for rehearsal-based continual learning while reducing the redundancy of data stored in memory?*
- Chapter 7: *What types of distribution shifts can be introduced by rehearsal and how can their negative effects be avoided?*

These four research questions and the corresponding methods introduced in the associated chapters contribute to a better understanding of catastrophic forgetting, improved buffer management during online continual learning, more efficient storage of data for rehearsal and an improved method for using the stored data during rehearsal.

The first main contribution of this thesis presented in chapter 4 is a novel method for determining the contribution from each weight of a DNN to an increase in loss during catastrophic forgetting. Its main purpose is to allow a more detailed analysis of the main challenge in continual learning, catastrophic forgetting. It is based on the approach proposed by Wiewel and Yang [94] but extends their work with a more advanced numerical integration method and experiments on more datasets and deeper architectures. The experimental results give more insight into which parts of a DNN are most vulnerable to catastrophic forgetting and underline the empirical evidence from related work, which shows that layers closer to the output of a DNN are affected more than those close to the input.

In chapter 5 the second main contribution in the form of a novel algorithm for selecting examples to store in rehearsal memory is introduced for the online continual learning setup, where data is available only as a stream of individual examples. The goal of this contribution is to provide a lightweight but still powerful method for selecting what data to store in the challenging online setting. While the algorithm and its information theoretic motivation are based on the work of Wiewel and Yang [126], the experimental evaluation is expanded to more datasets and compared with a wider selection of related methods.

Chapter 6 presents the third main contribution and is based on the work of Wiewel and Yang [125], which explores the use of synthetic data for improving memory-based continual learning through a more efficient way of storing data. It is a novel method of representing the learned synthetic examples in a rehearsal memory in a way that allows for sharing components that are common for a specific class and therefore increases memory efficiency.

The fourth and final main contribution of this thesis is outlined in chapter 7. It is based on the work of Wiewel, Bartler, and Yang [133], which tries to improve memory-based continual learning through including prior knowledge into the prediction of a DNN. It exploits information about the rehearsal process itself, like statistics of the data stored in memory and training data, in order to improve performance. The experimental results show that this method not only improves performance significantly when compared to baselines but is also competitive with related work.

Chapter 8 concludes this thesis with a summary and discussion of the main contributions to the field of continual learning. Finally an outlook and remaining open questions are presented in section 8.2.

Additional Work on Continual Learning There are also two additional publications [93, 111] that are part of the work done on continual learning but cover a related but distinctly different direction and hence are not included in this thesis as main chapters. Although these are also based on memory-based continual learning, they rely on a Variational Auto Encoder (VAE) as an architecture for generative rehearsal. This is in contrast to the main contributions presented above which store examples of training data in a memory and do not use any kind of generative model.

In an early work [93] Wiewel and Yang study the extension of an anomaly detection system based on a VAE with generative rehearsal in order to enable continual learning with it. While their experiments show success with regards to the continual learning aspect, the anomaly detection performance is not competitive with current approaches as it relies heavily on the reconstruction error of the VAE to detect anomalies.

As a continuation of their work in the field of anomaly detection and one-class classification, Wiewel, Brendle, and Yang propose to use multiple VAEs with a shared encoder as a model with generative rehearsal ability in [111] for continual learning. Their main idea is to treat the problem of continual learning as a growing set of one-class classification problem, where each VAE is specialized to detect if in input originates from its associated class or not while at the same time providing a generative model for it. While the experimental results of this approach show some success on rather simple datasets, it suffers from scalability issues since for each class that is to be learned a new decoder is needed.

List of Publications The following lists all publications that are presented in the main chapters of this thesis in the same order as the main chapters:

- Felix Wiewel and Bin Yang. "Localizing Catastrophic Forgetting in Neural Networks". In: *CoRR* abs/1906.02568 (2019)[1]

[1]Preprint

- Felix Wiewel and Bin Yang. "Entropy-based Sample Selection for Online Continual Learning". In: *2020 28th European Signal Processing Conference (EUSIPCO)*. 2021, pp. 1477–1481
- Felix Wiewel and Bin Yang. "Condensed Composite Memory Continual Learning". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8
- Felix Wiewel, Alexander Bartler, and Bin Yang. "Dirichlet Prior Networks for Continual Learning". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8

The two publications on continual learning that are not part of this thesis are:

- Felix Wiewel and Bin Yang. "Continual Learning for Anomaly Detection with Variational Autoencoder". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 3837–3841
- Felix Wiewel, Andreas Brendle, and Bin Yang. "Continual Learning Through One-Class Classification Using VAE". in: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 3307–3311

Chapter 2.

Continual Learning

2.1. Definition of Continual Learning

The informal definition of continual learning as solving a sequence of tasks in an incremental way given in the introduction is quiet general but imprecise. In order to precisely define continual learning and what we want to achieve by it, we have to first describe the problem setting we are operating in. For this, we start with the definition of a task.

Definition 1: A Task in the Context of Continual Learning

A task $\mathcal{T}$ in continual learning is represented by a finite set of N elements that form a dataset. In supervised continual learning these are pairs of inputs $x_i \in \mathcal{X}$ and outputs $y_i \in \mathcal{Y}$ which results in

$$\mathcal{T} = \{x_i, y_i\}_{i=1}^{N}, \tag{2.1}$$

where the function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps the input space $\mathcal{X}$ to the output space $\mathcal{Y}$ is generally unknown and to be learned by a DNN. In unsupervised continual learning only inputs and no outputs are given.

Tasks in continual learning can be of different sizes, from containing many thousand to only a single input-output pair, from different modalities, e.g. images, audio signals or text data, and have different underlying functions f, e.g. image classification, semantic segmentation, regression or text translation. A typical example for a task that appears frequently in the literature is given by a dataset of images $\mathbf{x}_i \in \{0, \ldots, 255\}^{H \times W \times C}$ with height H, width W and C channels with their associated class labels $y_i \in \mathbb{N}$.

Although this definition describes what a task is, the process of solving it, which is an integral part of continual learning, still remains undefined. Given that there are many different types of possible tasks, a definition for solving a task that is applicable to all of them is necessary. In the context of continual learning with DNNs the process of minimizing a suitable loss function by finding optimal parameters for it can be used to find such a common definition.

Definition 2: Solving a Task

The process of solving a task $\mathcal{T}$ using a DNN as given by Definition 1 is defined as finding the optimal parameters

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{T}, \boldsymbol{\theta}), \tag{2.2}$$

where $\mathcal{L}$ is a suitable loss function and $\boldsymbol{\theta}$ is a vector containing all parameters of the DNN.

Although this definition is sufficiently precise to define the process of continual learning with DNNs, it is flexible enough to be applicable for many different types of tasks. Depending on the type of a task $\mathcal{T}$, the choice of a suitable loss function $\mathcal{L}$ leads to different specific optimization problems to be solved. In the case of supervised image classification using empirical risk minimization [14], for example, a popular choice for a suitable loss function is the categorical cross entropy. In this case, solving a task by finding the optimal parameters according to (2.2) results in

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \frac{-1}{|\mathcal{T}|} \sum_{\mathbf{x},\mathbf{y}\in\mathcal{T}} \sum_{c=1}^{C} [\mathbf{y}]_c \ln([g_{\boldsymbol{\theta}}(\mathbf{x})]_c), \tag{2.3}$$

where $\mathbf{y}$ are one-hot encoded class labels and $g_{\boldsymbol{\theta}}(\mathbf{x})$ are probabilities over the C classes predicted by the DNN.

Given these definitions of a task and how it can be solved, the continual learning problem as it will be considered in the remainder of this thesis can be defined.

Definition 3: The Continual Learning Problem

Considering a DNN parameterized by $\boldsymbol{\theta}$ and given a sequence of M tasks $\mathcal{T}_{1\leq k\leq M}$, the continual learning problem requires finding a sequence of optimal parameters

$$\boldsymbol{\theta}_k^\star = \arg\min_{\boldsymbol{\theta}} \sum_{j=1}^{k} \mathcal{L}_j(\mathcal{T}_j, \boldsymbol{\theta}), \tag{2.4}$$

where at step k only $\mathcal{T}_k$ is accessible, $\mathcal{L}_j$ is a suitable loss function for $\mathcal{T}_j$ and $1 \leq k \leq M$.

This definition might seem rather strict when compared to human incremental learning as it prohibits any access to data of tasks $\mathcal{T}_{1\leq j\leq k-1}$ at step k while humans occasionally rehearse previously learned concepts when learning new ones. But, on the other hand, there is no objective reason why DNNs could not reach or even exceed human level performance in continual learning since they already outperform them on other problems as well. Relaxations to this problem which allow for limited access to the data of previous tasks, for example through a small rehearsal buffer, exist, but their usage is usually carefully limited. In addition to the restricted access to data of previous tasks and achieving minimum loss, there are also other desirable properties that any method for continual learning with DNNs should fulfill if it is to be implemented in practice, e.g. low computational complexity and limited memory requirements. An overview of metrics used throughout the literature for evaluating such methods is therefore given in section 3.5.

2.2. Continual Learning Scenarios

In addition to these general definitions of the continual learning problem and tasks, it is also important to study what changes between tasks can occur and how these can effect a DNN throughout the sequence of tasks. Although there are potentially many different scenarios for this, the following discussion is limited to three scenarios from supervised continual learning with examples of image classification as it is one of the most widely used combinations throughout the literature. These three characterizations of different scenarios were, to the best of our knowledge, first proposed by Hsu et al. in [68] and Van de Ven and Tolias in [91] and form a basis of the following discussion.

For the following discussion, a probabilistic view on tasks is adopted since a characterization in one of these three scenarios is based on how the statistical properties differ between tasks.

Definition 4: A Probabilistic View on Tasks

A task according to Definition 1 can be viewed as a set of N samples from the random variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ with joint probability density function (pdf) $p_{XY} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ while the function $f : \mathcal{X} \rightarrow \mathcal{Y}$ can be viewed as a transformation from X to Y and therefore induces the conditional distribution $p_{Y|X} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^+$.

Using this probabilistic perspective the marginal distribution over X p_X, can be obtained through marginalization of p_{XY} over Y. This allows for defining a domain, which describes the input space and the statistical properties of the input data, and an assignment, which describes the output space and the statistical properties of the output conditioned on the input.

Definition 5: Domain and Assignment

Given the input space $\mathcal{X}$ and the marginal pdf $p_X : \mathcal{X} \rightarrow \mathbb{R}^+$, a domain is defined as

$$\mathcal{D} = \{\mathcal{X}, p_X\}.$$

Given the output space $\mathcal{Y}$ and the conditional distribution $p_{Y|X} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^+$, an assignment is defined as

$$\mathcal{A} = \{\mathcal{Y}, p_{Y|X}\}.$$

Given these definitions, the task $\mathcal{T}$ is a finite set of samples from a domain $\mathcal{D}$ and an assignment $\mathcal{A}$.

These definitions follow the notation used in the literature on transfer learning [130] with exception of the term assignment, which is referred to as a task but was renamed in the context of this thesis in order to avoid confusion. Although these definitions are not strictly necessary in order to define the three different continual learning scenarios, they are none the less useful when comparing continual learning to other fields like transfer learning.

Incremental Class Learning The Incremental Class Learning (ICL) scenario was originally proposed in the context of continual learning for image classification with DNNs. It is characterized by a sequence of tasks where each new task contains data of at least one new previously unseen class and arises naturally when a classification system needs to be expanded with new classes. Given two tasks $\mathcal{T}_k$ with $\mathcal{D}_k, \mathcal{A}_k$ and $\mathcal{T}_{k+1}$ with $\mathcal{D}_{k+1}, \mathcal{A}_{k+1}$, Hsu et al. define this scenario in [68] such that $\mathcal{X}_k = \mathcal{X}_{k+1}$ while $p_{X,k} \neq p_{X,k+1}$ and $p_{Y,k} \neq p_{Y,k+1}$. While this definition uniquely defines what changes occur between tasks, it allows for two different ways of how to handle the new classes.

First consider the case where $\mathcal{Y}_k = \mathcal{Y}_{k+1}$ and $p_{Y,k} \neq p_{Y,k+1}$. In this case, the output space $\mathcal{Y}$ and the conditional distribution $p_{Y|X}$ remain unchanged between tasks while the change in p_Y is solely caused by a changing p_X. For the practical implementation with a DNN, this means that the number of neurons in its output layer remains constant throughout the complete sequence of tasks. So even though the first task might contain only a few classes, the output layer must be large enough to accommodate all classes that might be encountered in future tasks.

The second case arises when $\mathcal{Y}_k \subset \mathcal{Y}_{k+1}$ and $p_{Y,k} \neq p_{Y,i+k}$. This implies that a change in the marginal distribution p_Y is not only caused by a change in p_X but also by a changing $p_{Y|X}$. In contrast to the first case, where the same output layer can be used for all tasks, in this case the output layer needs to be expanded whenever a new task is learned in order to accommodate the newly added classes. But since $\mathcal{Y}_k \subset \mathcal{Y}_{k+1}$ this new expanded output layer is still capable of classifying previously learned classes as well.

Examples for this scenario are typically constructed using classification datasets that are split into two or more parts. Usually the split is done in such a way that one task contains all data of certain classes while other tasks do not contain any examples from those. An example for supervised image classification can be constructed as follows. Consider a first task $\mathcal{T}_1$ represented by a dataset containing RGB images $\mathbf{x}_{i,1} \in \{0, \dots 255\}^{1024\times1024\times3}$ that show cars of two different brands which are to be classified by their brand. The corresponding labels for these images are given by $y_{i,1} \in \{0, 1\}$. The second task $\mathcal{T}_2$ also contains RGB images $\mathbf{x}_{i,2} \in \{0, \dots 255\}^{1024\times1024\times3}$ but from two different car brands. The corresponding labels for this task are therefore $y_{i,2} \in \{2, 3\}$. It is obvious that $\mathcal{X}_1 = \mathcal{X}_2 = \{0, \dots 255\}^{1024\times1024\times3}$ and hence the input space remains unchanged. At the same time it is also clear that the marginal distributions over it change, i.e. $p_{X,1} \neq p_{X,2}$, since the car brands and their appearance change between the two tasks. Considering the output space there are two possible cases as mentioned above. First, one can define $\mathcal{Y}_1 = \mathcal{Y}_2 = \{0, 1, 2, 3\}$. Although task $\mathcal{T}_1$ contains only two classes, the output space can represent four classes for both tasks. In practice this requires a DNN with an output layer that has four neurons even for the first task. Or second, $\mathcal{Y}_1 = \{0, 1\} \subset \mathcal{Y}_2 = \{0, 1, 2, 3\}$ is considered. In this case, the output layer of a DNN needs to be expanded when switching from task $\mathcal{T}_1$ to $\mathcal{T}_2$ since the output space is growing. Regardless which of these two cases is considered both yield $p_{Y,1} \neq p_{Y,2}$ and therefore comply with the definition of ICL given above.

Incremental Domain Learning The second scenario, Incremental Domain Learning (IDL), is well known from the field of transfer learning. In this scenario, tasks differ only in their domain, specifically in the marginal distribution p_X over the same input space. So for any two tasks $\mathcal{T}_k$ with $\mathcal{D}_k, \mathcal{A}_k$ and $\mathcal{T}_{k+1}$ with $\mathcal{D}_{k+1}, \mathcal{A}_{k+1}$, $\mathcal{X}_k = \mathcal{X}_{k+1}$ and $p_{X,k} \neq p_{X,k+1}$ while $\mathcal{A}_k = \mathcal{A}_{k+1}$. These definitions imply that the function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ mapping the inputs from both domains $\mathcal{D}_k$ and $\mathcal{D}_{k+1}$ to their corresponding class labels remains fixed between tasks. Formally

$$\mathrm{dom}(f) = \mathrm{supp}(p_{X,k}) \cup \mathrm{supp}(p_{X,k+1}), \tag{2.5}$$

where dom() is the domain of a function, i.e. the set of all inputs for which the function is defined, and supp() is its support, i.e. the set of all inputs for which the function is non-zero. Or expressed in words: Task k contains inputs that only cover a part of the domain f, while task $k+1$ contains inputs from a different part. But a specific input is mapped to the same class label by the function f regardless of the domain it is observed in. It is important to clearly differentiate between the operator dom() and the domain $\mathcal{D}$ in Definition 5.

Since in IDL the input $\mathcal{X}$ and output space $\mathcal{Y}$ in addition to the function f remain unchanged between tasks, the input and output layers of a DNN used in such a scenario can be shared between tasks. In contrast to the later introduced ITL scenario, this also means that during inference there is no need to know from which task an input originated. Although each task represents a different part of $\mathrm{dom}(f)$ the DNN only learns a single unchanging mapping from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. Hence there is no need to use more than one output layer.

A typical example for IDL can be constructed for image classification. Suppose the first task $\mathcal{T}_1$ is given by a dataset containing RGB images $\mathbf{x}_{i,1} \in \{0, \ldots 255\}^{1024 \times 1024 \times 3}$ showing cars from two different brands at daytime captured with a resolution of one megapixel. The corresponding outputs $y_{i,1} \in \{0, 1\}$ encode the car brand with one brand represented by 0 and the other by 1. Now let the second task $\mathcal{T}_2$ be given by RGB images $\mathbf{x}_{i,2} \in \{0, \ldots 255\}^{1024 \times 1024 \times 3}$ showing exactly the same car instances[1] from the same brands but this time captured during nighttime with a resolution of one megapixel. Since this second dataset contains exactly the same objects

[1]Considering exactly the same car instances is not strictly necessary according to the definition above but it makes the example easier to grasp.

as the first just photographed at a different time, the corresponding outputs $y_{i,2} \in \{0, 1\}$ remain unchanged. Note that $f_1 = f_2$ and therefore the conditional distributions $p_{Y|X,1}$ and $p_{Y|X,2}$ also remain unchanged. This is possible since each domain only covers a part of the input space while being mapped to the same region in the output space. In this case the output spaces $\mathcal{Y}_1 = \mathcal{Y}_2 = \{0, 1\}$, the conditional distributions $p_{Y|X,1} = p_{Y|X,2}$ and the input spaces $\mathcal{X}_1 = \mathcal{X}_2 = \{0, \dots 255\}^{1024\times1024\times3}$ remain unchanged, since both datasets contain images of exactly the same cars corresponding to the same brands captured at the same resolution. But since the first task contains images captured at daytime and the second at nighttime, the marginal distributions over the input spaces differ, i.e. $p_{X,1} \neq p_{X,2}$.

Incremental Task Learning The last scenario, Incremental Task Learning (ITL), in general allows for the most changes between tasks as each task can feature a completely different type of problem to be solved. Because of this and in contrast to the two previously discussed scenarios, ITL generally also requires some architectural changes of a DNN. Given two tasks $\mathcal{T}_k$ with $\mathcal{D}_k, \mathcal{A}_k$ and $\mathcal{T}_{k+1}$ with $\mathcal{D}_{k+1}, \mathcal{A}_{k+1}$, Hsu et al. define it in [68] by $\mathcal{Y}_1 \neq \mathcal{Y}_2$. This definition also implies $p_{Y|X,1} \neq p_{Y|X,2}$ and $p_{Y,1} \neq p_{Y,2}$. But in addition to these differences in the assignments $\mathcal{A}_1$ and $\mathcal{A}_2$, changes between the domains $\mathcal{D}_1$ and $\mathcal{D}_2$ are generally also possible.

These changes in the output space generally require adding a separate output layer for every task in a practical implementation for supervised classification. Note that for ITL, in contrast to ICL, just expanding the output layer is not possible since $\mathcal{Y}_k \not\subset \mathcal{Y}_{k+1}$. In addition to this a separate input layer for a task might be needed as well if $\mathcal{X}_1 \neq \mathcal{X}_2$. During inference it might be possible to infer which input and output layer to use if every task has a unique combination of input and output space. But in general this is not the case and therefore a task identifier is required in order to perform inference. This necessity severely limits the applicability of ITL, since in practice as such a task identifier is usually not available.

A simple example for ITL would be a sequence of two tasks where the first task requires classification of RGB images into three classes, e.g. classifying cars by their brand or type, and the second task requires the regression of a real valued scalar quantity, e.g. their fuel consumption. Clearly $\mathcal{X}_1 = \mathcal{X}_2$. But it is also obvious that $\mathcal{Y}_1 = \{0, 1, 2\} \neq \mathcal{Y}_2 = \mathbb{R}^+$. The DNN used in this example therefore needs two different output layers, one with three neurons featuring a softmax activation function and another with just one neuron with a Rectified Linear Unit (ReLU) activation function. Given an input car image during inference it is then also necessary two know the task in order to select the correct output layer.

2.3. Online vs. Offline Continual Learning

Most research considers continual learning sequences whose tasks are subsets of datasets with thousands and even millions of training examples. These are then split in a relatively small number of tasks. As a result individual tasks themselves can feature thousands of training examples that are assumed to be accessible simultaneously in any order. These two properties enable the use of common training strategies for DNNs, where multiple epochs can be completed on each task and data in mini batches can be sampled uniformly at random from the current task. Such continual learning sequences are said to be offline, since each training example of a task can be revisited multiple times. Offline continual learning is typically characterized by independent and identically distributed (i.i.d.) samples because all training examples are independently drawn at random. The aforementioned scenarios ICL, IDL and ITL belong to the category of offline continual learning.

In contrast to this, data becomes available in a stream of individual training examples or small batches in online continual learning and each example can only be observed once. Furthermore, the data in such a stream can be ordered and therefore it is in general not i.i.d.. An example for such an online continual learning sequence for supervised image classification under the ICL scenario can be constructed by splitting any of the datasets introduced in section 3.6 into subsets containing one or more classes. But in contrast to offline continual learning the corresponding method under test is allowed to visit each training example of these subsets only once. In addition to this, the training data is not shuffled and even intentionally ordered, i.e. data might be ordered by their class labels. Another important difference between online and offline continual learning is the absence of clear task boundaries in online continual learning. In general there is no abrupt change in the stream of data between what is known as tasks in offline continual learning. In the case of supervised image classification, for example, there might be a gradual transition from one class to another in the stream of data. For such a transitional region, the probability of observing a sample from the current class might gradually decrease while it becomes increasingly likely to observe a sample of the new class. Furthermore, classes encountered previously might be absent from the stream for a while and then be encountered again later on. As a consequence, it is not possible to define tasks in the same sense as for offline continual learning as a clearly defined part in the sequence of encountered data.

The absence of clear task boundaries in online continual learning prevents a simple definition of the ICL, IDL and ITL scenarios. But the general distinction between learning new classes, domains and tasks can still be applied to online continual learning. Most literature [56, 81, 83, 84, 122, 126] and chapter 5 of this thesis, however, consider a stream of non i.i.d. data where new classes are introduced one at a time with clear transitions. This resembles an online version of ICL. All other chapters of this thesis address offline continual learning.

2.4. How Is Continual Learning Different from Related Fields?

Although continual learning with DNNs has only recently seen renewed interest, the idea of incremental learning and transferring knowledge has been explored in many other fields of machine learning. A comparison of commonalities and differences in their problem settings, goals and approaches is therefore important to understand the context of this thesis. To facilitate this, an overview of different related fields and concepts is given in the following.

Incremental Learning in General Given that the ability to incrementally learn is an important part of the learning process for humans, it is not surprising that imitating it using machine learning methods proceeding DNNs has been explored before. While some of them are inherently capable of incremental learning by design like the k-Nearest Neighbours (KNN) classifier [9], others need to be adapted or changed in some way to support it. Algorithms for incrementally expanding and updating a decision tree in a computationally efficient way, for example, were already studied by Schlimmer and Fisher in 1986 [7]. Another widely used method for classification is the Support Vector Machine (SVM) [12]. An incremental version of it was proposed by Domeniconi and Gunopulos in 2001 [18]. This shows that continual learning is neither a new concept nor restricted to DNNs in any way. But if they are to replace other traditional machine learning methods, it is necessary to overcome catastrophic forgetting and enable continual learning.

Transfer Learning Transfer learning, which studies transferring knowledge between models trained in different but related environments, is also a broad and well established field. Similar to incremental learning, it is not limited to DNNs but can also be applied as a general concept to many machine learning methods and algorithms. A recent comprehensive survey on it is given by Zhuang et al. in [130]. Although the transfer of knowledge is also an integral part of continual learning and can therefore be regarded as a common goal, both fields differ significantly. In transfer learning, the knowledge learned on a source domain is transferred to a target model operating in a related but different domain with the goal of improving the model performance in the target domain only. After the knowledge transfer between these two, the performance of the target model on the source domain is irrelevant. In fact, the target model might not even be capable of operating on the source domain at all. In continual learning, on the other hand, the accumulation over many tasks is critical and therefore a model capable of it must not only be able to perform well on the most recently learned task but all previously encountered ones as well. An example for transfer learning is unsupervised domain adaptation [118] where no labels in the target domain exist. There is also a continual learning variant of domain adaptation that aims at adapting to a sequence of domains [131, 132].

Multitask Learning Multitask learning is another field of machine learning that aims at exploiting the knowledge of multiple related tasks in order to improve the performance of a learning system on all of them. Similar to continual learning, multiple tasks have to be learned and knowledge has to be transferred between them. Much like in continual learning, multiple potentially very different tasks that are represented by individual training datasets have to be learned. But in contrast to continual learning, access to the datasets of previous tasks is not restricted when learning a new task. This means that joint training on all tasks simultaneously can be performed, which completely avoids the phenomenon of catastrophic forgetting. Recent surveys on multitask learning, methods on how to achieve it and applications are given in [127] and [124].

Federated Learning Federated learning is concerned with training machine learning models in a decentralized way. Data is processed locally on devices like smartphones with the goal of combining their individual knowledge in one shared model while minimizing the communication overhead between devices. Similar to continual learning, access to the complete dataset is limited in federated learning. But instead of a strictly temporal limitation that arises due to the necessity of learning on a sequence of tasks, the data being stored on isolated device leads to a spatial and potentially also a temporal limitation in federated learning. Recent surveys on this field include [113] and [104]. While the main challenge of continual learning with DNNs is overcoming catastrophic forgetting and transferring knowledge between tasks, the challenges of federated learning are the minimization of communication overhead between devices during learning, potentially very different characteristics regarding performance and available memory of devices, non-identically distributed data across them and preserving the privacy of device users.

2.5. Continual Learning in the Human Brain

Understanding the mechanisms that allow the human brain to acquire new knowledge effectively over long sequences of incremental learning [20] is important as it could hint to similarly effective methods and algorithms that allow DNNs to perform in the same way. While a comprehensive overview over the biological aspects of continual learning that summarizes many primary sources is given by Parisi et al. in [89], the following is restricted to the key aspects of continual learning in the human brain.

Although the human brain is not fully understood yet, research shows that its learning process features an intricate balance between stability, which prevents knowledge from being forgotten, and plasticity, which enables new knowledge to be incorporated. This interaction of both aspects not only varies between different parts of the brain but also over time [45, 86]. Since both mechanisms act against each other during the learning process, finding exactly the right balance between them is known as the stability-plasticity dilemma [22]. The brain is particularly plastic in early stages of development and becomes more stable over time but even in adulthood some plasticity remains [46].

A simple but widely known theory of neuron plasticity by Hebb [2], also known as Hebb's rule, states that the connection between two neurons is strengthened when one repeatedly activates another neuron. But learning using Hebb's rule is inherently unstable as there is no limit on how strong such connections can grow and therefore additional mechanisms for stabilizing the learning process are required [15, 17]. More recent research suggests a different computational model in which the activity of neurons in each area of the brain is a combination of feed-forward excitation, feedback and prior expectation [52]. This model, in addition to feed-forward functions, does also allow for generative models, much like memory recall, and enables the combination of prior expectation with current inputs in order to predict forward in time.

While these theories propose computational models for neural plasticity and to some degree stability, they do not detail the process of remembering previous experiences. The theory of Complementary Learning Systems (CLS) in the Hippocampus and Neocortex [16] gives insight into how these two parts of the human brain interact in order to achieve rapid learning while remembering previous experiences. Although both of them learn in a Hebbian like way, their learning speed and function in the overall process differs greatly. The Hippocampus is capable of rapid learning given new experiences and encodes them in sparse representations. The information incorporated into the Hippocampus can then be played back over time to the Neocortex which exhibits much slower learning and is responsible for long term memory retention. These two systems therefore complement each other with one being capable of rapid short term learning and the other responsible for long term memory. Some methods for continual learning with DNNs, like the one proposed in [110], even motivate their approach on these results from neuroscience and try to replicate a similar form of replay through the use of generative models. A more detailed description of such methods, which fall under the category of pseudo rehearsal or generative replay, is given in section 3.4.1.

The process of continual or incremental lifelong learning performed by the human brain has been studied quiet extensively in the field of neuroscience. But transferring and utilizing the mechanisms that are responsible for continual learning in the brain directly to DNNs poses some challenges. While DNNs might be loosely inspired by biological neural networks, their underlying working principal and the way of training them are fundamentally different [92, 105]. The widely used Backpropagation (BP) algorithm for training a DNN, whose invention is often attributed to Rumelhart, Hinton, and Williams [6], seems biologically implausible mainly because of three aspects.

1. BP uses an error signal that is propagated back throughout the whole network. This implies that weight updates of a neuron depend on all neurons and their activation proceeding it. This global dependency of weight updates is in contrast to the local learning performed in biological neural networks.

2. BP uses the same weights for propagating information forward and error signals backward, which implies symmetric connections in both directions. But although there is evidence for bidirectional connections in biological neural networks, there is no direct evidence for them being symmetric.

3. BP requires the function of a neuron to be differentiable. While the typical simplistic models of neurons in DNNs are differentiable, finding a derivative for their biological counterparts is difficult as their function depends on discrete spikes [21].

But despite of these fundamental differences between DNNs and biological neural networks findings from neuroscience might still lend themselves as valuable inspiration for finding new algorithms or modifications to BP for enabling continual learning in a similar way as they have inspired the creation of DNNs in the first place.

Chapter 3.

Literature Overview

Because continual learning with DNNs is a growing and active field of research, the following is not intended to be a complete presentation of the literature on this topic. It rather serves as an overview over the different types of approaches and a short introduction to selected canonical examples of those. An overview over which method can be applied to which continual learning scenario is presented in Tables 3.1 and 3.2. Besides these three scenarios an indication whether a method can be applied to online continual learning, where data arrives as a stream, or offline continual learning, where all data of a task can be accessed simultaneously, is given. Section 3.5 in addition presents metrics used to evaluate continual learning methods while sections 3.6 and 3.7 introduce some commonly used benchmark datasets and neural architectures in the field.

3.1. Regularization-based Methods

Given that catastrophic forgetting in DNNs is a consequence of their inherent plasticity and lack of stability, using some sort of regularization on their weights to stabilize them is an intuitive approach. It is therefore no surprise that regularization-based methods were proposed among the first when continual learning gained renewed interest [53, 63]. In general methods of this type share the common approach of penalizing the deviation of weights once they have been trained on a task in order to prevent them from changing too much and causing forgetting. Considering a continual learning problem according to Definition 3, a regularization-based

Table 3.1.: Overview of different methods and their applicability to scenarios and online continual learning.

Method	**ITL**	**IDL**	**ICL**	**Online**	**Offline**
			Regularization-based		
EWC [53]	✓	✓	✓		✓
R-EWC [70]	✓	✓	✓		✓
OSLA [75]	✓	✓	✓		✓
SI [63]	✓	✓	✓		✓
RWALK [66]	✓	✓	✓		✓
MAS [64, 82]	✓	✓	✓	✓	✓
LwF [55]	✓	✓	✓	✓	✓
EbLL [57]	✓	✓	✓		✓
DMC [112]	✓	✓	✓		✓
IMM [54]	✓	✓	✓		✓
CL-LROS [98]	✓			✓	✓
OGD [101]	✓	✓	✓		✓
			Structural		
PNN [48]	✓				✓
PC [77]	✓	✓	✓		✓
PN [72]	✓				✓
HAT [78]	✓				✓
RCL [80]	✓				✓
ILDA [76]	✓				✓
			Bayesian		
NCL [117]	✓	✓	✓		✓
VCL [74]	✓	✓	✓		✓
GVCL [119]	✓	✓	✓		✓

Table 3.2.: Overview of different methods and their applicability to scenarios and online continual learning.

Method	**ITL**	**IDL**	**ICL**	**Online**	**Offline**
			Memory-based		
DGR [59]	✓	✓	✓		✓
GEM [56]	✓	✓	✓	✓	✓
AGEM [84]	✓	✓	✓	✓	✓
GSS [83]	✓	✓	✓	✓	✓
ESS [126]	✓	✓	✓	✓	✓
MIR [81]	✓	✓	✓	✓	✓
ASER [122]	✓	✓	✓	✓	✓
BiC [95]	✓	✓	✓		✓
DCGM [129]	✓	✓	✓		✓
HAL [114]	✓	✓	✓		✓
CCMCL [125]	✓	✓	✓		✓
DPNCL [133]	✓	✓	✓		✓
ICaRL [58]	✓	✓	✓		✓
LUCIR [87]	✓	✓	✓		✓
PODNet [99]	✓	✓	✓		✓
TPCIL [109]	✓	✓	✓		✓
BSIL [116]	✓	✓	✓		✓
MBSIL [116]	✓	✓	✓		✓

method can be formalized as solving the sequence of optimization problems

$$\boldsymbol{\theta}_k^\star = \arg\min_{\boldsymbol{\theta}} \sum_{j=1}^{k} \mathcal{L}_j(\mathcal{T}_j, \boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta}, \boldsymbol{\theta}_{k-1}^\star, \boldsymbol{\theta}_{k-2}^\star, \ldots, \boldsymbol{\theta}_1^\star), \tag{3.1}$$

where $\mathcal{R}$ is a regularization depending not only on the weights $\boldsymbol{\theta}$ that are optimized in the k-th step but also on the optimal ones $\boldsymbol{\theta}_{k-1}^\star, \boldsymbol{\theta}_{k-2}^\star, \ldots, \boldsymbol{\theta}_1^\star$ from every previous step. As the goal of methods in this category is to prevent deviations of weights from their optimal value, $\mathcal{R}$ is generally a distance measure like the squared Euclidean or 2-norm, i.e. $\|\cdot\|_2^2$. But while this choice of distance measure might be suitable for preventing weights from deviating too much from their optimal values, it penalizes deviations with the same strength for all weights. This implies that every weight is the equally important for solving a particular task, which might not be true in general. A regularization-based method should therefore be capable to identify the importance of individual weights for a given task in order to only prevent their deviation (stability) while allowing less important weights to change freely in order to learn new tasks (plasticity). Canonical examples of such methods are Elastic Weight Consolidation (EWC) and Synaptic Intelligence (SI).

Elastic Weight Consolidation (EWC) Kirkpatrick et al. argue in [53] that DNNs are over-parameterized and many different parameter configurations can result in a similar performance. It is likely that the solution to a task that is to be learned lies in close proximity of the solution found on previous tasks. They further justify the use of a regularization by adopting a Bayesian perspective and pointing out that the posterior over weights of a DNN conditioned on some dataset is given by

$$\ln p(\boldsymbol{\theta}|\mathcal{T}) = \ln p(\mathcal{T}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) - \ln p(\mathcal{T}), \tag{3.2}$$

where $\ln p(\mathcal{T}|\boldsymbol{\theta})$ is equivalent to the negative loss function $-\mathcal{L}(\mathcal{T}, \boldsymbol{\theta})$ under the maximum likelihood approach to train a DNN. Assuming two independent tasks $\mathcal{T}_{k-1}$ and $\mathcal{T}_k$, this posterior of the weights conditioned on both tasks is then given by

$$\ln p(\boldsymbol{\theta}|\mathcal{T}_{k-1}, \mathcal{T}_k) = \ln p(\mathcal{T}_k|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}|\mathcal{T}_{k-1}) - \ln p(\mathcal{T}_k). \tag{3.3}$$

From this derivation of Kirkpatrick et al. to be valid, the formal definition of what they refer to as two tasks being independent can be deduced as

$$p(\mathcal{T}_{k-1}, \mathcal{T}_k) = p(\mathcal{T}_{k-1})p(\mathcal{T}_k) \text{ and } p(\mathcal{T}_{k-1}, \mathcal{T}_k|\boldsymbol{\theta}) = p(\mathcal{T}_{k-1}|\boldsymbol{\theta})p(\mathcal{T}_k|\boldsymbol{\theta}). \tag{3.4}$$

Using (3.3) Kirkpatrick et al. further argue that the posterior conditioned on both tasks depends only on the loss function of task $-\ln p(\mathcal{T}_k|\boldsymbol{\theta}) = \mathcal{L}_k(\mathcal{T}_k, \boldsymbol{\theta})$, i.e. the negative log-likelihood, and all information about task $\mathcal{T}_{k-1}$ is absorbed into the posterior $p(\boldsymbol{\theta}|\mathcal{T}_{k-1})$ including the importance of individual weights to task $\mathcal{T}_{k-1}$. EWC therefore uses a Laplace approximation based on [13] to this posterior, which uses a mean given by the optimal weights $\boldsymbol{\theta}^\star_{k-1}$ found on task $\mathcal{T}_{k-1}$ and the precision given by the diagonal elements of the Fisher Information Matrix (FIM) $\mathbf{F}_{k-1}$.

Definition 6: Fisher Information Matrix

Given a log-likelihood $\ln p(\mathbf{x}|\boldsymbol{\theta})$ over the random variable $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$, the Fisher Information Matrix (FIM) $\mathbf{F}$ is defined as a matrix containing the variance of a so called score function $\frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\mathbf{x}|\boldsymbol{\theta})$. Since the mean of this score function vanishes, it is given by

$$
\begin{aligned}
[\mathbf{F}]_{i,j} &= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}\left[\left(\frac{\partial}{\partial \theta_i} \ln p(\mathbf{x}|\boldsymbol{\theta})\right)\left(\frac{\partial}{\partial \theta_j} \ln p(\mathbf{x}|\boldsymbol{\theta})\right)\right] &\quad (3.5)\\
&= -\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})}\left[\frac{\partial^2}{\partial \theta_i \partial \theta_j} \ln p(\mathbf{x}|\boldsymbol{\theta})\right] &\quad (3.6)
\end{aligned}
$$

and measures the sensitivity of $\ln p(\mathbf{x}|\boldsymbol{\theta})$ with respect to changes in $\boldsymbol{\theta}$. Under some mild regularity conditions it is equivalent to the negative expectation of the Hessian matrix of the log-likelihood.

Given two consecutive tasks $\mathcal{T}_k$ and $\mathcal{T}_{k-1}$, the regularization used in EWC is therefore given by

$$
\begin{aligned}
\mathcal{R}(\boldsymbol{\theta}_k, \boldsymbol{\theta}^\star_{k-1}) &= \frac{\lambda}{2}(\boldsymbol{\theta}_k - \boldsymbol{\theta}^\star_{k-1})^T \operatorname{diag}(\mathbf{F}_{k-1})(\boldsymbol{\theta}_k - \boldsymbol{\theta}^\star_{k-1})\\
&= \frac{\lambda}{2}\sum_i [\mathbf{F}_{k-1}]_{ii}\left([\boldsymbol{\theta}_k]_i - [\boldsymbol{\theta}^\star_{k-1}]_i\right)^2, &\quad (3.7)
\end{aligned}
$$

where $\lambda \in \mathbb{R}$ is a parameter for controlling the regularization strength and $\mathbf{F}_{k-1}$ is the FIM estimated on task $\mathcal{T}_{k-1}$. For training on a sequence of more than two tasks, Kirkpatrick et al. propose to use a separate regularization term for each previously learned task or to combine all of those into one regularization term. The latter is possible since a sum of two quadratic terms in the form of (3.7) is itself a quadratic term.

Given that the FIM according to Definition 6 under mild regularity conditions is equivalent to the expected negative Hessian matrix of the log-likelihood from a previous task, an intuitive interpretation of EWC can be given as follows. Estimating the importance of parameters for a particular task using the Fisher Information assigns high values to those parameters who are likely to cause a significant decrease of the log-likelihood if these parameters are changed. Weighting a regularization that penalizes deviations from their optimal values found on the previous task therefore only consolidates parameters that are important to that task while allowing the remaining parameters with low Fisher Information to move freely.

Synaptic Intelligence (SI) Zenke, Poole, and Ganguli also propose a regularization-based method for overcoming catastrophic forgetting in [63]. While their approach utilizes the same functional form as EWC, namely a weighted sum of quadratic terms centered at the parameter values found during training on previous tasks, their method for determining the importance of individual parameters differs significantly. Instead of estimating it in a separate stage when training on a task is completed, as in EWC, Zenke, Poole, and Ganguli propose an online estimate that is calculated simultaneously while training the DNN. They base their approach on determining the contribution of each parameter to decrease the training loss as it moves through the parameter space. The authors argue that for an infinitesimal change $\boldsymbol{\delta_\theta}$ of parameters $\boldsymbol{\theta}$ the loss changes as

$$\mathcal{L}(\boldsymbol{\theta} + \boldsymbol{\delta_\theta}) - \mathcal{L}(\boldsymbol{\theta}) \approx \boldsymbol{\delta_\theta} \nabla \mathcal{L}(\boldsymbol{\theta}), \tag{3.8}$$

where the Nabla operator ∇ is used to evaluate the gradient of a loss function $\mathcal{L}$ at position $\boldsymbol{\theta}$. The authors further point out that the gradient is a conservative vector field[1] and the total change in loss therefore can be obtained by integrating over these infinitesimal changes along the parameter trajectory C through the parameter space. The total change in loss is then given by

$$\mathcal{L}(\boldsymbol{\theta}_k^\star) - \mathcal{L}(\boldsymbol{\theta}_k^0) = \int_C \nabla \mathcal{L}(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} = \sum_i \int_{C_i} \frac{\partial}{\partial \theta_i} \mathcal{L}(\boldsymbol{\theta}) \mathrm{d}\theta_i \equiv - \sum_i \left[\omega_k\right]_i, \tag{3.9}$$

where the path integral of the complete gradient vector along the trajectory is split up into a sum of contributions $[\omega_k]_i$ from each parameter θ_i. Note that the minus sign before the last sum is introduced by the authors as one is typically interested in decreasing the loss. For a practical implementation the authors further propose to approximate $[\omega_k]_i$ online as a running sum of the product between parameter update steps and the gradient. Since usually

[1]A vector field that results from taking the gradient of a function. Its line integral is path independent, i.e. changing the path between two endpoints of the line integral does not change its value.

Stochastic Gradient Descent (SGD) is used to train DNNs, the estimated gradient is noisy and therefore $[\boldsymbol{\omega}_k]_i$ will typically overestimate the true contribution of a parameter according to the authors. This source of noise is explicitly stated by Zenke, Poole, and Ganguli but an additional noise source results from the gradient not being constant along an update step of the parameters. Simple measures for preventing this source of noise in the estimate of $[\boldsymbol{\omega}_k]_i$ are therefore the choice of a small learning rate, which results in shorter update steps, or to introduce intermediate steps for which the gradient can be considered constant during every parameter update.

In addition to a parameter's contribution to decrease the loss, the authors argue that the distance $[\boldsymbol{\Delta}_k]_i \equiv \left[\boldsymbol{\theta}_k^\star - \boldsymbol{\theta}_k\right]_i$ that the i-th parameter moves throughout the training process is another measure of its importance for a particular task. Using these two measures the authors propose a regularization that has the same minimum as the loss function of previous tasks and yields the same $\boldsymbol{\omega}_k$ over a parameter distance $\boldsymbol{\Delta}_k$ as

$$\mathcal{R}(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k-1}^\star) = \lambda \sum_i \Omega_i^k ([\boldsymbol{\theta}_k]_i - \left[\boldsymbol{\theta}_{k-1}^\star\right]_i)^2, \tag{3.10}$$

where $\lambda \in \mathbb{R}$ is a parameter for controlling regularization strength. The individual weights are given by

$$\Omega_i^k = \sum_{j=1}^{k} \frac{[\omega_k]_i}{[\Delta_k]_i^2 + \epsilon}, \tag{3.11}$$

where $\epsilon \in \mathbb{R}$ is a small constant used to bound the weighting term as $\boldsymbol{\Delta}_k \rightarrow \mathbf{0}$. The authors also point out that squaring the distance a parameter moved throughout training is used to ensure that the regularization term carries the same unit as the loss function. While the contributions $\boldsymbol{\omega}_k$ are updated continuously throughout the training process, the cumulative weights Ω_i^k are only updated when the training on task $\mathcal{T}_k$ is complete. After these weights have been updated the contributions $\boldsymbol{\omega}_k$ are set to zero. In contrast to EWC the importance weights are updated in a cumulative way and therefore there are no separate importance weights for different tasks which need to be stored or combined.

3.2. Structural Methods

Instead of relying on additional regularization terms, structural methods for continual learning aim at preventing catastrophic forgetting through changes to the architecture of a DNN. This typically involves freezing or slowing down learning of weights that were learned on previous tasks in order to prevent forgetting. But since this also prevents learning new tasks, additional capacity, typically in the form of new layers, needs to be added for each new task. In addition to this, connections between parts of a DNN that have been trained on previous tasks and newly added layers are utilized to facilitate knowledge transfer from previous to new tasks.

Progressive Neural Network (PNN) A typical example of a structural method for continual learning is a PNN proposed by Rusu et al. in [48]. It instantiates so called columns with randomly initialized weights for each task that is to be learned. A column, in this case, is a complete DNN capable of solving the task at hand. After a column is trained on the task its weights are frozen in order to completely prevent forgetting. In order to enable the transfer of knowledge learned on previous tasks into the current task, lateral connections to feature maps at the same network depth to all previously learned columns are used. The input to a layer in the most recent column is therefore not only the output of a layer preceding it but also the outputs of all layers from previously learned columns at the same network depth. Although this approach completely eliminates catastrophic forgetting, its application is limited not only by a rapidly increasing computational complexity due to the lateral connections but also by requiring knowledge about the task identity during inference in order to select the output of a corresponding column. This restricts PNNs to be applied only to the rather limited ITL scenario discussed in section 2.2. Rusu et al. additionally propose so called adapters, which replace direct linear connections with non-linear ones, in order to improve initial conditioning and to reduce the amount of features transferred to the next column. These adapters are implemented as a single dense or convolutional layers with a kernel size of 1×1 and their size is chosen such that the amount of parameters from lateral connections is roughly equal to the amount of parameters in the first column.

Progress and Compress (PC) Although not strictly a structural method, PC proposed by Schwarz et al. in [77] is closely related to PNNs as it combines its basic principle of using columns for different tasks with the regularization approach of EWC. But instead of using one for each individual task a knowledge base column is used to combine all previously learned columns while an active column is used to learn new tasks. Overall this results in a two step process, where the active column with lateral connections to the knowledge base first is used to efficiently learn a new task and then incorporated or compressed into the knowledge base. During this compression phase previously learned weights are protected by a modified version of EWC that uses a running sum of all Fischer Information matrices instead of one for each task. While this approach alleviates the problems with a quickly growing number of parameters as in PNNs, it also eliminates the need to know the task identity during inference as the active column is incorporated into the knowledge base via distillation [39].

3.3. Bayesian Methods

Adopting a Bayesian perspective and analyzing the continual learning problem leads to another group of methods for avoiding catastrophic forgetting. These are not only able to express uncertainty about the weights learned during the training of a DNN but also provide a natural solution to incremental learning. First, a prior distribution $p(\boldsymbol{\theta})$ over the weights of a DNN $\boldsymbol{\theta}$ is chosen. The not normalized posterior after observing k tasks is then given according to Bayes rule as

$$p(\boldsymbol{\theta}|\mathcal{T}_{1:k}) \propto p(\boldsymbol{\theta}) \prod_{j=1}^{k} p(\mathcal{T}_j|\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathcal{T}_{1:k-1}) p(\mathcal{T}_k|\boldsymbol{\theta}), \tag{3.12}$$

where $\mathcal{T}_{1:k}$ is a short notation for $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, $p(\mathcal{T}_j|\boldsymbol{\theta})$ is the likelihood of task $\mathcal{T}_j$ and it is assumed that tasks are independent of each other. With this formulation a recursion for finding the posterior on task k based on the previous one can be identified as a product of the posterior found on task $k-1$ and the current likelihood. While this approach seems very intuitive and elegant, its application poses one major challenge common to most Bayesian methods: Finding a tractable normalized posterior or at least an approximation to it. Note that the posterior in (3.12) is only proportional to a prior times the likelihoods of all k tasks and lacks the normalization by $p(\mathcal{T}_{1:k})$.

Variational Continual Learning (VCL) Nguyen et al. propose their method VCL in [74], which is based on online variational inference [19] through the use of a projection operator

$$q_k(\boldsymbol{\theta}) = \arg\min_{q \in Q} \mathrm{KL}\left(q(\boldsymbol{\theta}) \middle\| \frac{q_{k-1}(\boldsymbol{\theta}) p(\mathcal{T}_k|\boldsymbol{\theta})}{Z_k}\right), \tag{3.13}$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence, Z_k is the intractable normalization constant and q is restricted to the set Q of allowed posteriors. It is applied recursively starting with $q_1(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ in order to approximate the true posterior $p(\boldsymbol{\theta}|\mathcal{T}_{1:k})$ of task k. Note that

Z_k is not needed for finding the optimum of this optimization problem making this approach tractable. For this, the Evidence Lower Bound (ELBO) given as

$$\mathcal{L}(q_k(\boldsymbol{\theta})) = \mathbb{E}_{q_k(\boldsymbol{\theta})}\left[\ln p(\mathcal{T}_k|\boldsymbol{\theta})\right] - \mathrm{KL}(q_k(\boldsymbol{\theta})\|q_{k-1}(\boldsymbol{\theta})) \tag{3.14}$$

is maximized with respect to the parameters of $q_k(\boldsymbol{\theta})$. While the KL divergence term in the ELBO can be evaluated analytically, Monte Carlo (MC) sampling in combination with the local reparameterization trick [41] is used to maximize the expectation term. Assuming the true posterior is of the same type as the allowed distributions $\mathcal{Q}$ at every time step, i.e. $p(\boldsymbol{\theta}|\mathcal{T}_{1:k}) \in \mathcal{Q}$, VCL will lead to exact Bayesian inference. In practice this assumption will typically not be true since VCL uses a Gaussian mean-field approximation. Instead it will only perform approximate inference and any errors will accumulate due to its recursive nature. Nguyen et al. therefore propose to store a small set of representative samples from each task that are not used in the recursive approximation of the posterior. Only right before a prediction is made, this set of samples is used to perform one last projection step following (3.12) in order to correct for errors introduced by the approximate inference and to prevent any further forgetting. For more details on how exactly VCL can be applied to continual learning with DNNs we refer the reader to [74].

Generalized Variational Continual Learning (GVCL) Building upon VCL, Loo, Swaroop, and Turner propose GVCL in [119] as a generalization of this approach and show that under certain conditions online EWC can be recovered as a special case of their method. For this, they point out that in practice the allowed posteriors $\mathcal{Q}$ are typically too simple to effectively approximate the true posterior. An experimentally justified method for improving the quality of this approximation is given by weighting the KL divergence term in the ELBO with a parameter $0 < \beta < 1$. Assuming Gaussian posteriors, i.e. $q_k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, Loo, Swaroop, and Turner show that in the limit $\beta \longrightarrow 0$ GVCL recovers online EWC.

3.4. Memory-based Methods

A simple but very effective method for overcoming catastrophic forgetting is to store past experience from previous tasks which can then be retrieved later and mixed with data of a new task. Although this approach is rather simple, there are several different variants of it and many methods that achieve state-of-the-art results utilize some sort of memory for storing and rehearsing previously encountered training examples. Methods of this category can be divided in roughly three different subcategories depending on the form in which past experience is stored.

In pseudo rehearsal past experience is stored in the form of a generative model that is trained to reproduce training examples from previously learned tasks. While these methods, in theory, are able to very efficiently store and replay past experience, their success strongly depends on the quality of generated training examples. A Generative Adversarial Network (GAN) [35], for example, is a very capable generative model that can be used for pseudo rehearsal [59] but on the other hand is notoriously difficult to train.

Core set methods, in contrast, store training examples directly. Methods of this category usually use some sort of selection mechanism for determining which data is most relevant for a given task and store only those in the memory. These mechanisms can be rather simple, e.g. clustering or distance based, but also be more complex and involved in how they determine which data is to be stored. Methods in this category can further differ in how they use the core set to avoid catastrophic forgetting.

Approaches of the third category, distillation methods, neither learn a generative model nor store training examples directly but instead learn synthetic ones. In contrast to pseudo rehearsal with generative models the process of learning these synthetic examples is performed in such a way that catastrophic forgetting is minimized. The synthesized data might therefore lack in visual quality when compared to GANs, for example, but can be better at preventing forgetting when used for rehearsal. Compared to core set methods, past experience can be stored more efficiently by synthesizing data more informative examples, which can lead to an improved performance given a fixed size memory.

Memory-based methods also feature some resemblance of the processes for remembering in the human brain as discussed in section 2.5, since it also seems to employ some sort of memory for storing past experience that is repeatedly played back in order to prevent forgetting. This, in addition to their simplicity and strong performance, makes memory-based methods an interesting and promising approach to continual learning. In the following some typical examples of these categories are presented in more detail.

3.4.1. Pseudo Rehearsal

Deep Generative Replay (DGR) Although there were some previous attempts at using generative models to mimic the behavior of a CLS similar to the Hippocampus and Neocortex in the human brain, Shin et al. were the first to propose a method DGR that utilizes a GAN in [59]. Their framework is centered around so called scholars, where each scholar consists of a generator, represented by a GAN, and the solver, which is typically a DNN for solving tasks. Training a scholar on some task $\mathcal{T}_k$ involves two steps.

In a first step, the generator is trained to reproduce samples from the data generating distribution of inputs from the current task $\mathcal{T}_k$ and all previously learned tasks $\mathcal{T}_{1:k-1}$. While the data of task $\mathcal{T}_k$ is available in this step, data of previous tasks has to be generated by the generator trained on the previous task $\mathcal{T}_{k-1}$. This makes training the generator a recursive process where all except data of the current task is generated using the generator trained on the previous task $\mathcal{T}_{k-1}$. Imperfections in these generated images can therefore accumulate over the repeated steps and cause severe performance degradation. It is therefore critical that the generative model is powerful enough to produce high-quality inputs. Shin et al. use an unconditional Wasserstein GAN with gradient penalty [51] for the generator. Note that as a consequence, it is only able to generate inputs $\mathbf{x}$ but not the corresponding label y.

After training the generator, the solver is trained on data of the current task $\mathcal{T}_k$ and data sampled from the generator that reproduces data from all previously learned tasks $\mathcal{T}_{1:k-1}$. But as a consequence of using an unconditional generative model, the labels of generated samples must be predicted by the solver trained on task $\mathcal{T}_{k-1}$. Similar to the training of a generator, any errors in the prediction of labels can accumulate over repeated training steps and cause a degradation in performance. Another aspect that needs to be controlled is the ratio of mixing training data from the current task $\mathcal{T}_k$ and data replayed by the generator. For this, Shin et al. propose to minimize

$$\mathcal{L}_k(\mathcal{T}_k, \boldsymbol{\theta}) = r\mathbb{E}_{p_{\mathcal{T}_k}(\mathbf{x},y)}\left[\mathcal{L}(f_k(\mathbf{x};\boldsymbol{\theta}), y)\right] + s\mathbb{E}_{p_{g_{k-1}}(\mathbf{x},y)}\left[\mathcal{L}(f_k(\mathbf{x};\boldsymbol{\theta}), f_{k-1}(\mathbf{x}))\right], \tag{3.15}$$

where r is the ratio of new and $s = 1 - r$ of rehearsal data, $f_k(\mathbf{x}; \boldsymbol{\theta})$ is the current solver, g_{k-1} is the generator and $f_{k-1}(\mathbf{x})$ is the solver trained on the previous task $\mathcal{T}_{k-1}$. Although the authors state that the mixing ratio s for the first task should be zero, they do not provide any information on how to select r on the proceeding tasks. In addition to weighting the losses during training using r, they also propose to weight these in a similar way during testing. While this might be a reasonable assumption to make, it complicates the process of comparing different methods as the testing not only depends on a hyperparameter that might be different in every experiment but it also provides the possibility to drastically reduce the loss for certain parts of the test data. For a discussion on fair and comparable test procedures and metrics, we refer to section 3.5.

Another strength of pseudo rehearsal with DGR is its general formulation that not only allows for many different generative models to be used as the generator but also many different solvers. It is therefore a natural consequence that it can be applied to all three continual learning scenarios described in section 2.2.

3.4.2. Core Set Methods

Core set methods use a small memory to store training examples in their original form and mostly differ in the way that these are selected and used throughout the training process. While there are many different methods that use a core set for continual learning, naive rehearsal with random sample selection (sometimes referred to as experience replay) is the simplest of them. It randomly selects training examples from a task to store in a memory and replaces examples from the buffer at random when its maximum storage capacity is reached. During training, data of the current task $\mathcal{T}_k$ is then mixed with randomly selected data from the memory. Despite its simplicity naive rehearsal is a strong base line that can outperform some much more complex continual learning methods [68] even in the most challenging of the scenarios discussed in section 2.2.

Building on this strong baseline, some methods try to improve various aspects of naive rehearsal. One of these is the way samples are selected for storing in the memory as not all training data might be equally useful for remembering a task. While random selection of naive rehearsal is trivial, determining which samples to store is in general a challenging task as the selected data needs to be diverse but at the same time representative. Although various heuristic methods, such as k-means clustering in input or feature space, the k-center algorithm or herding [29], have been proposed, they are not directly coupled to the continual learning problem and therefore struggle to consistently outperform random selection. In contrast to this, Borsos, Mutny, and Krause reformulate the problem of which samples to select for storing as a bi-level optimization problem and use a proxy model given by a Neural Tangent Kernel (NTK) [69] in order to efficiently solve it. Although this approximation leads to a much more efficient algorithm, the authors note that for a memory with more than 500 samples the computational overhead of their method becomes significant. Despite this, the experimental results of their method indicate that it is capable to consistently outperform random selection at least for small buffer sizes. This is generally desirable as data from more tasks can be stored in fixed size memory while achieving the same performance as random selection. Since their method only effects which samples from the training data are selected for storage, it can be combined with basically any method that makes use of a memory for preventing catastrophic forgetting. As an example of this, Borsos, Mutny, and Krause show that their method combined with VCL can significantly improve its performance when compared to random selection.

Gradient Episodic Memory (GEM) While the method used for selecting data to store is undeniably one way to improve the naive rehearsal base line, how this data is used when training on a new task is equally important. In naive rehearsal data is randomly sampled from the memory and mixed in a certain, usually predefined, ratio with new data to form a mini batch for training. The performance of this rather intuitive way of using the stored data depends on the mixing ratio r. But unfortunately it does not ensure that the loss on previously learned tasks increases whenever data is selected to form a mini batch. In order to ensure that the loss on previous tasks does not increase, Lopez-Paz and Ranzato propose their method GEM in [56]. To this end, they reformulate the optimization problem

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{L}_k(\mathcal{T}_k, \boldsymbol{\theta}) \\ &\text{s.t. } \mathcal{L}_j(\mathcal{M}_j, \boldsymbol{\theta}) \leq \mathcal{L}_j(\mathcal{M}_j, \boldsymbol{\theta}_j)\ \forall\ 1 \leq j < k, \end{aligned} \tag{3.16}$$

where $\mathcal{M}_k$ is the set of data from task $\mathcal{T}_k$ stored in memory. There are two key observations for their reformulation. First, it is unnecessary to store the weights $\boldsymbol{\theta}_k$ learned on previous tasks if during each update step it is guaranteed that the loss on previous tasks does not increase. And second, for small step sizes and a representative memory, an increase in loss can be determined by evaluating the inner product between the gradient w.r.t $\boldsymbol{\theta}$ of two losses: $\mathcal{L}_k(\mathcal{T}_k, \boldsymbol{\theta})$ evaluated on the current training dataset from task k and the loss $\mathcal{L}_k(\mathcal{M}_j, \boldsymbol{\theta})$ evaluated on the data of tasks $1 \leq j < k$ stored in memory. With these observations, the constraints of optimization problem 3.16 can be reformulated as

$$\langle \mathbf{g}, \mathbf{g}_j \rangle = \langle \frac{\partial \mathcal{L}_j(\mathcal{T}_j, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \frac{\partial \mathcal{L}_j(\mathcal{M}_j, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \rangle \geq 0\ \forall\ 1 \leq j < k. \tag{3.17}$$

Using this reformulation GEM reduces to ensuring that each update step when training task $\mathcal{T}_k$ aligns with the gradients evaluated on data from previous tasks $\mathcal{T}_j$ stored in memory $\mathcal{M}_j$ with $1 \leq j < k$. If all constraints are met, the update step can be performed without increasing the loss on previous tasks. But if one or more constraints are violated such an update step would increase the corresponding losses. In this case, GEM projects $\mathbf{g}$ to the closest gradient $\tilde{\mathbf{g}}$ measured using the Euclidean norm that satisfies all constraints. This projected gradient $\tilde{\mathbf{g}}$, which ensures that the loss on previous tasks does not increase, is then used instead for updating the weights of a DNN.

Another more computationally efficient version of GEM called Averaged Gradient Episodic Memory (AGEM) is proposed by Chaudhry et al. in [84]. Instead of enforcing each constraint in (3.16) individually, it only enforces the gradient $\mathbf{g}$ evaluated on the current task to align with the average of gradients $\bar{\mathbf{g}} = \frac{1}{k-1} \sum_{j=1}^{k-1} \mathbf{g}_j$ on all previous tasks. This reduces the multiple constraints of GEM to just one constraint, which can be enforced much more efficiently.

3.4.3. Data Distillation Methods

Another interesting approach to memory-based continual learning are methods based on data distillation. In contrast to core set approaches and pseudo rehearsal, neither original training examples nor a generative model are used for rehearsal but the data itself is learned. To this end, the rehearsal data are treated as trainable parameters and a suitable loss function is minimized by adapting those data using SGD or one of its variants. Through learning the data used in rehearsal, some advantages over these competing categories can be gained. First, learning a generative model that fully captures the data generating distributions of task, a challenging problem for pseudo rehearsal, is not necessary. In addition to this, the process of learning rehearsal data can be formulated in a way that is directly connected to minimizing catastrophic forgetting as it is done in [114]. Second, in contrast to core set methods data for rehearsal can be stored in a more efficient form since a dataset can be distilled into a small set of synthetic examples that essentially capture most of the information present in the original dataset. These properties make data distillation an interesting approach to continual learning as it circumvents the computational challenges associated with pseudo rehearsal while at the same time offering greater storage efficiency when compared to core set methods.

Hindsight Anchor Learning (HAL) An example for a data distillation method is HAL proposed by Chaudhry et al. in [114]. The authors introduce so-called anchor points $\mathbf{e}_k$ that are used to preserve the knowledge learned on task $\mathcal{T}_k$ by preventing a DNN from deviating much in their prediction for $\mathbf{e}_k$ when trained on a new task. As these anchor points are intended to prevent forgetting when training on future tasks, directly learning them is impossible. This would require knowing future tasks including their datasets before they are accessible. Instead, the authors propose to learn an approximation to them based on previous tasks. These are accessible, although only partially, through the memory $\mathcal{M}$ which stores training examples for rehearsal. According to Chaudhry et al. anchor points can therefore be learned by finding the synthetic input that undergoes maximum forgetting when the DNN is fine tuned on data from previous tasks. For each task $\mathcal{T}_k$ one anchor point per class c is therefore learned according to

$$\mathbf{e}_k = \arg\max_{\mathbf{e}} \mathcal{L}(\mathbf{e}, \tilde{\boldsymbol{\theta}}_{k-1}) - \mathcal{L}(\mathbf{e}, \boldsymbol{\theta}_k) - \gamma(\|\boldsymbol{\phi}(\mathbf{e}) - \boldsymbol{\phi}_k\|_2^2), \tag{3.18}$$

where $\tilde{\boldsymbol{\theta}}_{k-1}$ is an approximation to the parameters learned on the previous task $\mathcal{T}_{k-1}$. These are obtained by fine tuning the DNN on data of all previous tasks stored in the memory $\mathcal{M}_{1:k-1}$ for one epoch. By maximizing the difference in loss for an anchor point when using the current weights $\boldsymbol{\theta}_k$ and $\tilde{\boldsymbol{\theta}}_{k-1}$ an anchor point that undergoes maximum forgetting is found. To prevent this anchor point from becoming too different from the original data, a regularization term is used. It uses the Euclidean distance between an embedding of the anchor point $\boldsymbol{\phi}(\mathbf{e})$ and the mean embedding $\boldsymbol{\phi}_k$ of all original data from task $\mathcal{T}_k$. The hyperparameter γ is used to control the strength of this regularization. For creating an embedding the first part of the current DNN is used as a feature extractor $\boldsymbol{\phi}$. The learned anchor points are then used in conjunction with regular examples from the memory for rehearsal.

Dataset Condensation with Gradient Matching (DCGM) While HAL learns anchor points in addition to original training examples for rehearsal, DCGM proposed by Zhao, Mopuri, and Bilen in [129] learns a set of purely synthetic samples. Their main idea is to match the gradients of original training examples with those of synthetic data throughout the process of training a DNN. For this, they propose a distance measure based on the cosine similarity that is given by

$$D(\mathbf{A}, \mathbf{B}) = \sum_{i=1}^{\text{out}} \left(1 - \frac{\mathbf{a}_i^T \mathbf{b}_i}{\|\mathbf{a}_i\|_2 \|\mathbf{b}_i\|_2}\right), \tag{3.19}$$

where $\mathbf{a}_i$ and $\mathbf{b}_i$ are flattened vectors corresponding to the i-th output node, i.e. neuron, of a layer with weight matrix/tensor $\mathbf{A}$ and $\mathbf{B}$. For a dense layer parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{\text{in}\times\text{out}}$ these correspond to individual columns with shape in while for a convolutional layer parameterized by a weight tensor $\mathbf{W} \in \mathbb{R}^{\text{h}\times\text{w}\times\text{in}\times\text{out}}$ they correspond to the flattened gradients for one neuron with shape $\text{h}\times\text{w}\times\text{in}$. The distance $D(\cdot,\cdot)$ is then used to measure the distance between gradients of original training data and synthetic examples. Zhao, Mopuri, and Bilen then minimize it with respect to the latter during training of a DNN via SGD in order to learn the synthetic data. For their method to not only work on one specific initialization of a DNN, they repeat this process for many different randomly initialized DNNs which leads to an algorithm with two nested loops. For more details on their method we refer the reader to [129]. While DCGM is motivated by the authors as a means to speed up Neural Architecture Search (NAS), where instead of training on a full dataset the small set of synthetic examples can be used to evaluate an architectures performance, they also show that it can be used in continual learning. For this, DCGM is used to replace an existing sample selection method in order to build a rehearsal buffer.

3.5. Metrics

Given that continual learning with DNNs can be applied to many different problem settings and deals with learning on a sequence of potentially very different tasks, it is not trivial to effectively and fairly compare competing methods. Even comparing approaches proposed for the same type of machine learning problem using the same datasets can be difficult since there are many different ways a continual learning problem can be constructed, e.g. by using different scenarios as discussed in section 2.2. The work of Prabhu, Torr, and Dokania [107], for example, highlights the different experimental setups used by many publications on continual learning for image classification and shows that some setups are oversimplified and easy to perform on while simultaneously having little practical relevance. But even if the same scenario and experimental setup are used, comparing continual learning methods can be non-trivial as we need to compare the performance on a sequence of tasks rather than a single task.

Díaz-Rodríguez et al. therefore propose six metrics in [67] with the intention to unify the evaluation of continual learning methods and make them more comparable. The authors focus on the widely used setting of continual learning for supervised classification. Hence the underlying metric used to measure the performance is the common classification accuracy.

The first three of these metrics measure the performance of a method and were originally proposed by Lopez-Paz and Ranzato in conjunction with their method GEM discussed in section 3.4.2. They are calculated from a matrix $\mathbf{R}$ whose elements $[\mathbf{R}]_{ij} \in [0, 1]$ contain the accuracy of a considered method on the test dataset of the j-th task after training on the i-th task has completed. With this matrix the authors define their first metric Accuracy (A) as

$$\mathrm{A} = \frac{\sum_{i \geq j}^{M} [\mathbf{R}]_{ij}}{\frac{M(M+1)}{2}} \in [0, 1] \,, \tag{3.20}$$

where M is the number of tasks. Achieving a high A is desirable for all continual learning methods. As this metric measures the performance of a method over the complete sequence of tasks, Díaz-Rodríguez et al. argue that it is well suited for capturing not only the performance on a final task but the "dynamic aspects" of continual learning as well. This is in contrast to the original proposal of Lopez-Paz and Ranzato in [56], which only considers a methods performance at the end of a sequence.

The second metric, Backward Transfer (BWT), measures to what extend learning a task affects all previously learned ones. It is defined as

$$\mathrm{BWT} = \frac{\sum_{i=2}^{M} \sum_{j=1}^{i-1} ([\mathbf{R}]_{ij} - [\mathbf{R}]_{jj})}{\frac{M(M-1)}{2}} \in [-1, 1] \,. \tag{3.21}$$

While catastrophic forgetting currently is the biggest challenge of continual learning with DNNs, in the long term it is desirable to not only avoid performance degradation but to improve on previously learned tasks when learning new ones. BWT explicitly measures this effect and assumes positive values whenever the performance on previously learned tasks improves and negative ones when it degrades after a task has been learned. In general maximizing BWT is therefore the goal of all continual learning methods.

Similar to how learning a task can influence previously learned tasks, it can also affect performance on tasks that are encountered in the future. This can be quantified by Forward Transfer (FWT) which Díaz-Rodríguez et al. define as

$$\mathrm{FWT} = \frac{\sum_{i<j}^{M} [\mathbf{R}]_{ij}}{\frac{M(M-1)}{2}} \in [0, 1] \,. \tag{3.22}$$

Since FWT measures how well a DNN can generalize to unseen tasks in a continual learning sequence, the authors argue that a model capable of zero-shot learning can show signs of such a transfer. In contrast to the original proposal of FWT in [56], Díaz-Rodríguez et al. do not normalize this forward transfer by subtracting the test accuracy of a randomly initialized DNN on each task. Again, maximizing FWT is the goal of all continual learning methods.

In addition to only measuring the performance of a method for continual learning with DNNs, it is also important to consider its efficiency. This includes the size of a model, the number of samples stored in memory and the computational efficiency. In the case of memory-based methods discussed in section 3.4.2, a method that has a larger memory in general performs better than those with a smaller one. Similar arguments can be made for structural methods, where the model size directly influences how much parameters can be assigned to solving a single task, and in general for the computational efficiency. To measure these efficiencies, Díaz-Rodríguez et al. again propose three metrics.

Their Model Size (MS) metric measures the number of parameters, i.e. $||\boldsymbol{\theta}_k||_0$, at each task. Since some methods, e.g. EWC or PNN discussed in sections 3.1 and 3.2, grow the number of parameters or regularization weights for each task, this metric measures the growth in the number of parameters. It is given as

$$\text{MS} = \min\left(1, \frac{1}{M}\sum_{k=1}^{M} \frac{||\boldsymbol{\theta}_1||_0}{||\boldsymbol{\theta}_k||_0}\right) \in [0, 1] \,, \tag{3.23}$$

where the growth is averaged over all M tasks and a method that grows the number of parameters with each task has a MS smaller than one. Although not stated explicitly by the authors, using this definition simplifies the comparison of different continual learning methods since an ideal method would achieve MS close to one. It is however unclear why the authors propose to use the minimum operator since typically the size of a model only grows during continual learning and hence the average of fractions is smaller than one.

In order to quantify the number of samples stored in memory, Díaz-Rodríguez et al. propose Sample Storage Size (SSS). It is based on the number of bits $bit(\mathcal{M}_k)$ required for storing the rehearsal data of task $\mathcal{T}_k$ in memory and the storage requirements for the complete training dataset $bit(\mathcal{T}_{1:M})$. Given these, the SSS is defined by the authors as

$$\text{SSS} = 1 - \min\left(1, \frac{1}{M}\sum_{k=1}^{M} \frac{bit(\mathcal{M}_k)}{bit(\mathcal{T}_{1:M})}\right) \in [0, 1] \,. \tag{3.24}$$

A method that uses an overall smaller memory for all tasks obtains a higher SSS. An ideal continual learning method uses the smallest possible number of bits to store rehearsal data and therefore achieves a SSS close to one. Since it not only measures the memory size at the end of a continual learning sequence but averages the memory size over all tasks, it can also differentiate methods with a growing memory from those with a fixed size one even if their memories have the same storage requirement on the last task.

Methods with a growing buffer therefore achieve a higher SSS. But it is questionable if using a growing memory instead of completely filling it already on the first task is beneficial. In general the performance of a memory-based method is positively correlated with the size of its memory. Utilizing all available storage capacity by filling the buffer completely right from the first task and simply replacing old examples in the buffer on new tasks is therefore a simple but effective way to maximize performance. Before using the SSS to compare memory-based methods it should therefore always be analyzed if a growing memory is really better then a fixed size one.

The last metric proposed Díaz-Rodríguez et al. is Computational Efficiency (CE) which quantifies the number of Multiply-Accumulate (MAC) operations required for learning a task. It is defined as

$$\mathrm{CE} = \min\left(1, \frac{1}{M}\sum_{k=1}^{M}\frac{O_{\updownarrow}(\mathcal{T}_k)\cdot\epsilon}{1+O(\mathcal{T}_k)}\right) \in [0,1]\,, \tag{3.25}$$

where $O_{\updownarrow}(\mathcal{T}_k)$ measures the number of MAC operations for one forward and backward pass on $\mathcal{T}_k$, $O(\mathcal{T}_k)$ measures the total number of MAC operations required to learn $\mathcal{T}_k$ and $\epsilon \geq 1$ is a scaling factor. It essentially measures the average reciprocal number of forward and backward passes required to learn a task. Similar to the other metrics defined above, the reciprocal is used to ensure that the CE is between zero and one. Although a computationally efficient method with a CE close to one is generally preferred, it is also obvious that the actual implementation of a proposed algorithm can have a large influence on its computational efficiency. This makes comparing methods based purely on the CE difficult.

Although the above metrics were proposed in order to unify evaluations and enable fair comparisons of methods for continual learning with DNNs, they are not uniformly used by all authors. Instead, many publications considering the ICL scenario use an evaluation metric proposed by Rebuffi et al. in [58]. Since the authors only consider image classification, their metric is also based on the classification accuracy. They define the Average Incremental Accuracy (AIA) as

$$\mathrm{AIA} = \frac{1}{M}\sum_{k=1}^{M}\mathrm{Acc}_k \in [0,1]\,, \tag{3.26}$$

where Acc_k is the classification accuracy of a tested method evaluated on the test dataset containing all classes that have been learned up to and in the k-th task. Similar to the metric A

Table 3.3.: Overview of presented datasets.

Dataset	#Train	#Validation	#Test	Dimensions	#Classes
MNIST [30]	60000	N/A	10000	$28 \times 28 \times 1$	10
FashionMNIST [62]	60000	N/A	10000	$28 \times 28 \times 1$	10
SVHN [31]	73257	N/A	26032	$32 \times 32 \times 3$	10
CIFAR10 [27]	50000	N/A	10000	$32 \times 32 \times 3$	10
CIFAR100 [27]	50000	N/A	10000	$32 \times 32 \times 3$	100
ImageNet [43]	1281167	50000	100000	Variable	1000

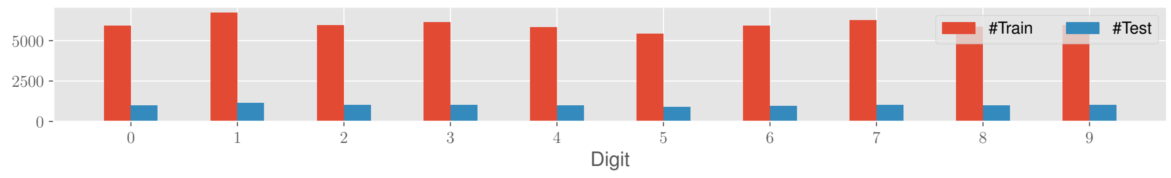


Figure 3.1.: Number of examples per digit in the training and test datasets of MNIST.

defined in (3.20), it considers not only the performance on the last task but also all intermediate ones as well and an ideal continual learning method achieves an AIA close to one. Both metrics are scalar and can therefore be used to quickly compare methods. But despite this similarity they are generally different as A averages the accuracy evaluated on test datasets of individual tasks while AIA averages the accuracy evaluated on a cumulative test dataset that grows larger with every task. Only if all tasks in a continual learning sequence contain an equal number of examples in their test datasets both metrics are equivalent as well.

3.6. Datasets

Continual learning with DNNs covers many different problem settings and scenarios with a diverse and open set of datasets. Due to the limited scope of this work, only supervised image classification is considered as it covers a large body of recently published research. In the following section several commonly used datasets, which are used to create continual learning sequences following the scenarios discussed in section 2.2, are presented. In addition to this, Table 3.3 provides an overview over the discussed datasets and their properties.

MNIST The MNIST dataset proposed by LeCun, Cortes, and Burges in [30] is arguably one of the simplest and most widely used datasets for image classification. It contains centered gray-scale images of handwritten digits with a resolution of 28×28 pixel and is split into 60000 training and 10000 test images. The number of examples per class is not exactly balanced as shown in Figure 3.1. But the imbalance is comparatively small and usually neglected in practice.

FashionMNIST Another simple image classification dataset is FashionMNIST proposed by Xiao, Rasul, and Vollgraf in [62]. Its general properties, resolution, number of examples and classes, are identical to MNIST such that it can serve as a drop-in replacement for the latter. FashionMNIST differentiates itself from MNIST only through the semantic meaning of its classes, i.e. different types of clothing, and the fact that it is completely balanced.

SVHN The SVHN dataset proposed by Netzer et al. in [31] contains colored images of digits which are obtained from house numbers in Google Street View images. These digits are centered and cropped to a resolution of 32×32 pixel. In contrast to MNIST, there can be neighboring digits next to the centered one which make SVHN much more challenging than MNIST. The number of examples per class in SVHN is not exactly but close to being balanced.

CIFAR10 The CIFAR10 dataset introduced by Krizhevsky in [27] features colored images of natural objects with a resolution of 32×32 pixel grouped into 50000 training and 10000 test images. Its 10 classes are in general harder to discriminate than those of the previously introduced datasets. This, in combination with the still small size of the dataset and images, makes it a more suitable choice for evaluating and comparing modern algorithms and methods. The number of examples per class in both, the training and test set, are again perfectly balanced.

CIFAR100 In addition to CIFAR10, Krizhevsky also introduced CIFAR100 in [27]. This dataset has the same number of training and test examples as CIFAR10 with the same resolution but 100 classes that are grouped in 20 superclasses. Given the greater number of classes in this dataset while simultaneously featuring similar object categories as CIFAR10, it can be considered a more difficult dataset. Despite their similar names, the classes in CIFAR100 and CIFAR10 are mutually exclusive as pointed out by Krizhevsky in [27, Chapter 3].

ImageNet The final and largest dataset is the ImageNet introduced by Russakovsky et al. for the "ImageNet Large Scale Visual Recognition Challenge" in [43]. It features 1281167 training, 50000 validation and 100000 test color images with varying resolutions showing natural objects from 1000 classes. Due to it being used in a competition, the test dataset is not directly accessible to the general public. Similar to most of the other presented datasets it is also almost perfectly balanced. Given its size and the large number of classes compared to all other datasets presented above, the ImageNet dataset can be considered as the most challenging but also most realistic dataset used in this thesis.

3.7. Neural Architectures

Similar to the datasets being used in the continual learning literature, the type of neural architecture used in these depends on the problem type. For supervised image classification the classic ResNet architecture [44] is used in many recent publications [56, 58, 96, 107, 114, 129]. ResNets can be scaled by adjusting the number of layers and implementations for them are widely available for common deep learning frameworks. These properties make ResNets a suitable architecture for evaluating continual learning methods despite their age. For experiments on ImageNet or a subset of its classes, ResNet18 is typically used while on smaller datasets, e.g. CIFAR10/100, ResNet32 is used. The input resolution for ResNet18 is chosen as 224×224 pixel while for ResNet32 it is 32×32 pixel for CIFAR10/100 and 28×28 pixel for MNIST and FashionMNIST. The structure of both variants is detailed in Table 3.4 and Table 3.6. For experiments on CIFAR10 in the online continual learning setting, a reduced version of ResNet18 with less filters in each ResBlock, a smaller filter size and a stride of one in the first convolution is used. Its details are shown in Table 3.5.

The operations used in these are:

- The two-dimensional convolution $Conv(k \times k, f, s)$, where k is the kernel size, f is the number of filters and s is the stride;
- Batch normalization [40] $Bnorm(m)$, where m is the momentum used in the moving average;
- The Rectified Linear Unit (ReLU) activation $ReLU()$;

Table 3.4.: Details on ResNet18.

Layer Name	**Operation**	**Output size**
Conv0	$Conv(7 \times 7, 64, 2)$	112×112
BNorm0	$Bnorm(0.9)$	112×112
ReLU0	$ReLU()$	112×112
ResBlock0	$ResBlock(3 \times 3, 64, 1)$	56×56
ResBlock1	$ResBlock(3 \times 3, 64, 1)$	56×56
ResBlock2	$ResBlock(3 \times 3, 128, 2)$	28×28
ResBlock3	$ResBlock(3 \times 3, 128, 1)$	28×28
ResBlock4	$ResBlock(3 \times 3, 256, 2)$	14×14
ResBlock5	$ResBlock(3 \times 3, 256, 1)$	14×14
ResBlock6	$ResBlock(3 \times 3, 512, 2)$	7×7
ResBlock7	$ResBlock(3 \times 3, 512, 1)$	7×7
Glbl. Avg. Pool	$GlblAvgPool()$	512
Dense	$Dense(n)$	n

- Residual Blocks $ResBlock(k \times k, f, s)$ where k is the kernel size, f is the number of filters and s is the stride applied in the first convolution;

- Global Average Pooling $GlblAvgPool()$

- and the final dense Layer $Dense(n)$ with n neurons.

For a detailed discussion on ResNets and how residual blocks are constructed, the reader is referred to the original publication [44].

For experiments on synthetic data presented in chapter 6 another architecture that is popular in the field of few-shot learning [50, 60] is used. Its structure is much simper than ResNet and features blocks of convolutional layers with ReLU activation and instance normalization [49]. The final layer in a block is an average pooling layer that reduces the feature dimension. Similar to [129] this architecture is referred to as ConvNet in this thesis and its structure is detailed in Table 3.7. The two additional layer types that are not already explained in the previous section are:

- The instance normalization [49] layer $Inorm()$ which does not rely on statistics estimated along the batch dimension but rather normalizes instance-wise;

- and average pooling $AvgPool(k \times k)$ which pools the input by taking the average over a $k \times k$ grid.

Table 3.5.: Details on reduced ResNet18 for online continual learning on CIFAR10.

Layer Name	**Operation**	**Output size**
Conv0	$Conv(3\times3,20,1)$	32×32
BNorm0	$Bnorm(0.9)$	32×32
ReLU0	$ReLU()$	32×32
ResBlock0	$ResBlock(3\times3,20,1)$	32×32
ResBlock1	$ResBlock(3\times3,20,1)$	32×32
ResBlock2	$ResBlock(3\times3,40,2)$	16×16
ResBlock3	$ResBlock(3\times3,40,1)$	16×16
ResBlock4	$ResBlock(3\times3,80,2)$	8×8
ResBlock5	$ResBlock(3\times3,80,1)$	8×8
ResBlock6	$ResBlock(3\times3,160,2)$	4×4
ResBlock7	$ResBlock(3\times3,160,1)$	4×4
Glbl. Avg. Pool	$GlblAvgPool()$	160
Dense	$Dense(n)$	n

Table 3.6.: Details on ResNet32.

Layer Name	**Operation**	**Output size**
Conv0	$Conv(3\times3,64,1)$	32×32
BNorm0	$Bnorm(0.9)$	112×32
ReLU0	$ReLU()$	32×32
ResBlock0	$ResBlock(3\times3,16,1)$	32×32
ResBlock1	$ResBlock(3\times3,16,1)$	32×32
ResBlock2	$ResBlock(3\times3,16,2)$	32×32
ResBlock3	$ResBlock(3\times3,16,1)$	32×32
ResBlock4	$ResBlock(3\times3,16,1)$	32×32
ResBlock5	$ResBlock(3\times3,32,2)$	16×16
ResBlock6	$ResBlock(3\times3,32,1)$	16×16
ResBlock7	$ResBlock(3\times3,32,1)$	16×16
ResBlock8	$ResBlock(3\times3,32,1)$	16×16
ResBlock9	$ResBlock(3\times3,32,1)$	16×16
ResBlock10	$ResBlock(3\times3,64,2)$	8×8
ResBlock11	$ResBlock(3\times3,64,1)$	8×8
ResBlock12	$ResBlock(3\times3,64,1)$	8×8
ResBlock13	$ResBlock(3\times3,64,1)$	8×8
ResBlock14	$ResBlock(3\times3,64,1)$	8×8
Glbl. Avg. Pool	$GlblAvgPool()$	64
Dense	$Dense(n)$	n

Table 3.7.: Details on ConvNet used for generating synthetic data.

Layer Name	**Operation**	**Output size**
Conv0	$Conv(3 \times 3, 128, 1)$	32×32
INorm0	$Inorm()$	32×32
ReLU0	$ReLU()$	32×32
Avg. Pool0	$AvgPool(2 \times 2)$	16×16
Conv1	$Conv(3 \times 3, 128, 1)$	16×16
INorm1	$Inorm()$	16×16
ReLU1	$ReLU()$	16×16
Avg. Pool1	$AvgPool(2 \times 2)$	8×8
Conv2	$Conv(3 \times 3, 128, 1)$	8×8
INorm2	$Inorm()$	8×8
ReLU2	$ReLU()$	8×8
Avg. Pool2	$AvgPool(2 \times 2)$	4×4
Dense	$Dense(n)$	n

These different choices for the normalization and pooling layers are due to the application of ConvNet for generating synthetic data which usually requires taking the gradient with respect to the input of a DNN. Since the output of an average pooling is influenced by all inputs on the $k \times k$ grid, a denser flow of gradients is maintained compared to the common max pooling layer. Additionally, synthetic data is usually generated in very small amounts as it is intended to compress the information in an original training dataset. Reliably estimating batch statistics therefore becomes challenging and leads to the replacement of batch normalization with instance normalization layers. These do not rely on batch statistics and therefore are more suitable for generating synthetic data than common batch normalization layers.

Chapter 4.

Localizing Catastrophic Forgetting in Neural Networks

In this chapter a method for determining which part of a DNN contributes with what extend to a change in loss when it is trained on a continual learning sequence without any measures for preventing catastrophic forgetting is presented. While the work by Wiewel and Yang [94] is the basis for this chapter, we extend their work with experiments on more datasets, a deeper DNN architecture and an improved numerical integration method. Experiments on the commonly used MNIST , FashionMNIST and CIFAR10 datasets are performed using the popular ResNet32 architecture. The results are discussed, compared and validated with related work in the field of continual learning.

4.1. Motivation

In section 3.5 several metrics that are suitable to evaluate the performance and efficiency of a method for continual learning with DNNs are discussed. Except for the efficiency metrics, that measure how much resources are needed in a specific method, all of them quantify how well a given DNN performs on a sequence of continual learning tasks. In the case of supervised image classification considered in this work, the preferred underlying metric from which those are derived is the classification accuracy. Their scalar nature and the fact that they capture the central challenge of continual learning, namely catastrophic forgetting, allows for fast and simple comparisons between competing methods.

But despite their ability to quantify catastrophic forgetting, they provide no insight into which parts of a DNN contribute to it in what extend. Studying catastrophic forgetting and localizing it in a DNN is not only important for a better understanding of the phenomenon itself but can also guide the development of new methods and algorithms for overcoming it. Given the hierarchical structure of a typical DNN used in computer vision tasks, where the first layers learn general and later ones more specific features, different parts might experience catastrophic forgetting to a differing extend. Yosinski et al. point out in [37] that the general features learned in the first layers of a DNN are observed mostly independent of the used loss function and natural image dataset while features found in the last layers are highly dependent on those. Considering the definition of a task in continual learning as discussed in section 2 it can therefore be hypothesized that different layers in a DNN are affected to a differing extend by catastrophic forgetting. More specifically, as the first layers learn features that are more general and potentially transferable between tasks, the effect of catastrophic forgetting on them might be lower than for later layers. The central research question we try to answer in this chapter is therefore:

Which parts of a DNN contribute with what extend to a change in loss when it is trained on a continual learning sequence without any measures for preventing catastrophic forgetting?

In this chapter, we introduce a method for estimating the contribution of individual parameters to the commonly observed phenomenon of catastrophic forgetting in continual learning with DNNs. Our approach is inspired by the parameter space view of SI, discussed in section 3.1, and is based on tracking the contribution of individual parameters to the change in loss when changing between two tasks. This allows for studying and localizing catastrophic forgetting at different resolutions by aggregating the individual contributions of parameters over layers or even bigger substructures of a DNN. We use this method to empirically study catastrophic forgetting in supervised image classification under the three different scenarios discussed in section 2.2 on select datasets and neural architectures.

4.2. Method

This section is structured in three parts. First, the main idea of how parameter specific contributions can be estimated and an approach for validating the found parameter specific contributions are described. This is followed by a section covering how this basic idea can be applied to continual learning in order to analyze catastrophic forgetting. Finally, a practical implementation using numerical integration is presented.

4.2.1. Training DNNs: Moving Through Parameter Space

According to Definition 2, solving a task $\mathcal{T}$ in the context of continual learning is done by finding the optimal parameters $\boldsymbol{\theta}^\star$ that minimize a suitable loss function $\mathcal{L}(\mathcal{T}, \boldsymbol{\theta}) \in \mathbb{R}$. For this, the DNN is first initialized with random weights which are then repeatedly updated using SGD or a variant of it for N iterations in order to minimize the given loss. During the i-th step of this process the current parameter vector is given by $\boldsymbol{\theta}_i$ and the corresponding loss can be evaluated by simply determining $\mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_i)$. Given these definitions, the following introduces two different levels of abstraction with which the training dynamics can be described. For this, the process of training is viewed as moving along a trajectory C through parameter space in discrete steps. Its starting point is therefore given by the randomly initialized parameter vector $\boldsymbol{\theta}_1$ and its endpoint by the optimal weights $\boldsymbol{\theta}^\star = \boldsymbol{\theta}_N$.

Starting with the higher level of abstraction, only the scalar loss $\mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_i)$ is evaluated at every time step. The overall change in loss is therefore given by

$$\Delta\mathcal{L} = \mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_N) - \mathcal{L}(\mathcal{T}, \boldsymbol{\theta}_1), \tag{4.1}$$

i.e. the difference between $\mathcal{L}$ evaluated at time step N and one. This commonly used way of describing the training dynamics is simple, fast and easy to visualize as only the loss, which describes the model performance in one scalar, needs to be evaluated. But since it summarizes the performance of a DNN in a single scalar, the contributions of individual parameters to the decrease in loss are indistinguishable. In other words, the change in loss is quantified by summarizing the contributions of all parameters into a single scalar.

More insight can be gained if the training process is viewed as moving along the continuous trajectory C through the parameter space. Under this perspective, an infinitesimal change in loss is given by

$$\mathrm{d}\mathcal{L} = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}, \boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \tag{4.2}$$

where $\nabla_{\boldsymbol{\theta}}$ is the Nabla operator and $\mathrm{d}\boldsymbol{\theta}$ is an infinitesimal step in the parameter vector. Using this, the overall change in loss is equivalently given as

$$\Delta\mathcal{L} = \int_C \mathrm{d}\mathcal{L} = \int_C \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T},\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \tag{4.3}$$

where the infinitesimal loss changes are integrated along the continuous trajectory C through the parameter space. It is critical to note that although (4.1) and (4.3) are in fact equivalent, the computational complexity for evaluating them differs significantly. While the former only requires the evaluation of $\mathcal{L}(\mathcal{T},\boldsymbol{\theta})$ at two different points, the latter requires integrating the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T},\boldsymbol{\theta})$ along a potentially long continuous trajectory C. But crucial to our approach (4.3) can be decomposed into

$$\Delta\mathcal{L} = \int_C \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T},\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} = \sum_j \int_{C^j} \nabla_{\theta_j}\mathcal{L}(\mathcal{T},\boldsymbol{\theta})\mathrm{d}\theta_j = \sum_j [\mathbf{c}]_j\,, \tag{4.4}$$

where the vector $\mathbf{c}$ contains the contributions for each parameter in $\boldsymbol{\theta}$ and C^j describes the integration bounds for the individual parameter θ_j. Based on this observation, we can evaluate the path integral while keeping the parameter specific contributions separate.

4.2.2. Application to Continual Learning

The previous paragraph considered two different levels of abstraction for analyzing the training process of a DNN. As shown above, using the path integral along the training trajectory C through parameter space allows for determining individual contributions of parameters to a change in loss. This can now be used to study the effects of catastrophic forgetting on a sequence of continual learning tasks. For this, we consider a sequence of two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$.

First, the DNN is trained on task $\mathcal{T}_1$ to convergence after which the optimal parameters $\boldsymbol{\theta}_1^\star$ are obtained. This is the starting point of our approach as the DNN is now capable of solving the first task, characterized by a low loss $\mathcal{L}(\mathcal{T}_1,\boldsymbol{\theta}_1^\star)$. All changes in loss up to this point are of minor interest as they only account for learning the first task starting from randomly initialized weights. Tracking the changes along its corresponding trajectory C_1 is done by only evaluating the loss function on the test dataset of the first task.

Next, the DNN is trained only on the data of task $\mathcal{T}_2$ without any measure to prevent catastrophic forgetting until convergence and the optimal parameters $\boldsymbol{\theta}_2^\star$ are obtained. During this process, the change in loss on task $\mathcal{T}_1$ is of special interest as any increase in it quantifies the amount of catastrophic forgetting experienced by the DNN. We therefore evaluate the path integral from (4.4) on the test dataset of task $\mathcal{T}_1$ while moving along the training trajectory C_2 of task $\mathcal{T}_2$, i.e. we evaluate $\int_{C_2} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}$. The parameter specific contributions can be used for further analysis of which parameters or layers contributed in what extend to a change in loss on the first task.

4.2.3. Numerical Integration: An Approximate Solution

Determining the path integral analytically is infeasible. The reason for this is twofold. During training we only observe discrete points along C_2 and not a continuously changing parameter vector $\boldsymbol{\theta}$. But even if the continuous path C_2 would be known, evaluating the gradient along it is impractical since an analytical expression for the gradient is unavailable. We therefore resort to numerical integration methods in order to approximate the path integral in a computationally efficient way.

During training of a DNN discrete points on the path C_2, i.e. the parameter vector at the i-th iteration when training on task $\mathcal{T}_2$, are observed. The exact path through the parameter space between two consecutive iterations, however, is unknown. But considering that the learning rates and therefore step sizes are usually small when training a DNN, it is reasonable to assume that the parameter vector moves along a direct path between iterations. The path C_2 is therefore approximated by straight line segments C_2^i connecting $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i-1}$.

Definition 7: Simpson's Rule

Named after the British mathematician Thomas Simpson (1710-1761), Simpson's rule [24] is a numerical approximation of a definite integral. It is given by

$$\int_{x_0}^{x_1} f(x) \mathrm{d}x \approx h \frac{f(x_0) + 4f(x_0 + h/2) + f(x_1)}{6}, \tag{4.5}$$

where $h = x_1 - x_0$. It is a third order approximation, i.e. it is exact for polynomials up to and including order three, and requires three evaluations of the function f. Its application to rotational solids is equivalent to the "Fassregel" already published by Keppler in 1615.

Since each contribution vector $\mathbf{c}^i$ evaluated along the line segment C_2^i corresponds to a definite integral, well known quadrature methods can be used to approximate it component wise. For contributions during the i-th training iteration, Simpson's rule yields

$$\begin{aligned}\mathbf{c}^i &= \left[\int_{C_2^i} \nabla_{\theta_j}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta})\mathrm{d}\theta_j\right]_j \\ &\approx \mathbf{h} \odot \frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_i) + 4\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_{i-1} + \mathbf{h}/2) + \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_{i-1})}{6},\end{aligned} \tag{4.6}$$

where $\mathbf{h} = \boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}$ is the update step performed in this iteration and the Hadamard product $\odot$ is used to perform an element-wise multiplication. The total contribution vector $\mathbf{c}$ from (4.4) is simply the accumulation of these for each training iteration, i.e. $\mathbf{c} = \sum_i \mathbf{c}^i$. With this, our method for localizing catastrophic forgetting is summarized using pseudo code as shown in Algorithm 1.

Algorithm 1: Pseudo code for localizing catastrophic forgetting using SGD

```
Input: Number of iterations $N_1, N_2$; Learning rate $\gamma$, Tasks $\mathcal{T}_1, \mathcal{T}_2$, Loss function $\mathcal{L}$
1  Randomly initialize DNN weights $\boldsymbol{\theta}_0$;
2  for $1 \leq i \leq N_1$ do
3      Update weights: $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} - \gamma\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_{i-1})$;
4  end
5  Initialize contribution vector: $\mathbf{c}_{N_1} = \mathbf{0}$;
6  for $N_1 \leq i \leq N_2 + N_1$ do
7      Update weights: $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} - \gamma\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_2, \boldsymbol{\theta}_{i-1})$;
8      Determine step: $\mathbf{h} = \boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}$;
9      Update contribution:
         $\mathbf{c}_{i+1} \leftarrow \mathbf{c}_i + \mathbf{h} \odot \frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_i) + 4\nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_{i-1} + \mathbf{h}/2) + \nabla_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_{i-1})}{6}$;
10 end
Output: Contribution vector $\mathbf{c}_{N_1+N_2}$
```

4.3. Experiments

In this section we use our method to visualize catastrophic forgetting for all three continual learning scenarios discussed in section 2.2. For this, we construct corresponding continual learning sequences using some of the datasets discussed in section 3.6 and train ResNet32 on them without any measure against catastrophic forgetting. Throughout these sequences we record scalar performance metrics and the individual contributions of parameters using our method. We visualize both types of metrics and verify the estimated contributions by comparing the changes in loss according to (4.1) and (4.4).

4.3.1. Hyperparameters

For all experiments we first train ResNet32, discussed in section 3.7, for 4000 iterations on the first task $\mathcal{T}_1$ using the SGD optimizer with a learning rate of 0.1, a batch size of 128 and a momentum of 0.9. We then train on the second task $\mathcal{T}_2$ for 10 iterations with the same batch size and optimizer parameters while tracking the changes in loss on the test dataset of task $\mathcal{T}_1$ according to section 4.2. Although training and tracking for only 10 iterations on the second task is not enough to reach convergence, it is sufficient to show the effects of catastrophic forgetting for the different scenarios. All results are reported as the average over 10 independent runs with different random initializations.

4.3.2. ICL Experiment Setup

Constructing the ICL sequence is done by splitting a dataset for supervised image classification into two parts. Although it is not strictly necessary, each part contains the same number of classes in order to achieve the same class balance in every task. For this experiment to be comparable with the following ones, we chose the same datasets for all of them. These are MNIST, FashionMNIST and CIFAR10. They are split into task $\mathcal{T}_1$ containing the first half of all classes while task $\mathcal{T}_2$ contains the remaining ones. Again the input space remains unchanged, i.e. $\mathcal{X}_1 = \mathcal{X}_2$, while the output spaces are different, i.e. $\mathcal{Y}_1 \subset \mathcal{Y}_2$. But unlike in the ITL sequence since $\mathcal{Y}_1$ is a subset of $\mathcal{Y}_2$ the same output layer can be used for both tasks. Note that this also implies that during training on task $\mathcal{T}_1$ only the first five output neurons are used to predict and compute gradients. During the second task all ten output neurons are used for training. Tracking the change in loss on task $\mathcal{T}_1$ according to algorithm 1, however, is again performed using only the first five output neurons. In addition to this the marginals over both, the input and output spaces, change between tasks, i.e. $p_{X,1} \neq p_{X,2}$ and $p_{Y,1} \neq p_{Y,2}$.

4.3.3. IDL Experiment Setup

The IDL sequence is constructed as follows: Task $\mathcal{T}_1$ consists of the original dataset with all its classes. The second task $\mathcal{T}_2$ is created by flipping each image in the dataset upside down while keeping its label unchanged. In this case, the input spaces $\mathcal{X}_1 = \mathcal{X}_2$ are again identical while the marginal distributions over it change, $p_{X,1} \neq p_{X,2}$. But in contrast to the ITL sequence described later, flipping the images does not change the label and therefore the label spaces $\mathcal{Y}_1 = \mathcal{Y}_2$ and the conditional distributions $p_{Y,1|X,1} = p_{Y,2|X,2}$ are identical. This implies that in the implementation of this experiment only one output layer can be used for both tasks.

4.3.4. ITL Experiment Setup

For constructing an ITL sequence, two supervised classification tasks $\mathcal{T}_1$ and $\mathcal{T}_2$ are constructed. Task $\mathcal{T}_1$ consists of classifying the first half of all classes from a dataset and task $\mathcal{T}_2$ requires the classification of all remaining classes. Using the notation introduced in section 2.2, the input spaces $\mathcal{X}_1 = \mathcal{X}_2$ are identical for both tasks. But since both tasks contain different classes, the marginal distributions over the input spaces are not identical, $p_{X,1} \neq p_{X,2}$. Although both tasks are supervised image classification tasks, the output spaces are treated separately, i.e. $\mathcal{Y}_1 \neq \mathcal{Y}_2$, which requires a separate output layer for each task. Therefore it is important to use the correct output layer when evaluating the gradients in algorithm 1.

4.3.5. Scenario-Specific Challenges

The three scenarios described above differ significantly in their definition and therefore also pose different challenges regarding continual learning. The ICL scenario is probably the most intuitive one for an image classification problem as it simply adds more classes to be classified with each new task. It is, however, also the most challenging one. The reason for this is twofold. First, the final layer of a DNN, i.e. the classification layer, needs to be expanded for accommodating these new classes. After this expansion the layer has weights that were trained on previous tasks while those of the newly added output neurons are just randomly initialized. Training this layer to not only distinguish between the newly added classes but also the previously learned ones is challenging, especially since access to data of previous tasks is severely restricted. Second, newly added classes might differ significantly in their class-defining visual features. This might require changes to layers that are associated with feature extraction. For differences in high-level features, deeper layers close to the final layer need to be adapted. For more general low-level features the first layers of a DNN need to be changed. Although such changes are generally simple to accomplish by retraining with all data, they can lead to severe catastrophic forgetting in continual learning. When localizing changes in loss using the previously introduced method, high contributions of the very first and/or later layers including the final one to catastrophic forgetting are expected.

The IDL scenario is less challenging than ICL since it does not require to expand the last layer of a DNN. The challenges associated with this expansion therefore do not apply to IDL. But the visual features both high- and low-level might still change significantly when learning a new domain. Forgetting might therefore still occur but usually to a lesser extend. If the class-defining features in two different domains are similar enough only small adaptations might be needed to learn the new domain and therefore only limited forgetting might occur. When localizing a change in loss significant contributions are to be expected mostly for the feature extracting layers, i.e. the very first ones for low- and the last layers for high-level features. In contrast to ICL, much less loss change contribution is expected for the final layer of a DNN.

The last scenario ITL is the least challenging, at least in the way it is usually constructed in the literature and also this thesis. The reason for this is that although in general ITL allows for task to be completely different, e.g. image classification in the first and semantic segmentation in the second task, it is often constructed from tasks of the same type, i.e. here image classification. This in combination with the separate final layer for each task means that only limited changes are required to the feature extracting part of a DNN that is actually shared between tasks. If the class-defining features of the classes from these tasks are also similar, there is only very limited potential for catastrophic forgetting to occur. During localization of changes in loss, only the feature extracting layers might show some contribution while the classification layer contributes nothing. The reason for this is the use of a separate final layer for every task.

4.4. Results

The estimated changes in loss for each of the experiments described above are shown in Figure 4.1. For a clear visualization the contributions of all parameters in a layer according to the structure of ResNet32 described in Table 3.6 have been aggregated by summing their contributions. Each row shows the results for a particular dataset. The top row shows those of MNIST, the second row those of FashionMNIST and the last row show the results for

CIFAR10. The left column shows the total contribution of each layer while the right column shows it normalized by the number of parameters in each layer. Results for the experiments using different continual learning scenarios are color coded as red, blue and purple for ICL, IDL and ITL.

4.4.1. Observations

Comparing the results shown in Figure 4.1 it is obvious that the contribution to a changing loss is different for the three scenarios. But for each scenario some similarities can be observed between the datasets. This section presents the qualitative observations provided by the localization of changes in loss. Quantitative results are presented and discussed in section 4.4.2.

For ICL significant contributions of later layers and the final layer are observed for every dataset. This aligns with the expectation formulated in section 4.3.5. While for MNIST and FashionMNIST all layers on average show a positive contribution, i.e. an increase in loss, ResBlock14 and the last Dense layer actually show a significant negative contribution, i.e. a decrease in loss for CIFAR10. Although this is somewhat surprising, the overall change in loss is still positive for CIFAR10 as reported in Table 4.2 and 4.3. Hence there is still catastrophic forgetting as can be observed from the scalar metrics discussed in section 4.4.2. Another difference for ICL between the datasets is an increasingly positive contribution of the first layer. This also aligns with the previously formulated expectations since the low-level class-defining features in MNIST are rather simple. Digits are simply combinations of straight and curved lines. Classes of FashionMNIST on the other hand also have texture and more complex shapes. The classes of CIFAR10 posses arguably even more sophisticated low-level features since they show natural objects. Consequently the contribution to a change in loss of the first layer increases from MNSIT, to FashionMNIST and is highest for CIFAR10. When considering the contribution per parameter the first and last layer even show larger contribution than the hidden layers for FashionMNIST and CIFAR10. For MNIST the first layer shows a similar contribution per parameter as the hidden layers while again the last layer shows a much larger one.

The contributions for IDL on the other hand are more similar for the tested datasets. In this case, most of the overall contribution is attributable to the later feature extracting layers while the last layer shows a smaller even negative contribution. This again agrees with the expectations formulated in section 4.3.5. Similar to ICL the contribution of first layers again increases from MNIST over FashionMNIST up to CIFAR10. This can best be observed by using the contribution per parameter[1]. A possible explanation for this is again the increasing complexity of class-defining visual features in these datasets.

In the case of ITL there are only few layers which show noticeable contributions. These are mostly the later feature extracting layers while the last Dense layer as expected has always a contribution of zero. The observable contributions are mostly small and strictly negative for MNIST and FashionMNIST. They also closely match the difference in scalar loss as shown in Table 4.3. Hence these results do not seem plausible. A possible reason for this might be that the changes in loss are so small that errors in the numerical integration have a significant impact. In contrast to this, a positive contribution that is close to the difference in loss determined using scalar metrics is observed for ResBlock14 on CIFAR10. A possible reason for this could be that the separate final layer that is used for every task in ITL is not sufficient to account for all required changes to learn the new classes. Hence adaptations in the preceding layers are required. Since these are shared between all tasks, changes to them might cause catastrophic forgetting.

4.4.2. Scalar Metrics

In addition to the estimated contributions to the change in loss on task $\mathcal{T}_1$, we also evaluate the loss and accuracy on both tasks throughout the continual learning sequences. The loss and accuracy are shown in Table 4.2 where $\mathcal{L}(\mathcal{T}_i, \boldsymbol{\theta}_j^\star)$ is the loss evaluated on task $\mathcal{T}_i$ using the optimal weights $\boldsymbol{\theta}_j^\star$ found by training on task $\mathcal{T}_j$ and $\mathrm{A}(\mathcal{T}_i, \boldsymbol{\theta}_j^\star)$ is the corresponding accuracy.

Since no measure for protecting the DNN against catastrophic forgetting is used, a decrease in performance on task $\mathcal{T}_1$ is to be expected after task $\mathcal{T}_2$ is learned. The expected changes in accuracy A are summarized in Table 4.1. After the DNN is trained on the first task (left column) it achieves a high accuracy on this task. Since it was not trained on the second task at this point, it will achieve only low performance on it. After the DNN is trained on the second task (right column) a high accuracy is to be expected on this task. But since no measures against catastrophic forgetting were used, its performance on task $\mathcal{T}_1$ will deteriorate due to forgetting. For the loss an opposite trend can be observed since a low loss corresponds to a high accuracy and vice versa.

[1]Note the different scales for every dataset.

Table 4.1.: Expected accuracy changes.

Accuracy A	$\boldsymbol{\theta}_1^\star$	$\boldsymbol{\theta}_2^\star$
$\mathcal{T}_1$	High	Forgetting
$\mathcal{T}_2$	Low	High

Since catastrophic forgetting effects the performance of ResNet32 on task $\mathcal{T}_1$, we focus our discussion of the scalar metrics on the first task. Similar to the results discussed in the previous section, there is a significant difference in the scalar metrics between different continual learning scenarios as well. Across all dataset and scenario combinations the loss $\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_2^\star)$ is significantly higher than $\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_1^\star)$. But comparing the scenarios it is obvious that this increase is smaller for ITL when compared with the other two scenarios. The reason for this is that during IDL and ICL all weights of the DNN are shared between tasks, while in ITL a separate output layer is used for every task. Most of the changes required during ITL for learning a new task can therefore be accounted for by this last layer. The preceding layers remain mostly unchanged. This is also reflected in Figure 4.1 where only ResBlock14 showed a significant contribution to the changing loss during ITL on CIFAR10. Interestingly, the increase in loss is slightly higher for IDL than ICL on MNIST and CIFAR10. This is in contrast to FashionMNIST where ICL shows a significantly higher increase in loss.

The ordering in difficulty from ICL being the most challenging over the less demanding IDL to the least difficult ITL scenario as discussed in section 4.3.5 is based on the performance of a DNN. This is measured using not the loss but accuracy as a metric. The ordering is therefore reflected in the lower half of Table 4.2. Comparing $\mathrm{A}(\mathcal{T}_1, \boldsymbol{\theta}_1^\star)$ to $\mathrm{A}(\mathcal{T}_1, \boldsymbol{\theta}_2^\star)$ the largest decrease is observed for ICL, followed by IDL and finally ITL. Although the changes in loss for ICL are slightly smaller than those for IDL on MNIST and CIFAR10, the accuracy shows a much larger decrease for ICL than IDL.

4.4.3. Result Verification

Conclusions can only be derived from the estimated contributions shown in Figure 4.1 if these closely match the actual changes in loss. For this, the change in loss on task $\mathcal{T}_1$ are determined directly by (4.1) and compared with (4.4). The results for different continual learning scenarios and datasets are shown in Table 4.3. While the actual change in loss $\Delta\mathcal{L}$ is close to the estimated $\Delta\mathcal{L}_{est}$ for ICL and IDL on all datasets, there is a noticeable difference between both for ITL. On MNIST and FashionMNIST our algorithm seems to severely underestimate the change in loss and on CIFAR10 it seems to slightly overestimate it. Although the cause for this behavior remains unclear, one possible explanation could be an approximation error due to the numerical integration using Simpson's rule. But since the changes in loss are small and catastrophic forgetting seems to be much less severe for ITL when compared to the other scenarios, this discrepancy can be tolerated. Overall we are interested in studying a significant change in loss due to catastrophic forgetting which seems to only be present in the ICL and IDL scenarios.

4.5. Related Work

Our method is inspired by the parameter space view on the training process of DNNs as proposed by Zenke, Poole, and Ganguli in conjunction with their method SI [63]. The same perspective is also adapted by Chaudhry et al. for their method Riemannian Walk (RWALK) proposed in [66]. But while the contribution of parameters to a change in loss for SI and RWALK is determined on the same task $\mathcal{T}_k$ as the DNN is trained on, our method tracks the change in loss on the previously learned task $\mathcal{T}_{k-1}$. Similar to our work, Goodfellow et al. study the phenomenon of catastrophic forgetting in [34]. But their empirical study considers only scalar metrics and studies the effects of different activation functions and Dropout [36] on the continual learning performance of a DNN. Overall our findings agree with Yosinski et al. [37] in a sense that the first layers of a DNN seem to learn more general and therefore transferable features when trained on natural image datasets like those used in this thesis. We base this claim on the results in Figure 4.1 which show that, at least for the used datasets and the ResNet32 architecture, the later layers contribute more to a change in loss. This increased contribution can partly be explained by a greater number of parameters in these layers but even when normalizing by the number of parameters the contribution per parameter is increased for later layers. Although our experiments are limited to only one neural architecture and a small number of datasets, Wu et al. [95] also conclude that the last layer in a DNN gets biased towards the most recently learned classes in an ICL sequence which agrees with our findings as well.

4.6. Conclusion

By interpreting the process of training a DNN as moving along a trajectory in parameter space, similar to SI, we derived Algorithm 1 in section 4.2 for estimating the contribution of individual parameters to a change in loss. For this, Simpsons rule is used to numerically approximate the path integral along the training trajectory while keeping the contributions of all parameters separate. The resulting contribution vector **c** can then be aggregated over layers and even bigger structures, e.g. ResBlocks, that contain multiple layers in order to visualize the contributions of these. Using this algorithm we try to answer the research question from the motivation:

Which parts of a DNN contribute with what extend to a change in loss when it is trained on a continual learning sequence without any measures for preventing catastrophic forgetting?

For this, we first train a ResNet32 on task $\mathcal{T}_1$ until convergence and then train it on a different task $\mathcal{T}_2$ while simultaneously tracking the change in loss on task $\mathcal{T}_1$ as described in section 4.3. The resulting estimated contributions are shown for each individual layer in Figure 4.1 and discussed in section 4.4. The exact contributions differ for different datasets and scenarios but there are patterns that can be recognized in how the contributions are distributed across ResNet32. For ICL mostly later layers and the final dense layer contribute to a change in loss. The first layer also has a contribution that increases from MNIST over FashionMNIST to CIFAR10. Interestingly, the contributions of ResBlock14 and the final dense are negative on CIFAR10. But the overall change in loss remains positive and significant catastrophic forgetting is observed. IDL shows similar results with the dense layer and ResBlock14 contributing significantly less than the layers proceeding them. For ITL only ResBlock14 shows the highest contribution while layers preceding it show much less and the final dense layer zero contribution.

Comparing the scalar metrics reveals that although the increase in loss is larger for IDL than ICL on MNIST and CIFAR10, the drop in accuracy is largest for ICL, followed by IDL and smallest for ITL on all tested datasets. This aligns with the expected order of difficulty for the three continual learning scenarios. The result of ITL being the least challenging scenario is somewhat counter intuitive but is due to the reason that separate output and/or input layers are used for different tasks. This is not the case for ICL and IDL where the same input and output layers are used for different tasks. This makes a solo comparison of the loss change between ITL and ICL/IDL not fair.

In the literature and this thesis the problem types, i.e. image classification, are identical for every task. This is in agreement with the general definition of ITL but at the same time also quiet restrictive. The definition allows for each task to have arbitrary problem types. A much more challenging ITL sequence might have one task requiring image classification, another one semantic segmentation and a final one image synthesis. Although such sequences might be more challenging and realistic they are also much more difficult to construct and will complicate comparisons with other methods which have never addressed this mixed task setup. Hence this thesis follows the literature and only constructs ITL sequences using image classification tasks.

Table 4.2.: Loss and accuracy for ResNet32 trained on different datasets and continual learning scenarios.

Loss	$\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_1^\star)$	$\mathcal{L}(\mathcal{T}_1, \boldsymbol{\theta}_2^\star)$	$\mathcal{L}(\mathcal{T}_2, \boldsymbol{\theta}_1^\star)$	$\mathcal{L}(\mathcal{T}_2, \boldsymbol{\theta}_2^\star)$
Scenario		**MNIST**		
ICL	0.0279±0.066	1.49±0.48	6.81±0.92	0.906±0.18
IDL	0.0163±0.0016	1.62±0.53	6.32±0.31	0.783±0.25
ITL	0.00563±0.0017	0.135±0.22	4.59±0.64	0.409±0.092
Scenario		**FashionMNIST**		
ICL	0.281±0.018	3.03±1.3	9.33±0.74	1.69±0.33
IDL	0.305±0.025	0.785±0.082	5.25±0.41	0.846±0.12
ITL	0.273±0.018	0.743±0.5	10.0±0.98	1.96±1.0
Scenario		**CIFAR10**		
ICL	0.42±0.039	1.75±0.11	7.56±0.15	1.53±0.062
IDL	0.659±0.052	2.22±0.16	3.31±0.15	1.99±0.097
ITL	0.445±0.016	1.32±0.33	10.6±0.31	0.594±0.065
Accuracy	A$(\mathcal{T}_1, \boldsymbol{\theta}_1^\star)$	A$(\mathcal{T}_1, \boldsymbol{\theta}_2^\star)$	A$(\mathcal{T}_2, \boldsymbol{\theta}_1^\star)$	A$(\mathcal{T}_2, \boldsymbol{\theta}_2^\star)$
Scenario		**MNIST**		
ICL	0.993±0.014	0.486±0.12	0.000432±0.00087	0.67±0.11
IDL	0.995±0.00044	0.597±0.046	0.436±0.013	0.786±0.054
ITL	0.998±0.00067	0.968±0.044	0.354±0.025	0.861±0.039
Scenario		**FashionMNIST**		
ICL	0.948±0.0019	0.285±0.12	0.0±0.0	0.355±0.1
IDL	0.929±0.0024	0.764±0.023	0.276±0.01	0.754±0.029
ITL	0.948±0.0034	0.88±0.06	0.213±0.017	0.515±0.11
Scenario		**CIFAR10**		
ICL	0.896±0.0073	0.255±0.053	0.0±0.0	0.335±0.05
IDL	0.829±0.011	0.417±0.026	0.353±0.011	0.463±0.026
ITL	0.895±0.0039	0.769±0.035	0.049±0.0034	0.792±0.023

Table 4.3.: Loss on task $\mathcal{T}_1$ for ResNet32. $\Delta\mathcal{L}$ is the change in loss evaluated by (4.1) while $\Delta\mathcal{L}_{est}$ is the change in loss determined by summing up all parameter specific contributions in (4.4).

Change	$\Delta\mathcal{L}/\Delta\mathcal{L}_{est}$	$\Delta\mathcal{L}/\Delta\mathcal{L}_{est}$	$\Delta\mathcal{L}/\Delta\mathcal{L}_{est}$
Scenario	**MNIST**	**FashionMNIST**	**CIFAR10**
ICL	$1.461/1.470 \approx 0.9939$	$2.753/2.736 \approx 1.0062$	$1.333/1.442 \approx 0.9244$
IDL	$1.604/1.601 \approx 1.0019$	$0.480/0.492 \approx 0.9756$	$1.563/1.835 \approx 0.8518$
ITL	$0.130/-0.437 \approx -0.2975$	$0.469/-0.059 \approx -7.9491$	$0.880/0.995 \approx 0.8844$

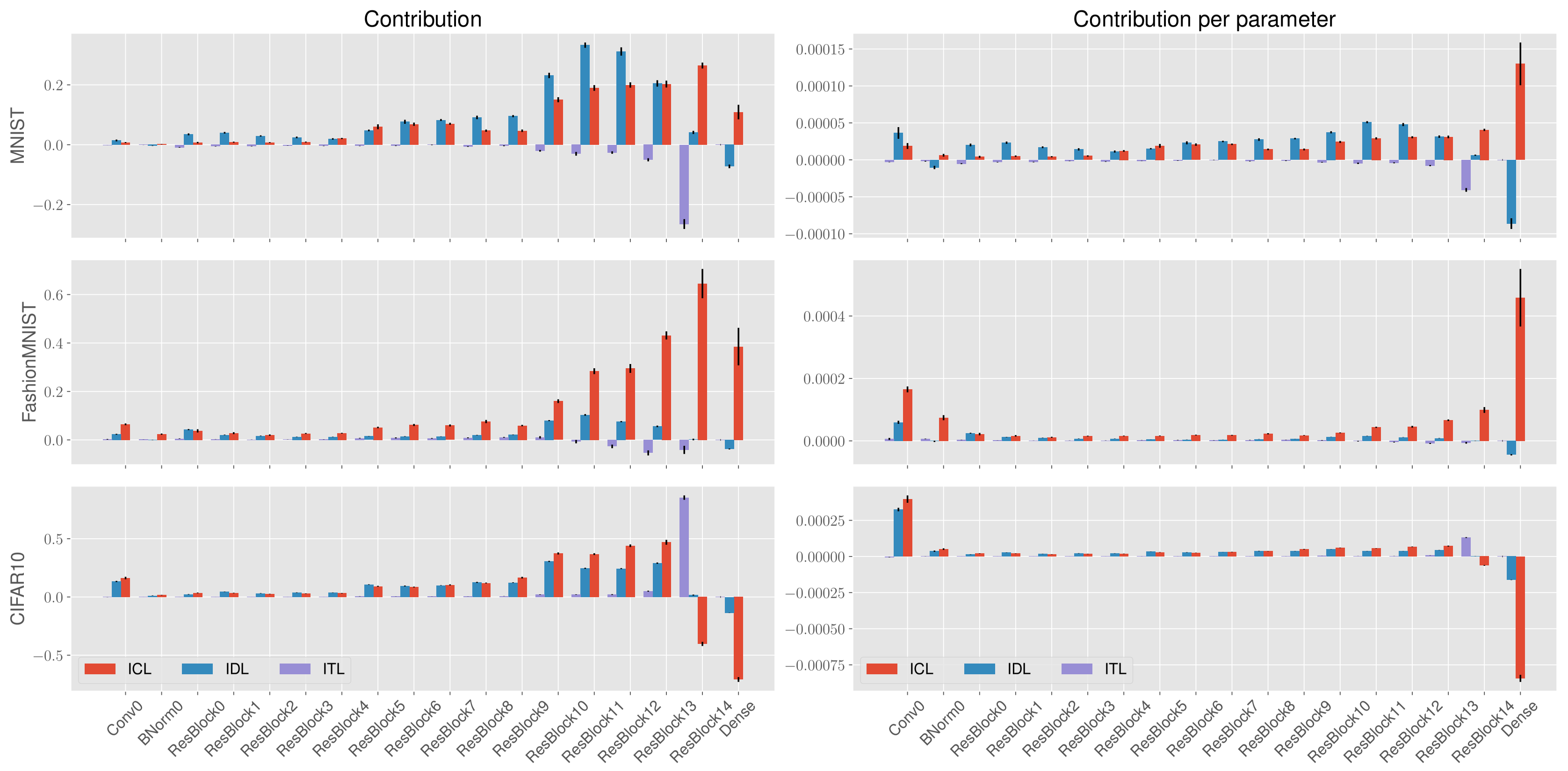


Figure 4.1.: Contributions to change in loss on task $\mathcal{T}_1$ estimated using algorithm 1. From top to bottom each row shows the results for MNIST, FashionMNIST and CIFAR10. The left column shows the total contribution of each layer in ResNet32 while the right column shows the contribution normalized by the number of parameters in each layer. The results for ICL experiments are shown in red, those for IDL in blue and ITL are shown in purple. Each colored bar represents a result averaged over 10 independent runs and the corresponding standard deviation divided by ten is shown as a black bar.

Chapter 5.

Online Continual Learning and Buffer Management

This chapter presents an algorithm first proposed by Wiewel and Yang in [126] for managing a small rehearsal memory in the online continual learning setting where data becomes available in a stream of individual examples or small batches at a time. Given these constraints, online continual learning more closely resembles the environment power and compute restricted autonomous systems could encounter in the real world. It is therefore of great interest as it would allow such systems to accumulate knowledge of potentially long time periods without forgetting. The method itself is motivated and derived by adopting an information theoretic perspective and applying the well known principle of entropy maximization. Experimental results for different online continual learning problems on commonly used benchmark datasets are presented while the proposed algorithm is compared with related methods.

5.1. Motivation

Although continual learning problems share a common definition, i.e. Definition 3, there exist differences that lead to significantly different challenges that need to be overcome. The first and most obvious one is the underlying problem type of each task in a continual learning sequence. It is common in the literature that all tasks originate from the same type of machine learning problem, i.e. supervised image classification. But in general this is not required and each task can be of a different type and possibly feature a differing dataset. In addition to this, there can be different continual learning scenarios as discussed in section 2.2 for supervised image classification. These can lead to different continual learning sequences even if all tasks are of the same type and use the same dataset. Finally, the way data is made available to a method during continual learning is another distinctive feature.

Although the properties of online continual learning as presented in section 2.3 might seem overly restrictive at first glance, they represent the environment that an autonomous learning system might encounter in the real world more closely than its offline counterpart. The autonomous system might be able to observe its surrounding and learn from it through interacting with it. But due to limited storage space or other reasons it might be prevented from accumulating a large dataset for offline training and therefore needs to be capable of online learning. In this case, it is also impossible to access data at random. Studying and developing methods for online continual learning with DNNs is therefore an important part of the research body on continual learning. The potential application in autonomous systems with often limited computational and storage resources in addition to the online aspect requires powerful but also efficient methods. Examples of online continual learning for autonomous systems include Simultaneous Localization and Mapping (SLAM) [134], Visual-Inertial Odometry (VIO) [135][1] and depth estimation [136][1].

The method Gradient based Sample Selection (GSS) proposed by Aljundi et al. in [83], for example, uses a small rehearsal buffer in combination with an algorithm that uses gradient information in order to decide which examples to store and keep in it for online continual learning with DNNs. It is motivated by a constraint reduction of an optimization problem that ensures a non-increasing loss on all training examples stored in the rehearsal buffer. But although GSS achieves strong results, it is computationally intensive and even a faster but inexact greedy alternative of the original algorithm requires the calculation of gradients for many examples from the buffer whenever a new training example is observed. This computational complexity makes it unattractive for power and computationally limited autonomous systems and motivates us to search for a more efficient and potentially even better algorithm. In this chapter, we therefore intend to answer the following research question:

How can the buffer of a rehearsal-based method for continual learning be managed in a simple but effective way to enable online continual learning?

[1]Preprint

An important thing to note is that the focus of this chapter is purely on optimizing the management of a rehearsal buffer, i.e. which data to store and what to replace, not how this stored buffer is used in training. Therefore, only random sampling from the rehearsal buffer and mixing this data is used during training for the method proposed in this chapter. A more complex and improved way of using stored rehearsal data during continual learning is presented in chapter 7 of this thesis.

In this chapter, we adopt an information theoretic perspective and use the well known Shannon entropy [1] in order to derive an algorithm in section 5.2. It is capable to decide which examples should be stored and which ones replaced by them in a rehearsal buffer. The experimental setups and results for comparing our method to related work are presented in section 5.3. Finally the chapter is closed with a conclusion in section 5.4.

5.2. Method

This section is structured in three parts. First, the basic idea of our approach that allows us to rank rehearsal buffers of the same size according to their informational value in section 5.2.1 is introduced. Building on this, an optimization problem that aids as a rule for selecting which examples to store in a rehearsal buffer of fixed size is introduced in section 5.2.2. Our algorithm is then derived as an approximate solution to the optimization problem in section 5.2.3.

5.2.1. A Probabilistic Perspective on Sampling from a Rehearsal Buffer

All memory-based methods for continual learning with DNNs discussed in section 3.4 share a rehearsal buffer that serves as a memory from which previously encountered training examples are drawn randomly and interleaved with new data during training on a new task as their fundamental protection against catastrophic forgetting. In the following we use the notation $\mathcal{M} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$ for a set of N training examples stored in the rehearsal buffer. In the case of an online continual learning on supervised image classification, each training example is given by an image $\mathbf{x}_i$ with its corresponding class label $\mathbf{y}_i$ as a one-hot vector. Selecting examples uniformly at random from $\mathcal{M}$ is then equivalent to sampling from the distribution

$$p(\mathbf{x}, \mathbf{y}|\mathcal{M}) = p(\mathbf{x}|\mathbf{y}, \mathcal{M})p(\mathbf{y}|\mathcal{M}), \tag{5.1}$$

which serves as an approximation to a true data generating distribution $p(\mathbf{x}, \mathbf{y})$ of the observed data and depends on the set of training examples $\mathcal{M}$ that are stored. Note that the rehearsal buffer $\mathcal{M}$ contains only samples from $p(\mathbf{x}, \mathbf{y}|\mathcal{M})$ while its exact form is unknown.

Interpreting the process of randomly selecting examples from the rehearsal buffer as sampling from a distribution that depends on $\mathcal{M}$ enables us to rank different buffers based on their contents. For this, we can think of a rehearsal buffer as a communication channel between previously learned tasks and the current task during continual learning. Choosing different training examples on previous task to store in the rehearsal buffer is then equivalent to sending different messages through time. This similarity to a classical communication channel allows for using well established principles from the field of information theory to analyze and optimize the composition of a rehearsal buffer. As one of the most central quantities in information theory, the Shannon entropy [1] is a natural choice for this since it measures the informational value of a random variable given its probability density function. For a pair of random variables X and Y it is given by

$$H(X, Y) = -\mathbb{E}_{p(\mathbf{x},\mathbf{y})} \left[\log p(X, Y)\right], \tag{5.2}$$

where $p(\mathbf{x}, \mathbf{y})$ is the joint probability density or mass function. Note that the entropy is usually measured in bits and the logarithm is taken with base 2. But since we are just interested in comparing the entropy of different distributions, the absolute value and therefore its unit are irrelevant.

Much more important for the derivation of our method is its interpretation as the number of bits on average required to describe the random variables, i.e. its informational value. Overall, the rehearsal buffer is intended to serve as a memory of previously learned knowledge and sampling from it should provide useful information. The entropy as an informational value of a random variable serves as a way to quantify this rather vague concept of providing useful information. Under this perspective, a rehearsal buffer with high entropy associated with sampling uniformly at random from it provides more useful information than one with a low entropy. Suppose, for example, that the rehearsal buffer is filled with N copies of the same training example. It would certainly not provide much useful information as sampling from it is totally predictable and correspondingly the associated entropy would be zero. Any other buffer with more than one unique stored example would feature a higher entropy and also be considered to provide more useful information during rehearsal.

5.2.2. Entropy Maximization as a General Rule for Sample Selection

Given this perspective and notation on a rehearsal buffer and randomly sampling from it introduced in the previous section, a simple rule for selecting training examples to store in a rehearsal buffer can be derived. For this, consider an already filled rehearsal buffer $\mathcal{M}$ of size N. If an unseen training example is encountered we have to decide between $N + 1$ different actions: We can replace one of the N already stored examples with the new one or we can simply leave the rehearsal buffer unchanged and move on to the next training example from the data stream. Each of these actions results in a new rehearsal buffer $\mathcal{M}_l$ for $0 \leq l \leq N$, where $\mathcal{M}_0$ denotes an unchanged rehearsal buffer and $\mathcal{M}_l$ for $l \neq 0$ is the result of replacing the k-th stored with the new training example. Following the premise of maximizing the informational value when sampling from the rehearsal buffer, selecting an action to enact out of these $N + 1$ possible ones is performed by solving the optimization problem

$$\max_l H(X, Y|\mathcal{M}_l) = \max_l -\mathbb{E}_{p(\mathbf{x},\mathbf{y}|\mathcal{M}_l)} \left[\log p(X, Y|\mathcal{M}_l)\right], \tag{5.3}$$

where $H(X, Y|\mathcal{M}_l)$ denotes the entropy of $p(X, Y|\mathcal{M}_l)$. The idea of maximizing the entropy for selection problems is well known in maximum entropy sampling [8] or the training of decision trees [5] but was never applied to sample selection in the context of online continual learning. Although the above optimization problem is easily formulated, solving it efficiently is not trivial. The first challenge lies in the fact that it is a discrete optimization problem since we are optimizing over buffer states. Thus gradient-based methods can not be used as the gradient with respect to the buffer contents does not exist. In addition to that, the exact form of $p(\mathbf{x}, \mathbf{y}|\mathcal{M}_l)$ is unknown and has to be approximated. Finally, using an exhaustive search over all potential solutions is possible but infeasible since the entropy has to be determined for all of the potentially thousands of possible actions for each newly encountered training example. We therefore restrain from finding an exact solution to the optimization problem given in (5.3) and instead propose an algorithm to solve it approximately in the next section.

5.2.3. An Approximate Solution to an Infeasible Optimization Problem

In the following we assume that the underlying type of problem in an online continual learning sequence is supervised image classification as this allows us to easily compare our approach to related work such as GSS. The motivation and introduction of our approach, however, is general enough such that is can be applied to other settings as well. The starting point for our approximate method is a decomposition of the joint entropy of two random variables X and Y into

$$H(X, Y|\mathcal{M}_l) = H(X|Y, \mathcal{M}_l) + H(Y|\mathcal{M}_l), \tag{5.4}$$

where $H(X|Y, \mathcal{M}_l)$ is the conditional entropy of X conditioned on Y. Maximization of $H(X, Y|\mathcal{M}_l)$ in general is therefore performed by maximizing $H(Y|\mathcal{M}_l)$ only over Y while $H(X|Y, \mathcal{M}_l)$ must be maximized jointly over X and Y. Considering our intention of maximizing the joint entropy by selecting the action on our buffer that maximizes the joint entropy, this decomposition is not simplifying the original problem, at least if we intend to strictly maximize $H(X, Y|\mathcal{M}_l)$. This is due to the fact that replacing an example in $\mathcal{M}_l$ with a new one in general has an influence on both $H(X|Y, \mathcal{M}_l)$ and $H(Y|\mathcal{M}_l)$ at the same time.

But if we instead allow for an action on the buffer to occasionally cause a decrease in the joint entropy, the decomposition above allows for a simple approximate algorithm for maximizing $H(X, Y|\mathcal{M}_l)$. For this, we first select the subset of actions on the buffer that maximize $H(Y|\mathcal{M}_l)$ and then choose from those the action that maximizes $H(X|Y, \mathcal{M}_l)$. This is an approximate solution to the optimization problem in (5.3) since it optimizes sequentially first over Y and then over X instead of jointly optimizing over X and Y. As our experimental results in section 5.3 show, it is a reasonable simplification that allows for a fast but still capable algorithm.

Building on this basic idea and the restriction to supervised image classification our method is derived in the following. Since in this case Y is a discrete random variable representing class labels, it follows a categorical distribution $p(\mathbf{y}|\mathcal{M}_l)$ whose entropy is at its maximum for equal class probabilities.

The first step in our method is therefore to select a subset from all possible actions that result in a more uniform distribution of classes in the rehearsal buffer. This is done by replacing an example of the majority class in the rehearsal buffer with a new one. Any action in this subset then increases $H(Y|\mathcal{M}_l)$ given that the replacing example is of a class other than the majority class. Otherwise the balance of classes in the buffer and therefore also $H(Y|\mathcal{M}_l)$ remain unchanged. If it is not possible to determine a single majority class, i.e. two or more classes have the same and highest number of examples in the rehearsal buffer, one of these is selected randomly for replacement.

While choosing a subset of actions in order to maximize $H(Y|\mathcal{M}_l)$ is rather simple under the given circumstances, the next step of our method is complicated due to the lack of knowledge about an exact analytical form of $p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l)$. An approximation of it in form of a Kernel

Density Estimator (KDE) is given by

$$p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l) \approx \frac{1}{M_y} \sum_{m=1}^{M_y} K(\mathbf{x} - \mathbf{x}_y\,[m]), \tag{5.5}$$

where M_y is the number of examples $\mathbf{x}_y$ in $\mathcal{M}_l$ with label y and $K(\mathbf{x}) \in \mathbb{R}^+$ is a kernel function. Choosing a Gaussian, i.e. $K(\mathbf{x}) = (2\pi)^{-1/2} \mathrm{e}^{-\|\mathbf{x}\|_2^2/2}$, as the kernel function yields

$$\begin{aligned} H(X|Y, \mathcal{M}_l) &= -\mathbb{E}_{p(\mathbf{x}|\mathbf{y},\mathcal{M}_l)} \left[\log p(X|Y, \mathcal{M}_l)\right] \\ &\approx -\mathbb{E}_{p(\mathbf{x}|\mathbf{y},\mathcal{M}_l)} \left[\log \frac{1}{M_y\sqrt{2\pi}} \sum_{m=1}^{M_y} \mathrm{e}^{\dfrac{-\|\mathbf{x} - \mathbf{x}_y\,[m]\,\|_2^2}{2}}\right], \end{aligned} \tag{5.6}$$

where we did not include a kernel width parameter for a lighter notation and since it does not change the following reasoning. With this particular choice of a kernel function, the second step of our method is then equivalent to solving the optimization problem

$$l^\star = \arg\max_l H(X|Y, \mathcal{M}_l) = \arg\max_l \int -p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l) \log p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l) \mathrm{d}\mathbf{x}, \tag{5.7}$$

where $l^\star$ is the index of a particular $\mathcal{M}_{l^\star}$ that from all possible memories achieves the highest conditional entropy. Similar to Gaussian mixture models [25], there is unfortunately no analytical solution for determining the entropy of this KDE although $p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l)$ is known according to (5.5). It can therefore only be evaluated by approximations, e.g. Monte Carlo sampling. For our approach we choose to approximate it by

$$l^\star = \arg\max_k H(X|Y, \mathcal{M}_l) \approx \arg\max_l \frac{-1}{M_y} \sum_{m=1}^{M_y} \log \frac{1}{M_y\sqrt{2\pi}} \sum_{n=1}^{M_y} \mathrm{e}^{\dfrac{-\|\mathbf{x}_y\,[m] - \mathbf{x}_y\,[n]\,\|_2^2}{2}}, \tag{5.8}$$

where the expectation operator is approximated by sampling only at the modes $\mathbf{x}_y\,[m]$ of $p(\mathbf{x}|\mathbf{y}, \mathcal{M}_l)$. Although this is a rather imprecise approximation due to a potentially small number of samples with which an expectation over a high-dimensional density is evaluated, it can be argued that the highest density is concentrated around the modes. From this approximation a simple decision rule for deciding which sample stored in the rehearsal memory is to be replaced is derived. More details on this derivation is presented in the appendix under section A. This results in

$$l^\star = \arg\max_l H(X|Y, \mathcal{M}_l) \approx \arg\max_l \sum_{m=1}^{M_y} \sum_{n=1}^{M_y} \|\mathbf{x}_y\left[m\right] - \mathbf{x}_y\left[n\right]\|_2^2, \tag{5.9}$$

which implies that the action performed on the buffer that maximizes the conditional entropy $H(X|Y, \mathcal{M}_l)$ is approximately equal to the action that maximizes the sum of squared distances between the kernel modes, i.e. the stored examples of class y in the rehearsal memory.

Finally, we propose a simple and fast algorithm for solving the optimization problem presented in (5.9) by randomly selecting the i-th example of the majority class in the buffer for replacement with probability

$$p_i = \frac{1 - d_i/d_{max}}{\sum_j \left(1 - d_j/d_{max}\right)}, \tag{5.10}$$

where d_i is the minimum distance of the i-th example to all other examples of the majority class and d_{max} is the maximum of these minimum distances used for normalization. This again does not strictly maximize the distance between all examples of the majority class in the buffer during every step but it replaces examples which lie close to another example of the same class with high probability. If during this replacement process the newly added example lies even closer to another one of the same class, it will be selected for replacement in the next iteration with an even higher probability. Overall this prevents a static buffer when it achieves maximum entropy and does not change anymore regardless of how many new examples of the same class are encountered. Such a static buffer can have a negative impact on performance as repeatedly sampling from the same small buffer can cause overfitting of a DNN to the stored examples. This risk is reduced when samples in the buffer are replaced regularly. Finally, we summarize the steps of our method discussed above in Algorithm 2 and call it Entropy-based Sample Selection (ESS).

5.3. Experiments and Results

The experiments presented in this section are structured in two parts: A direct comparison with GSS on sequences from the MNIST and CIFAR10 datasets. This is followed by a comparison with other recently published methods, like Maximally Interfered Retrieval (MIR) [81] and Adversarial Shapely value Experience Replay (ASER) [122], which propose improved

Algorithm 2: Entropy-based Sample Selection

```
Input: $\mathbf{x}$, $y$, Buffer $\mathcal{M}$
Output: Updated $\mathcal{M}$
1  if $|\mathcal{M}| < N$ then
2      $\mathcal{M} \leftarrow \mathbf{x}, \mathbf{y}$
3  end
4  if $|\mathcal{M}| = N$ then
5      $C \leftarrow$ samples of majority class in $\mathcal{M}$;
6      for $\mathbf{x}_i \in C$ do
7          $d_i = \min_{\mathbf{x}_j \in C \setminus \mathbf{x}_i} \|\mathbf{x}_i - \mathbf{x}_j\|_2$
8      end
9      $d_{max} \leftarrow \max_i d_i$;
10     $i \sim p_i = \frac{1 - d_i/d_{max}}{\sum_j \left(1 - d_j/d_{max}\right)}$;
11     $\mathbf{x}_i, \mathbf{y}_i \leftarrow \mathbf{x}, \mathbf{y}$
12 end
```

rehearsal mechanisms in addition to a sample selection strategy. Experiments in the second part are performed using the CIFAR10 and CIFAR100 datasets only.

The online continual learning sequences that these methods are evaluated on differ significantly. Similar to Aljundi et al., we evaluate and compare our method with GSS in the first part on shorter sequences from the MNIST and CIFAR10 datasets. In the experiments of Aljundi et al. an equal number of examples per class in the online continual learning sequence is referred to as balanced while sequences with 10 times more examples from one class are defined as unbalanced[2]. These experiments are intended to test the ability of sample selection strategies to maintain a representative rehearsal memory. In addition, these sequences are constructed only from a subset of the full datasets. Although Aljundi et al. do not provide a reason for this in [83] we follow their experimental setup for a fair comparison. When comparing our method with MIR and ASER in the second part, only balanced sequences from the full CIFAR10 and CIFAR100 datasets are used.

Neither GSS nor MIR or ASER utilize knowledge distillation [39] or any form of data augmentation although these methods are standard techniques in order to improve the performance of continual learning methods. Especially in the online setting, where each example is only accessible once to the model and only very small rehearsal memories are used, it seems counter

[2]Note that this definition is arbitrary but was used in order to compare our methods with those from the literature in a fair way.

Table 5.1.: Final classification accuracy in % of evaluated methods on the balanced online continual learning sequences of the reduced MNIST dataset.

	Buffer size $\|\mathcal{M}\|$		
Method	300	400	500
Random	27.4±1.7	28.3±2.5	30.4±4.1
Reservoir	79.0±3.6	76.9±5.2	78.4±2.8
GSS-IQP [83]	75.9±2.5	82.1±0.6	84.1±2.4
GSS-Greedy [83]	82.6±2.9	84.6±0.9	84.8±1.8
ESS (Ours)	**84.7±1.3**	**86.0±1.3**	**87.4±1.1**

intuitive to not use these methods. But in order for a fair comparison, we also do not use these techniques for our method.

5.3.1. Direct Comparison with GSS

First, GSS is evaluated on MNIST by splitting the complete dataset into five tasks, $\mathcal{T}_{1\leq i\leq 5}$, with two classes each in the order $0, 1 \rightarrow 2, 3 \rightarrow 4, 5 \rightarrow 6, 7 \rightarrow 8, 9$. Instead of using all available training data we follow Aljundi et al. and use only 1000 randomly selected examples per task. Although not explicitly mentioned in [83], it is evident from the publicly available original implementation that each task contains an equal number of examples from its two corresponding classes. GSS is evaluated on this balanced online continual learning sequence using a two layer MLP with 100 neurons in each hidden layer with a ReLU activation for different buffer sizes. The output layer is expanded according to the ICL scenario description presented in section 2.2 to accommodate the new classes of every new task. In order to obtain comparable results, we evaluate our method on the same sequence using a MLP of identical size with the same buffer sizes. We train our model using SGD with a learning rate of 0.1 and a batch size of ten. Similar to GSS, we allow our method to train the MLP for multiple steps, four in this case, on a mini batch of new training data while mixing it with newly sampled data from the rehearsal buffer in each step before moving on the next mini batch in the sequence. All runs are repeated ten times with independent random initializations and results are reported in Table 5.1 as the average and standard deviation of the final classification accuracy on the complete test dataset.

First, the results of our method are compared with the baselines of randomly selecting examples to be replaced from the buffer and Reservoir Sampling[3] [4] in Table 5.1. It is obvious that the latter is a strong baseline as it comes close to the performance of our method while the former is comparatively weak. Both versions of GSS, the exact Interger Quadratic Program (IQP) solution and the greedy approximation of it, clearly beat the Random and Reservoir Sampling baseline for $|\mathcal{M}| > 300$. Interestingly, the greedy approximation achieves better results than the exact IQP solution for GSS, especially when $|\mathcal{M}| = 300$. In this case the latter is even worse than Reservoir Sampling. Our method, ESS, outperforms not only the baselines but also both versions of GSS.

In addition to just testing the performance on a balanced online continual learning sequence, where each task has the same number of training examples, we follow Aljundi et al. and test our approach on unbalanced sequences. For this, we create a sequence with 2000 examples for one and 200 for all remaining tasks. The first sequence has 2000 training examples from classes 1 and 2 while the remaining ones feature only 200. During the second sequence classes 3 and 4 are represented by 2000 while the remaining tasks feature only 200 training examples. This is repeated until five sequences are created. Due to the sections with differing length in an unbalanced sequence, it is much more difficult for an algorithm to properly manage the buffer. It is, for example, unknown when a much longer section begins and how long it is. Therefore, a more elaborate buffer management algorithm when compared with balanced sequences is needed. Reservoir Sampling, for example, is designed specifically for the case of balanced sequences and hence will perform much worse on unbalanced ones.

Table 5.2 summarizes the results of this experiment with a buffer size of 300 and all other hyperparameters are identical to those of the previous experiment. Both variants of GSS again outperform the baselines significantly. Similar to the previous one, our method outperforms GSS in this experiment as well, especially for the first sequence. Given that the total number of examples in each of the five tested sequences is identical, a sample selection method should lead to roughly the same final accuracy. While GSS in both of its forms shows more stable results than reservoir sampling, our method shows even less variation in the final accuracy.

The direct comparison of our method with GSS is concluded by an experiment on the CIFAR10 dataset. For this a balanced online continual learning sequence is constructed by splitting the dataset into five tasks with 2000 samples each. These tasks again feature an equal number of examples from two consecutive classes of the original dataset. Instead of the small MLP

[3]The Reservoir Sampling algorithm begins with filling the buffer $\mathcal{M}$. For every newly arriving sample $\mathbf{x}_i, \mathbf{y}_i$ it generates a random integer j uniformly in the interval $[1, \ldots, i]$. If $j \in [1, \ldots, |\mathcal{M}|]$ it replaces the j-th sample in the buffer with $\mathbf{x}_i, \mathbf{y}_i$.

Table 5.2.: Final classification accuracy in % of evaluated methods on the unbalanced online continual learning sequences of the MNIST dataset.

	Sequence				
Method	1	2	3	4	5
Random	44.4±4.8	40.8±4.2	35.3±3.1	34.9±2.1	21.8±2.6
Reservoir	53.6±5.1	63.9±2.1	76.4±1.9	73.2±3.0	67.7±2.6
GSS-IQP [83]	75.9±3.2	76.2±4.1	79.1±0.7	76.6±2.0	74.7±1.8
GSS-Greedy [83]	71.2±3.6	78.5±2.7	81.5±2.3	79.5±0.6	79.1±0.7
ESS (Ours)	**82.3±2.1**	**82.2±1.7**	**82.4±2.3**	**81.8±2.7**	**82.6±1.6**

Table 5.3.: Final classification accuracy in % of evaluated methods on the balanced online continual learning sequences of the reduced CIFAR10 dataset.

Method	Buffer size $|\mathcal{M}| = 1000$
Random	17.3±0.6
Reservoir	32.9±2.1
GSS-Greedy [83]	33.6±1.7
ESS (Ours)	**37.6±2.0**

used in the previous experiments on MNIST, a reduced version of ResNet18 as described in Table 3.5 from section 3.7 and a rehearsal memory of size 1000 is used. All other hyperparameters, however, remain unchanged and still no data augmentation, not even the standard horizontal flipping and random cropping used for training on CIFAR10 in the original ResNet publication [44], is used.

Comparing the results of this experiment shown in Table 5.3 reveals that, unlike in the previous experiments on MNIST, GSS performs only marginally better than reservoir sampling. The random selection baseline unsurprisingly achieves the worst results and our method again shows a significant improvement over GSS with an approximately 12% higher average accuracy and similar variance. A possible reason for GSS being outperformed by our method could be its reliance on gradient information. While our approach is completely independent from the DNN used as a classifier, the only information that GSS uses to select which samples to store and which to replace depends on the data itself but also the weights and loss function utilized in the gradient calculation. Since GSS additionally only considers gradients of individual samples, any noise in this gradient might have a significant impact on the selection.

5.3.2. Comparison with other Related Methods

While the first part of our experiment section focuses on the direct comparison of the sample selection strategies only, this section compares our method not only to GSS and baselines but also other methods for online continual learning. These also feature improved rehearsal strategies in addition to a sample selection strategy and therefore improve an additional aspect of rehearsal based online continual learning. MIR, for example, selects those samples from the rehearsal memory that show the highest increase in loss if an update step would be performed on a mini batch containing only the new data. These are then combined with the new data in order to perform an actual update step. As a consequence, they have the potential to outperform our method.

In contrast to the previous experiments, the following ones all use balanced online continual learning sequences that are constructed using the complete CIFAR10 or CIFAR100 dataset. While the former is split into five tasks featuring two classes each, the latter is split into ten tasks where each of those features ten classes. Again a reduced ResNet18 as described in Table 3.5 is trained using SGD with a learning rate of 0.1 and batch size of ten. But unlike in the previous experiments where the online continual learning sequences are constructed using only a subset of the original datasets, we only perform one update step on each mini batch consisting of new and randomly sampled examples from the rehearsal memory.

The results for CIFAR10 with buffer sizes of 200, 500 and 1000 are shown in Table 5.4. Similar to the previous experiment on the reduced CIFAR10 dataset, GSS and our method outperform the random selection baseline. Unlike in the previous experiment, both do not outperform reservoir sampling for $|\mathcal{M}| > 200$. Only MIR and ASER are able to achieve a significantly higher accuracy than reservoir sampling in this experiment. Given that both of these use an improved version of rehearsal when compared to our method and GSS, this is not surprising as this setup with a balanced online continual learning sequence does not pose a serious challenge to the sample selection algorithm. The strong performance of the reservoir sampling baseline is testimony of this, since it was originally proposed in [4] as an optimal selection strategy from balanced sequences and typically fails on unbalanced ones. It is also worth mentioning that the advantage of MIR and ASER over our method decreases with an increasing buffer size and all methods except the weak random selection baseline achieve a similar accuracy for a buffer size of 1000. EWC as a regularization-based method achieves to lowest results and is even worse than the random selection baseline. The performance of AGEM does not seem

Table 5.4.: Final classification accuracy in % of evaluated methods on the balanced online continual learning sequences of the CIFAR10 dataset. † Results reported in [122].

	Buffer size $\|\mathcal{M}\|$		
Method	200	500	1000
Random	18.5±0.3	18.6±0.3	18.8±0.2
Reservoir	24.6±2.1	31.3±3.4	40.6±4.0
EWC [53]†	17.9±0.3	17.9±0.3	17.9±0.3
AGEM [84]†	22.7±1.8	22.7±1.9	22.6±0.6
GSS [83]†	26.9±1.2	30.7±1.2	40.1±1.4
MIR [81]†	**28.3±1.6**	35.6±1.2	42.4±1.5
ASER [122]	26.4±1.5	**36.3±1.2**	**43.5±1.4**
ESS (Ours)	23.7±2.5	30.3±2.3	40.4±2.5

to increase with an increasing buffer size, which seems counter intuitive and might hint at its simplifications over GEM being overly simplistic.

The CIFAR100 dataset is arguably more challenging than CIFAR10 as it includes ten times more classes with ten times fewer examples per class while featuring natural objects of a similar complexity. The experimental results in Table 5.5 are therefore obtained with buffer sizes of 1000, 2000 and 5000 while all other hyperparameters are identical to the CIFAR10 experiment. Although the underlying online continual learning sequence is also balanced as in the previous experiment, our method outperforms the random selection baseline and reservoir sampling. But only for $|\mathcal{M}| > 1000$ does our method show a significantly higher result than Reservoir Sampling. GSS is also only in this case able to outperform Reservoir Sampling. In addition, our method also outperforms GSS and again achieves results very close to MIR and ASER especially for a larger rehearsal memory. Comparing ESS with GSS on CIFAR10 and CIFAR100 is seems that only for buffer sizes greater than 1000 is our method able to outperform GSS on balanced sequences.

5.4. Conclusion

The unique challenges in online continual learning where data becomes available in a stream of individual examples or small batches and the need for computationally algorithms result in the research question:

Table 5.5.: Final classification accuracy in % of evaluated methods on the balanced online continual learning sequences of the CIFAR100 dataset. † Results reported in [122].

	Buffer size $\|\mathcal{M}\|$		
Method	1000	2000	5000
Random	6.9±0.3	6.9±0.2	6.9±0.3
Reservoir	9.7±0.7	10.8±0.7	11.2±0.5
EWC [53]†	4.8±0.2	4.8±0.2	4.8±0.2
AGEM [84]†	9.5±0.4	9.3±0.4	9.7±0.3
GSS [83]†	9.3±0.2	10.9±0.3	15.9±0.4
MIR [81]†	11.2±0.3	14.1±0.2	21.2±0.6
ASER [122]	**14.0±0.4**	**17.2±0.5**	**21.7±0.5**
ESS (Ours)	10.3±0.8	13.2±1.1	19.9±1.5

How can the buffer of a rehearsal-based method for continual learning be managed in a simple but effective way to enable online continual learning?

In this chapter, we adopt a probabilistic perspective on the process of selecting samples to be stored in a small memory for a later use in rehearsal during online continual learning. By using the well known entropy maximization principle, we derive our method ESS summarized in Algorithm 2. The experimental results for the MNIST and CIFAR10 datasets obtained in a direct comparison with the related sample selection method GSS in section 5.3.1 show that ESS is capable of outperforming it on balanced and unbalanced online continual learning sequences. Further experiments on the CIRAR10 and CIRAR100 datasets in section 5.3.2 partially confirm this finding. A comparison to other recently proposed methods that feature improved sample selection and rehearsal strategies shows that our method is outperformed by MIR and ASER. But although their advantages for smaller buffer sizes was substantial, ESS achieves only marginally worse results for larger rehearsal memories.

Our method ESS is much simpler than GSS since it only requires distance calculations on the data in a rehearsal memory and keeps track of how much examples per class are stored. Additionally it is capable to outperform GSS in most of the tested dataset and buffer sizes. Only for balanced sequences and rehearsal buffers sizes smaller than 1000 on CIFAR10 is ESS outperformed by GSS. The comparison to MIR and ASER, which both not only use improved sample selection but also rehearsal strategies, shows that there is still some potential for improvement in online continual learning. This can be realized by improving how data from the rehearsal memory is used during training in addition to only optimizing the sample selection strategy. Such an improved sampling from the buffer can in principal also be applied to ESS. The focus of this work, however, is purely on one aspect of rehearsal-based continual learning, namely improving the buffer management, i.e. which data to store and what to replace, and not how to use the stored data during training. An improvement to this aspect is presented in chapter 7 of this thesis.

Chapter 6.

Dataset Condensation and Synthetic Data for Rehearsal

This chapter presents a method published by Wiewel and Yang in [125] for learning a small set of synthetic data for rehearsal during continual learning. It is based on DCGM and uses the same approach of matching gradients for learning synthetic data. The main novelty of this method compared with DCGM and others is the way synthetic data is represented, which allows for sharing common components between individual samples in order to avoid storing redundant information and therefore to improve memory efficiency. Being able to learn synthetic data directly in a more efficient representation is of interest since it allows for improving rehearsal-based continual learning by simply storing more information in a fixed-size memory when compared with other methods. Experimental results on commonly used datasets indeed indicate a significantly increased performance over related approaches especially for small rehearsal memories. In addition to this comparison with related methods, the learned data is compared to original and synthetic data generated by other approaches.

6.1. Motivation

Storing data for rehearsal is a simple but effective approach for mitigating catastrophic forgetting and enabling continual learning with DNNs. It is therefore a central component in many recently proposed methods, some of which are referenced in section 3.4 and the other chapters of this thesis. An intuitive correlation between the performance of a method and the size of its rehearsal memory is not only observable in the results presented in section 5.3 but many publications on continual learning as well [56, 58, 81, 83, 87, 97, 99, 107, 109, 116]. This in combination with a typically limited size of the rehearsal memory makes compression of stored data a promising direction for improving the already strong performance of rehearsal. The goal of this is to capture as much information as possible of a particular dataset with a small set of compressed examples and therefore utilize the memory in an efficient way.

There are common compression algorithms like JPEG for images and MP3 for audio which are in widespread use today. In addition compression based on deep learning has also been explored in the literature [61, 73, 85, 115]. Both of these approaches can greatly reduce the amount of storage required to store, for example, an image during rehearsal. A learning based approach for storing more data in the context of continual learning is explored by Hayes et al. in [103]. They combine an encoder with weights that are fixed after a pre-training phase with Vector Quantization (VQ) and a trainable classifier in order to greatly reduce the training and storage cost for continual learning. This approach essentially projects the original problem from the input space to a much lower dimensional feature space that not only speeds up training but also permits to store much more data in a fixed size rehearsal memory. A similar approach with a fixed pre-trained encoder but without any further compression in the latent space is proposed by Pellegrini et al. in [106]. The authors focus on reduced computational complexity for real-time continual learning in applications that are restricted in their storage capacity and computational power.

In contrast to these approaches that compress original training examples either in the input space directly or in a learned feature space, there is also a recently proposed approach that learns synthetic data as discussed in section 3.4.3. Instead of compressing data these approaches learn entirely synthetic data that is not present in the original training dataset. This offers another degree of freedom since the content of data can be learned to capture the most essential information. HAL [114], for example, learns one anchor point per class that is intended to minimize catastrophic forgetting. It thereby couples the process of learning synthetic data with the objective of overcoming catastrophic forgetting. An anchor point can therefore be interpreted as synthetically generated data that is representative and explicitly minimizes forgetting. Instead of learning just one synthetic example per class, DCGM [129] learns multiple examples in order to maximize the performance of a DNN when trained on this synthetic dataset. Although this approach lacks a direct connection to continual learning or catastrophic forgetting, it can still be applied to it as the authors show with their experiments.

These approaches of learning synthetic data offer new possibilities of improving rehearsal based continual learning since they are orthogonal to the already existing and proven techniques of data compression mentioned above, i.e. the learned data can additionally be compressed in the same way as original training data. HAL and DCGM demonstrated the feasibility of such approaches in the context of continual learning but can still be improved upon, which is the motivation for this chapter. Although DCGM achieves better results than using original data for rehearsal, it is ignoring possible redundancy in the learned data. In the case of supervised image classification for example, it learns multiple synthetic examples of the same class which might share common features that are stored individually with every learned example. This motivates the following research question:

How can synthetic data be learned for rehearsal-based continual learning while reducing the redundancy of data stored in memory?

For this, Wiewel and Yang propose their method Condensed Composite Memory Continual Learning (CCMCL) in [125], which is based on DCGM and improves upon it by representing each synthetic example as a weighted combination of shared components. The remainder of this chapter introduces their method, presents an experimental evaluation and closes with a short conclusion on its effectiveness and applicability in continual learning.

6.2. Method

Since the proposed CCMCL method is based on learning synthetic data via the process of dataset condensation, the first part of this chapter outlines the basic idea and practical implementation of it. Although multiple different approaches for this have been proposed in recent publications [79, 121, 123, 128], the focus of this chapter is on DCGM proposed by Zhao, Mopuri, and Bilen in [129] since it is the underlying algorithm of CCMCL and was shown to improve rehearsal performance by the authors. A key aspect that is not covered in detail by the authors is how to implement it for rehearsal in the context of continual learning. We therefore devote section 6.2.2 to outline a possible implementation that includes rehearsal explicitly. The key innovation of CCMCL over DCGM, namely sharing common features between synthetic examples, is introduced in section 6.2.3. Finally, the complete algorithm is presented and discussed in section 6.2.4.

6.2.1. Dataset Condensation with Gradient Matching

According to [129] the goal of dataset condensation is to find a set of synthetic examples such that a model trained on them has a similar generalization performance as one that is trained on the original dataset. Approaches for this can be formulated in different ways.

Wang et al. [79], for example, propose a nested optimization

$$\mathcal{S}^\star = \arg\min_{\mathcal{S}} \mathcal{L}(\boldsymbol{\theta}_\mathcal{S}, \mathcal{T}) \quad \text{subject to} \quad \boldsymbol{\theta}_\mathcal{S} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{S}), \tag{6.1}$$

where $\mathcal{T}$ is the original training dataset and $\mathcal{S}$ is the set of synthetic examples. The main idea is that the parameters $\boldsymbol{\theta}_\mathcal{S}$ depend on synthetic data. This allows for optimizing the data $\mathcal{S}$ through minimization of a loss on the training dataset. While this formulation leads to a direct minimization of a loss $\mathcal{L}$ on the original training dataset, it does not scale since the inner optimization, i.e. the training on synthetic examples, can involve large models and potentially many iterations to find $\boldsymbol{\theta}_\mathcal{S}$.

Another approach mentioned by Zhao, Mopuri, and Bilen [129] matches the parameters $\boldsymbol{\theta}_\mathcal{S}$ of a DNN trained on the set of synthetic examples with those of one trained on the original training dataset. But this does not scale either as the authors point out. The reason for this is again that this approach requires a costly nested optimization similar to (6.1) where the inner problem involves a potentially large scale training of a DNN on the synthetic data.

In order to overcome this lack of scalability, the authors therefore propose to match the gradients of a DNN obtained on the original and synthetic data throughout the training process. Performing this for multiple different random initial parameters then yields

$$\mathcal{S}^\star = \arg\min_{\mathcal{S}} \mathbb{E}_{p(\boldsymbol{\theta}_0)} \left[\sum_{t=0}^{T-1} D(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t, \mathcal{T}), \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t, \mathcal{S})) \right], \tag{6.2}$$

where T is the number of training iterations, $p_{\boldsymbol{\theta}_0}$ is the distribution for random initialization and $D(\cdot, \cdot)$ is a distance measure based on the cosine similarity that is discussed in more detail throughout section 3.4.3. The main idea of this approach is that synthetic data $\mathcal{S}$ leads to gradients that are similar to those determined on original data $\mathcal{T}$. Crucially, this needs to be ensured not only for one specific step in the training process and only one random initialization but for all steps and many different initial weights $\boldsymbol{\theta}_0$. The authors also argue that while this might constrain the overall optimization process more than matching the endpoints of training, it allows for a more guided optimization. In addition, this approach is much faster since it does not require a nested optimization with a costly inner training on synthetic data.

Zhao, Mopuri, and Bilen further point out some key aspects that contribute to improved results of their method. One of these is the separate processing of individual classes in their experiments on image classification problems. As the authors point out, this reduces the memory requirement and leads to better convergence of the learned synthetic data since imitating the mean gradient of an individual class is easier than to imitate that of multiple classes. Furthermore, since only a small set of synthetic examples is used during training, estimating reliable statistics for batch normalization layers is difficult. To overcome this problem, the authors use instance normalization. Their experimental results also suggest that leaky ReLU performs better than ReLU and average outperforms max pooling. As a possible explanation for this a denser flow of gradients from the output to synthetic inputs is put forward by the authors.

Dataset condensation using DCGM not only allows for learning synthetic images in the case of supervised image classification, but also to learn the corresponding labels as well. This is possible since both contribute to the loss in a differentiable way if the common categorical cross entropy loss is used. A gradient with respect to the labels can therefore theoretically be used to optimize the complete synthetic example, which contains both the input and its corresponding label. But experiments by Zhao, Mopuri, and Bilen have shown that a joint optimization of both is challenging. According to the authors these difficulties are caused by the close dependence between the contents of input and their label, which leads them to only synthesize the input, e.g. an image, for a given and fixed label.

6.2.2. Adaptation to Rehearsal

Applying dataset condensation for rehearsal in the context of continual learning comes with some aspects that differ from the original context. Instead of having access to the complete training dataset, in a typical continual learning setup only a small part of it is available at any time. In the case of supervised image classification for example, only a few classes might be present in the current task and all previously encountered ones are only accessible through the rehearsal memory. This is in contrast to the original setup for which dataset condensation is proposed for. Even if the stored data is the result of applying dataset condensation, some performance degradation is to be expected since not all original training data is available.

Solving the optimization problem (6.2) requires minimization of the expected distance between gradients with respect to many different initial parameters $\boldsymbol{\theta}_0$ drawn from the distribution $p_{\boldsymbol{\theta}_0}$. As a strategy for approximating this expectation, Zhao, Mopuri, and Bilen propose to simply reinitialize the DNN used for dataset condensation after a certain number of training steps T have been used to minimize the distance between gradients. In the original setup considered by the authors this is not a problem since the goal is to generate only synthetic data which can be used in any following step, for example to speed up hyperparameter optimization. But in the context of continual learning a sequence of tasks is to be learned with limited storage capacity. In order to reinitialize the DNN without losing weights learned on previously encountered tasks, they have to be stored in a copy at the beginning of every task. This requires additional memory that is not available for rehearsal anymore. But fortunately such a copy is also required for knowledge distillation that is commonly used in many continual learning methods. Therefore no additional memory is required if dataset condensation is used in conjunction with it since the weights stored for knowledge distillation can be reused for dataset condensation. After this process is completed on a task the DNN can simply be initialized with the weights stored for knowledge distillation.

6.2.3. Sharing Common Components for Increased Memory Efficiency

Although the inspiring work of Zhao, Mopuri, and Bilen already shows that synthetic data can be used in order to compress the information in a training dataset into a small set of synthetic examples, learning and storing them individually can introduce redundancy, since features that define a particular class are learned and stored with each example separately. Choosing a representation that shares components among these examples therefore has the potential to greatly reduce memory requirements as a feature has to be learned and stored only once and can be shared among a potentially large number different training examples.

To this end, Wiewel and Yang propose in [125] to represent each synthetic example

$$\mathbf{x}_j = \sigma\left(\sum_{i=1}^{P} w_{ij}\mathbf{c}_i\right) \tag{6.3}$$

as a composition of P components $\mathbf{c}_i$ weighted by the example-specific weights $w_{ij} \in \mathbb{R}$. In order to ease optimization and ensure a valid range, e.g. of pixels in a synthetic image, the function σ is applied to this composition. Image pixels are normalized such that their values fall into the interval $[0, 1]$ and hence the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ is used in this thesis. By representing synthetic data in this way, common features can be learned as a combination of different weighted components and therefore reduce memory requirements in rehearsal. It is not guaranteed that different components are learned and redundancy among them is avoided. But the much smaller number of them compared to the amount of original data they are learned from leads to different components as observed in the experiments presented in section 6.3.3.

In order to illustrate this, an example of storing Q S-dimensional examples individually as it is done in DCGM and according to (6.3) is considered. If these are stored individually, $|\mathcal{M}_{ind}| = Q \times S$ real numbers are required where the subindex "ind" refers to the storage of individual samples. CCMCL on the other hand requires storing only $|\mathcal{M}_{CCMCL}| = P \times (S + Q)$ real numbers with P components $\mathbf{c}_i \in \mathbb{R}^S$ and $Q \times P$ weights $w_{ij} \in \mathbb{R}$. If $S \gg Q$, which typically holds true for high-dimensional data used as input to DNNs, the storage requirement of CCMCL is dominated by the components while any contribution due to storing the associated weights can be neglected. This means that in this case storing additional synthetic examples in the rehearsal memory comes at a very small cost when compared to storing them individually.

Figure 6.1 demonstrates the advantage of this approach over individual storage for the CIFAR10 dataset that features comparatively small images with a resolution of only 32×32 pixel. It shows the amount of real numbers to be stored in logarithmic scale on its Y-axis over the number of corresponding examples Q on its X-axis. Storing them individually, i.e. $|\mathcal{M}_{ind}|$, is depicted in red while the memory requirement of CCMCL, i.e. $|\mathcal{M}_{CCMCL}|$, is shown in blue for different numbers of components. It is obvious that when P synthetic examples are to be stored using P components, CCMCL has a slightly higher memory requirement than storing them individually since the corresponding weights w_{ij} have to be stored as well. But this is quickly negated when even slightly more examples are added to the rehearsal memory since for these only their weights have to be stored. But of course there is a compromise to be made between the number of used components and weights. If too few of the former are used, the synthetic data might lack variety as there are not enough components to accurately represent all features of a class. Using too much components on the other hand adds significant storage overhead that might not be necessary.

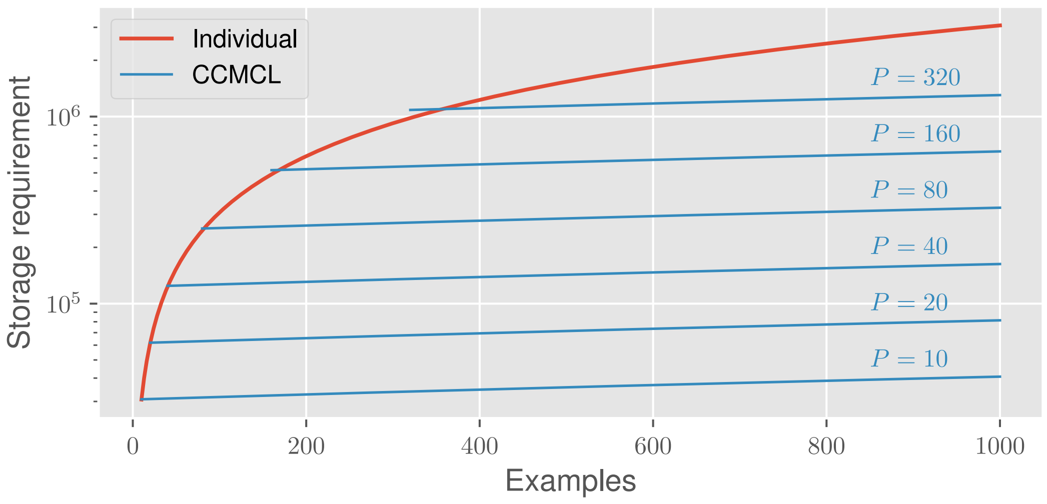


Figure 6.1.: Comparison of memory requirement for storing CIFAR10 images with a resolution of 32×32 pixel individually versus sharing a certain number of common components between them.

6.2.4. Algorithm

Combining dataset condensation with the described adaptations for rehearsal and sharing components according to (6.3) yields the full algorithm of CCMCL proposed by Wiewel and Yang [125] that is summarized with pseudo code in Algorithm 3. It covers the main steps necessary to generate synthetic examples and update the rehearsal memory which can be used as a substitute for regular data in any rehearsal based approach. In CCMCL this memory stores synthetic data following (6.3) and hence it is called composite memory. Therefore only the aspects different to common rehearsal-based approaches are emphasized.

First the components $\mathbf{c}_i$ and weights w_{ij} are randomly initialized. For a continual learning setup based on supervised image classification, Wiewel and Yang propose to use a uniform random initialization of the components $\mathbf{c}_i$ in the interval $[0, 1]$ while the weights w_{ij} are initialized according to a standard normal distribution. This is followed by a loop of K outer iterations at the beginning of which the DNN is initialized randomly. Similar to DCGM this is intended to approximate the expectation operator in (6.2), which ultimately leads to synthetic data that is useful for training a DNN from any initial parameters. The next step involves T iterations of computing the gradients of a training loss function $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}$ for both a batch of original and synthetic data. Then these gradients are used to minimize the distance $D(\cdot, \cdot)$ between them with respect to the synthetic data, i.e. the components $\mathbf{c}_i$ and weights w_{ij} that are used to compose $\mathbf{x}_j$. In contrast to common training methods for DNNs based on SGD, the gradient here is taken not with respect to the weights of a DNN but rather to its inputs. Depending on the used network architecture, training data and optimizer used for this, only a few iterations I are required until the gradients are matched. Finally, a batch of rehearsal data is sampled from the memory and used to train the parameters for J iterations. The output of this algorithm is a rehearsal memory that has been updated with synthetic data represented according to (6.3).

Algorithm 3: Dataset condensation using CCMCL

```
Input: Rehearsal memory $\mathcal{M}$, task $\mathcal{T}_k$, composite memory $\mathcal{M}_\mathcal{S}$, training batch size $B_T$,
       condensation batch size $B_M$, training learning rate $\gamma_T$, condensation learning rate
       $\gamma_C$, outer iterations $K$, inner iterations $T$, matching iterations $I$, training iterations
       $J$, DNN $f_\theta$
Initialize composite memory $\mathbf{c}_i$ and $w_{ij}$;
for $K$ iterations do
    Reinitialize model $f_\theta$;
    for $T$ iterations do
        Sample from individual classes of task $\mathcal{B}_T \sim \mathcal{T}_k$;
        Sample from composite memory $\mathcal{B}_M \sim \mathcal{M}_\mathcal{S}$;
        Compute gradient $\mathbf{g}_T = \nabla_\theta \sum_{\mathbf{x},y\in\mathcal{B}_T} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$;
        Compute gradient $\mathbf{g}_M = \nabla_\theta \sum_{\mathbf{x},y\in\mathcal{B}_M} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$;
        for $I$ iterations do
            Update composite memory $\mathcal{M}_\mathcal{S} \leftarrow \mathcal{M}_\mathcal{S} - \gamma_C \nabla_\mathcal{S} D(\mathbf{g}_T, \mathbf{g}_M)$;
        end
        Sample from rehearsal memory $\mathcal{B}_B \sim \mathcal{M}$;
        for $J$ iterations do
            Update DNN parameters $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma_T \nabla_\theta \sum_{\mathbf{x},y\in\mathcal{B}_B \bigcup \mathcal{B}_T} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$;
        end
    end
end
Update memory $\mathcal{M} \leftarrow \mathcal{M} \bigcup \mathcal{M}_\mathcal{S}$;
Output: Updated rehearsal memory $\mathcal{M}$
```

6.3. Experiments and Results

For evaluating the CCMCL method outlined in Algorithm 3 its performance is evaluated on four datasets: MNIST, FashionMNIST, SVHN and CIFAR10 which are summarized in section 3.6. It is compared against the baseline of naive rehearsal with a growing memory, DCGM as proposed by Zhao, Mopuri, and Bilen [129], BiC introduced by Wu et al. in [95] and the Upper Bound (UB) given by training on all data simultaneously. Similar to previous chapters only the ICL scenario is evaluated for the same reasons of being the most challenging out of the three scenarios presented and since it is widely used in the literature as a benchmark. But CCMCL can also be applied to the ITL and IDL scenarios without any modifications. The reason for this is that it only relies on differentiating a loss function with respect to the input in order to generate synthetic data. The same holds true for the other methods mentioned above.

6.3.1. Hyperparameters and Architecture

For all experiments presented in the remainder of this chapter the ConvNet architecture is used. Its structure is presented in Table 3.7. It is a small DNN architecture that is especially popular in the field of few-shot learning [50, 60]. Since this field also deals with rather small datasets, the building blocks and structure of the ConvNet are adapted in a way that mitigates problems specific to such few amounts of data.

First, it avoids the use of batch normalization since this layer requires the estimation of statistics over many different batches in order to work properly. Instance normalization as proposed in [49] is used due to its reliance on only individual instances instead of complete batches for normalization. This leads to a more stable and overall better performance of DCGM according to Zhao, Mopuri, and Bilen and is therefore also used for the evaluation of CCMCL. Second, it replaces max with average pooling which leads to a more dense flow of gradients as the pooling operation is differentiable with respect to every input. This results in an overall more stable and better generation of synthetic data as shown in [129]. For a fair comparison, the ConvNet architecture is used for all methods compared in this chapter.

The hyperparameters used for CCMCL as described in Algorithm 3 are summarized in Table 6.1. They are largely adopted and only slightly modified from those used for DCGM by Zhao, Mopuri, and Bilen [129] while only a limited manual optimization was performed on them. This revealed that the subjective visual quality of the generated synthetic data increased with the number of iterations for the outer and inner loop in addition to the batch size used for distillation. During training SGD is used while the generation of synthetic data is performed using the RMSprop optimizer. Because the MNIST and FashionMNIST datasets feature significantly less complex images than SVHN or CIFAR10, only half of the training and outer loop iterations are needed for CCMCL to converge. These reduced numbers are shown in Table 6.1 first while the iterations used for SVHN and CIFAR10 are shown after the forward

Table 6.1.: Hyperparameters used throughout all experiments on CCMCL.

Parameter	**Value**	**Parameter**	**Value**
Training learning rate γ_T	0.01	Matching iterations I	10
Training iterations for each task	500/1000	Training iterations J	1
Condensation learning rate γ_C	0.1	Training batch size B_T	128
Outer iterations K	50/100	Condensation batch size B_M	256
Inner iterations T	10		

slash. All results are reported as an average and standard deviation over five independent runs. No data augmentation or learning rate schedule is used for any of the compared methods.

As mentioned in section 6.2.3, $|\mathcal{M}_{CCMCL}| = P \times (S + Q)$ real numbers need to be stored by CCMCL for Q synthetic S-dimensional examples represented by P components $\mathbf{c}_i$ and $P \times Q$ weights w_{ij}. Since typically $S \gg Q$, the storage requirement of CCMCL is dominated by the components P and only a small overhead is required to store the $P \times Q$ weights. The number of components for CCMCL is therefore chosen as the memory size, i.e. an equal amount of components per class P_{class} are stored such that the complete rehearsal memory is filled. This results in a slightly higher storage requirement for CCMCL when compared with the other methods due to the additional memory that is required to store the $P \times Q$ weights w_{ij} associated with each synthetic example. This overhead is included in Figure 6.3 and also in Table 6.2 and Table 6.3 where it is included in a separate row. The number of stored synthetic examples per class Q_{class} is defined using the hyperparameter U as $Q_{class} = U \times P_{class}$.

In order to determine the optimal U for every dataset, a search over values between two and 16 is performed on the held out validation parts of the datasets. The results represented by the mean and standard deviation over five independent runs are shown in Figure 6.2. Comparing the results for different choices of U reveals that a larger U and correspondingly number of stored synthetic examples per class Q_{class} can lead to a higher accuracy. But a larger U also requires to store weights which increases the storage overhead and can also lead to instabilities in the process of learning the synthetic examples. This can be observed for the more complex datasets SVHN with $U = 8/16$ and for CIFAR10 with $U = 16$. In these cases the accuracy first increases with the memory size but than plateaus are even decreases. The optimal value for U is therefore dependent on the dataset. It should be chosen as large as possible in order to maximize performance. But at the same time it needs to be small enough to avoid instabilities in the generation of synthetic data. Based on the results shown in Figure 6.2, $U = 16/16/4/8$ are chosen for the MNIST/FashionMNIST/SVHN/CIFAR10 datasets.

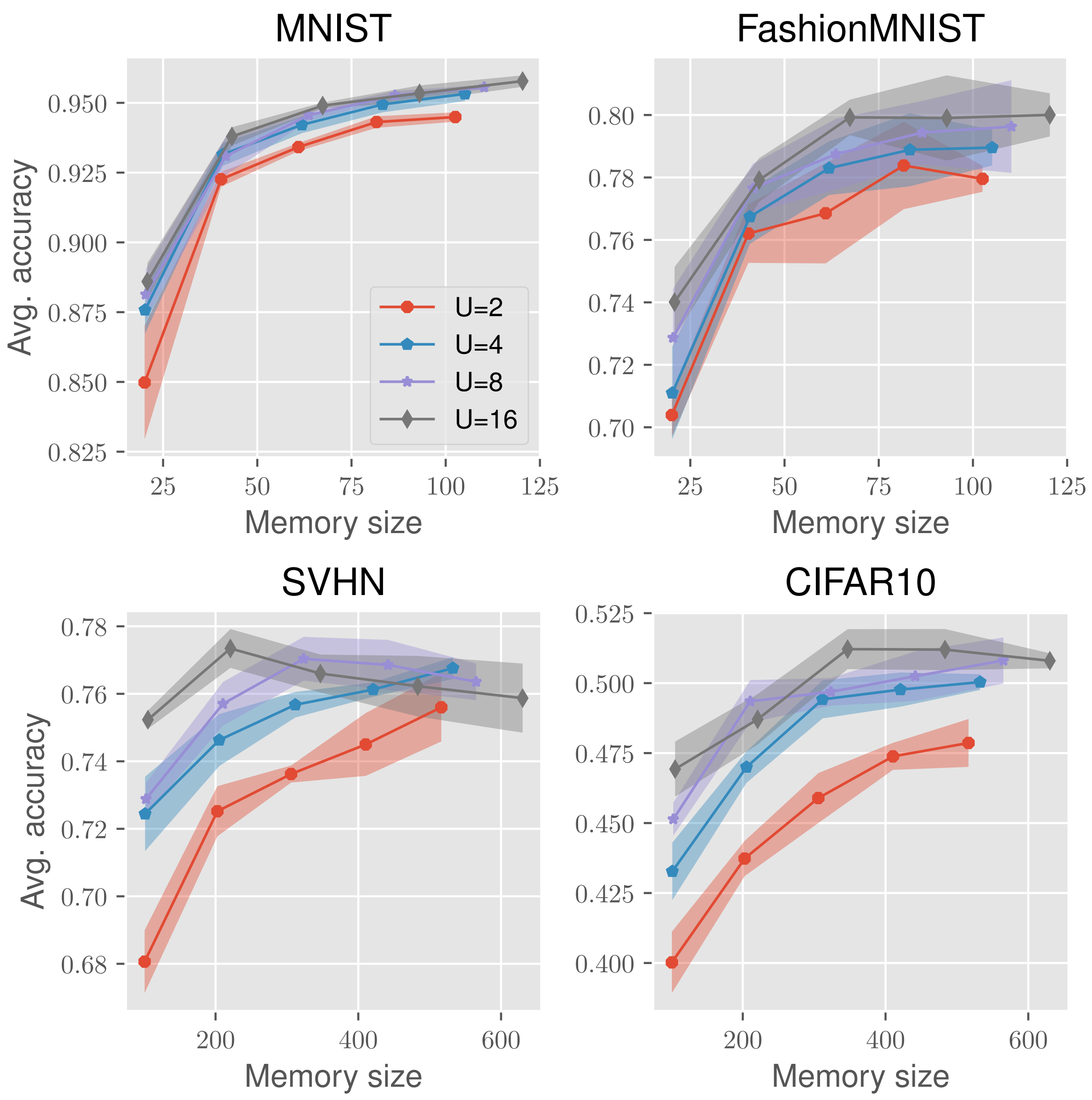


Figure 6.2.: Results of different values for the hyperparameter U for the MNIST, FashionMNIST, SVHN and CIFAR10 datasets.

6.3.2. ICL Results

For all experiments the corresponding datasets are split into five tasks with two classes each in order to form an ICL sequence. This split is performed in such a way that each task contains classes with an increasing class label, i.e. $0, 1 \rightarrow 2, 3 \rightarrow \ldots \rightarrow 8, 9$. Every method is then evaluated on a sequence for five independent runs each with a different random initialization of the DNN parameters, components $\mathbf{c}_i$ and the weights w_{ij}. The average and standard deviation of the final accuracy on the complete test dataset after the final task is trained are reported.

Figure 6.3 shows this average accuracy after the final task is trained over various rehearsal memory sizes measured in number of samples. Since CCMCL stores not only components c_{ij} but also additional weights w_{ij}, its required memory size is not an integer and slightly higher than those of the other methods. Given that the generated synthetic data for both DCGM and CCMCL is intended to capture a training dataset in a small set of synthetic examples, all experiments in the following are performed with comparatively small rehearsal memories. Ultimately it is desirable for all memory-based methods for continual learning to achieve the best performance with a minimum of stored data. This allows for learning more tasks with the same storage requirement or reducing the required memory size for the same performance. For the rather simple MNIST and FashionMNIST datasets the maximum size of this memory is 20 to 100 images at the end of each ICL sequence. On the more complex SVHN and CIFAR100 datasets its size is increased to a range of 100 to 500. In all experiments an equal amount of examples per class are stored during each task. Compared with the total amount of samples in the training sets, i.e. 60000 for MNIST/FashionMNIST and 50000 for CIFAR10/CIFAR100, only a small fraction of the training data is stored in the rehearsal memories.

Without using rehearsal and assuming the DNN is capable of perfectly discriminating between all classes, the average accuracy after task $\mathcal{T}_5$ is learned would be $A = 0.2$. This is due to the forgetting of all but the most recent task. Using a rehearsal memory prevents forgetting and avoids overfitting to the last learned task.

Comparing the results shown in Figure 6.3, it is evident that there is a correlation between the performance of a method and the rehearsal memory size. But although there is a significant increase in performance when its size is increased, this effect slows down with an increasing rehearsal memory size. This is especially noticeable for the comparatively simple datasets MNIST and FashionMNIST. A possible explanation for this is that most of the information contained in a dataset can be captured with a rather small amount of examples, i.e. basic class features. But capturing the information contained in more varied classes, e.g. those

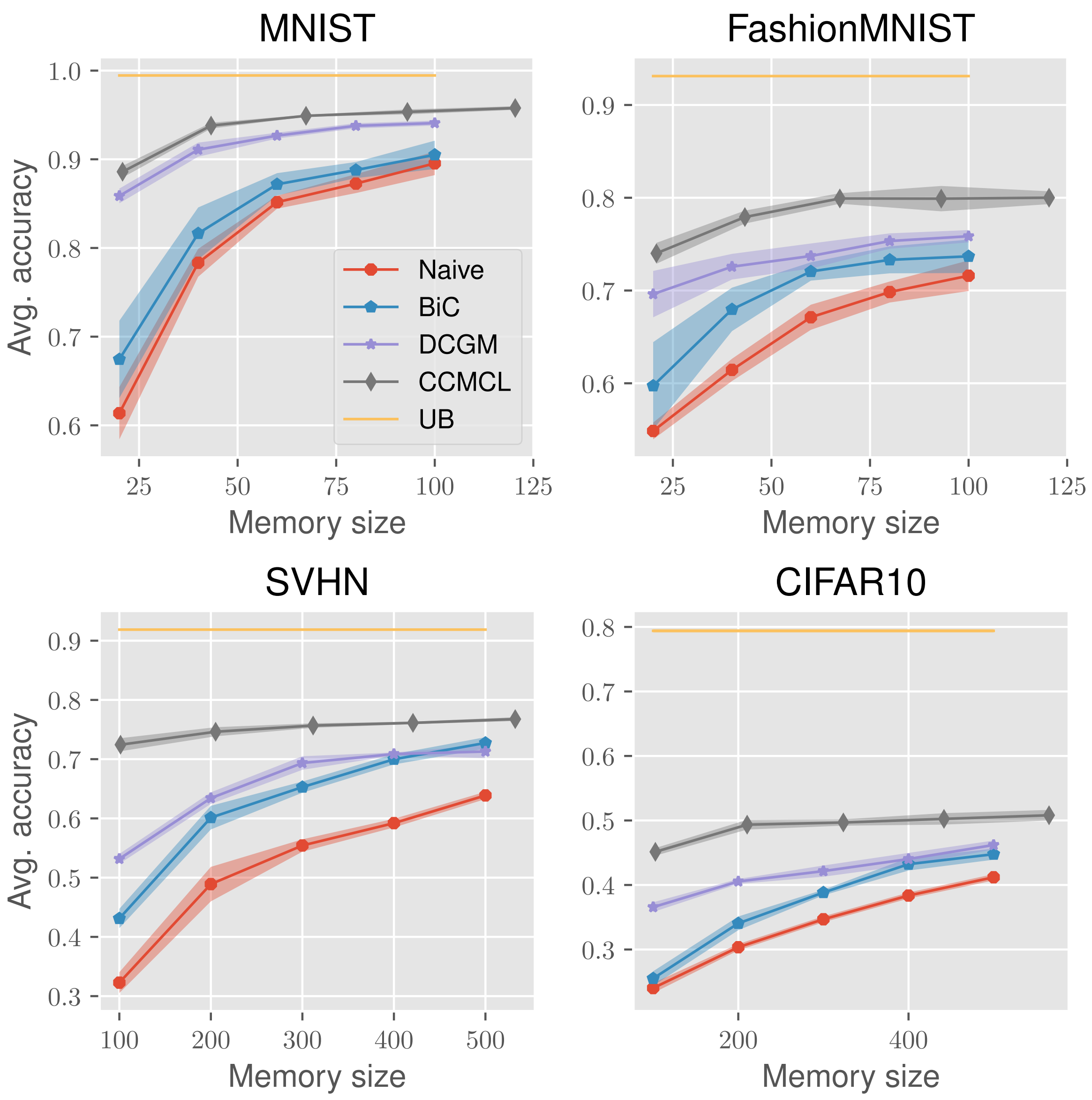


Figure 6.3.: Results of different methods for incremental class learning sequences of the MNIST, FashionMNIST, SVHN and CIFAR10 datasets.

of the SVHN and CIFAR10 datasets, requires much more stored data to effectively prevent catastrophic forgetting. This becomes evident by a comparison with the Upper Bound (UB), which is given by training on all data simultaneously. On MNIST CCMCL and DCGM achieve a result close to the UB already with memory size of 40 and show only a slow increase in performance when the memory sizes is increased. The gap between CCMCL and the UB is much larger for the other datasets. Especially on CIFAR10, the rehearsal methods achieve results that are significantly lower than the UB even for a memory size of 500. But there is still a significant increase in performance when the memory size is increased.

The naive rehearsal baseline, which randomly selects samples to store and random sampling from the memory during training of new tasks, achieves the worst results. But it profits the most from an increase in size of its rehearsal memory. It is outperformed significantly by BiC in all experiments. Especially on the more complex SVHN and CIFAR10 datasets, BiC outperforms naive rehearsal by a large margin that is even increasing slightly with the rehearsal memory size. This again suggests that while most information of a simple training dataset can be captured with a rather small amount of data, learning a task with more complex data requires much more data. When using naive rehearsal an additional problem is overfitting to the memory as many training iterations are performed on a limited amount of data. Measures intended to prevent overfitting, like those proposed with BiC, therefore show a significantly better performance than naive rehearsal. CCMCL on the other hand stores much more synthetic samples in its memory with only a small overhead in storage requirement. It therefore is much less prone to overfitting when compared with naive rehearsal. But although all methods exhibit an increased variance in their results on FashionMNIST compared to all other datasets, the highest variance in the results is observed for BiC on this dataset. It is also worth noting that there is a rather early plateauing of its performance when compared with other methods on this dataset.

The characterizing feature of CCMCL when compared with DCGM is the more memory efficient representation of synthetic data according to (6.3). It is therefore unsurprising that CCMCL outperforms DCGM for a similar memory size. On all datasets CCMCL achieves a significantly higher final accuracy when compared with DCGM. It is worth noting again that the additional storage overhead due to the weights w_{ij} for CCMCL is also included in Figure 6.3.

Table 6.2.: Average accuracy and standard deviation of experiments on MNIST and Fashion-MNIST. *Overhead applies only to CCMCL and describes the additional storage requirement of the weights w_{ij} measured in equivalent sample, e.g. an overhead of 1.0 requires storing the same amount as is required for one image.

Memory size	20	40	60	80	100
Overhead*	0.82	3.27	7.35	13.06	20.41
			MNIST		
Naive rehearsal	61.4	78.3	85.2	87.3	89.5
	±2.9	±1.46	±0.7	±1.1	±1.3
BiC [95]	67.4	81.6	87.2	88.8	90.5
	±4.4	±2.9	±1.3	±0.9	±1.6
DCGM [129]	85.9	91.1	92.7	93.8	94.1
	±0.8	±0.8	±0.3	±0.3	±0.3
CCMCL (Ours)	**88.6**	**93.8**	**94.9**	**95.3**	**95.8**
	±0.7	**±0.3**	**±0.1**	**±0.3**	**±0.2**
Upper Bound (UB)			99.452 ± 0.017		
Memory size	20	40	60	80	100
Overhead*	0.82	3.27	7.35	13.06	20.41
			FashionMNIST		
Naive rehearsal	54.9	61.4	67.1	69.8	71.6
	±0.9	±1.2	±1.4	±1.1	±1.7
BiC [95]	59.7	68.0	72.1	73.3	73.7
	±4.7	±2.3	±1.0	±1.5	±1.8
DCGM [129]	69.6	72.6	73.7	75.8	76.1
	±2.5	±1.4	±1.3	±0.8	±0.7
CCMCL (Ours)	**74.0**	**77.9**	**79.9**	**79.9**	**80.0**
	±1.1	**±0.7**	**±0.6**	**±1.4**	**±0.7**
Upper Bound (UB)			93.116 ± 0.083		

Table 6.3.: Average accuracy and standard deviation of experiments on SVHN and CIFAR10. *Overhead applies only to CCMCL and describes the additional storage requirement of the weights w_{ij} measured in equivalent sample, e.g. an overhead of 1.0 requires storing the same amount as is required for one image.

Memory size	100	200	300	400	500
Overhead*	1.30	5.20	11.72	20.83	32.55
			SVHN		
Naive rehearsal	32.3	48.9	55.4	59.2	63.9
	±1.8	±2.9	±1.0	±0.7	±0.6
BiC [95]	43.1	60.1	65.3	70.0	72.7
	±1.6	±2.0	±0.9	±0.9	±1.0
DCGM [129]	53.2	63.4	69.4	70.9	71.3
	±0.8	±1.0	±1.1	±0.3	±1.1
CCMCL (Ours)	**72.4**	**74.6**	**75.7**	**76.1**	**76.8**
	±1.1	**±0.8**	**±0.4**	**±0.2**	**±0.3**
Upper Bound (UB)			91.868 ± 0.100		
Memory size	100	200	300	400	500
Overhead*	2.60	10.42	23.44	41.67	65.10
			CIFAR10		
Naive rehearsal	24.0	30.4	34.7	38.4	41.2
	±0.8	±0.5	±0.5	±0.5	±0.5
BiC [95]	25.5	34.1	38.8	43.2	44.8
	±1.1	±1.1	±0.4	±1.0	±0.9
DCGM [129]	36.6	40.5	42.1	44.0	46.2
	±0.7	±0.4	±0.9	±0.9	±0.7
CCMCL (Ours)	**45.1**	**49.4**	**49.7**	**50.2**	**50.9**
	±0.6	**±0.8**	**±0.5**	**±0.9**	**±0.8**
Upper Bound (UB)			79.398 ± 0.279		

6.3.3. Qualitative Comparison

In addition to comparing the performance of all analyzed approaches based on the classification accuracy, it is also of interest to compare original data with the learned synthetic data produced by CCMCL and DCGM. This can give insights into whether these two approaches generate not only data that leads to a similar DNN performance when trained upon and in addition if the synthetic data looks subjectively similar to original data. For this, the contents of their memories with a size of 100 is plotted for naive rehearsal, DCGM and CCMCL on all datasets in Figure 6.4.

The first row in Figure 6.4 shows the examples stored by naive rehearsal, which are used as a sample of original data and serve as a reference. For each of the datasets, 10 examples of each class are shown in one row. Although those do not capture all different styles of instances in a class, they give an indication on the visual appearance of original data. It is immediately evident that the data of MNIST and FashionMNIST is rather simple compared to SVHN and CIFAR10 since it is not only gray scale but also seems to feature less variation even considering the very limited sample size shown.

Comparing to this the synthetic data learned by DCGM in the second row, a lack of contrast and a significant noise can be observed. This is especially noticeable for the gray scale images where the background of original data is completely black while the synthetic data shows not only a brighter background but also a non uniform one. The lack of contrast can be explained by the pre-processing applied before plotting, which normalizes the pixel of each image into to the interval $[0, 1]$. Since there is no mechanism that prevents DCGM from learning pixel values outside of this range, the background pixel are close to but not exactly zero. The noise also results from this and is inherent to the process of learning synthetic data. The severity of this phenomenon depends partly on the DNN architecture used for learning the synthetic data and is also observed by Zhao, Mopuri, and Bilen [129]. A similarly reduced sharpness and increased noise can be observed for the SVHN and CIFAR10 datasets when comparing the synthetic data of DCGM with original data. Nonetheless, there is a certain similarity between the synthetic and original data showing similar features. It is also interesting to observe that the synthetic data learned on SVHN only shows a central digit whereas the original data shows multiple digits in many instances. This is due to the fact that labels for SVHN are derived from the center digit, which therefore should be the characteristic feature used by a DNN to discriminate between classes. Hence the synthetic data is learned in such a way by DCGM that

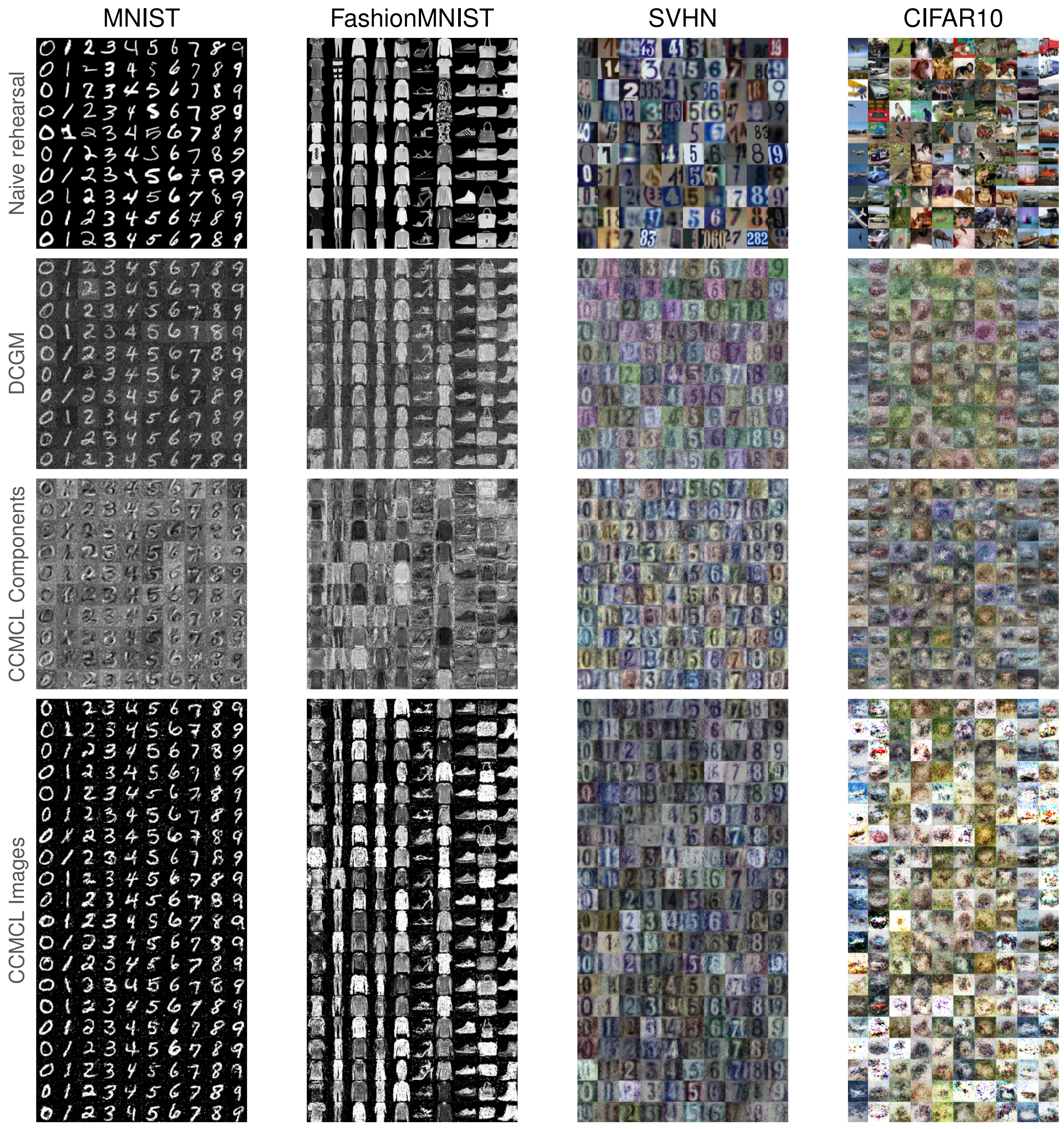


Figure 6.4.: Examples of images stored by naive rehearsal, DCGM and CCMCL. For the latter components and reconstructed composite images are shown in the last two rows for $U = 2$.

only this feature is present. On CIFAR10 it is difficult to identify finer details in the synthetic data but the overall shape and color palette of individual classes can be identified.

For a comparison of the synthetic data learned by CCMCL to those generated by DCGM and original data, the components $\mathbf{c}_i$ are shown in the third and images generated from them using weights w_{ij} according to (6.3) are shown in the fourth row. Each image is comprised of ten weighted components and a total of 200 composite images.

Similar to the synthetic data learned by DCGM, these components show a significant level of noise. At least on the gray scale datasets, its intensity appears to be even higher. On the SVHN and CIFAR10 datasets there is also visible noise but subjectively the overall contrast seems to be higher than in the data shown in the second row. Another interesting observation is that the components seem to be partly or completely inverted. This is especially noticeable on the gray scale datasets as here the original data shows digits with a high intensity while some components are partly or even completely lower in intensity than the background. In combination with the weighting this allows CCMCL to subtract or add features when combining the components into an image and therefore increases the variety of images that can be stored.

An exemplary illustration of this process is shown in Figure 6.5 for the fifth stored example of the first class, i.e. T-shirt, from the FashionMNIST dataset. While the top row shows all ten learned components and the corresponding weight for one synthetic example of the T-shirt class, the bottom row shows the cumulative image according to (6.3) from just one component on the left to all ten on the right. Comparing these components to the final image, it is obvious that none of them individually shows a close resemblance to the final image. But through the process of weighting and combining them, the components accumulate to the final image. It is also worth noting that beginning with the second component noise in the background is significantly suppressed. This it due to the sigmoid function applied pixel-wise to the weighted components. Due to its asymptotically flat behavior, small variations in the intensity are drastically reduced. More examples from other datasets are shown in Figure 6.6, Figure 6.7 and Figure 6.8.

When comparing the synthetic data learned by CCMCL directly with those from DCGM, a much higher contrast and lower noise is observed. But compared with original data, there is still noticeable noise and a lack of sharpness present that degrade the perceived quality of it. In addition to this, the synthetic data generated by CCMCL shows more variety since it can store twice the amount of images at the cost of a small overhead. For the MNIST, FashionMNIST and SVHN datasets, it generates not only data that is better suited for rehearsal but also data

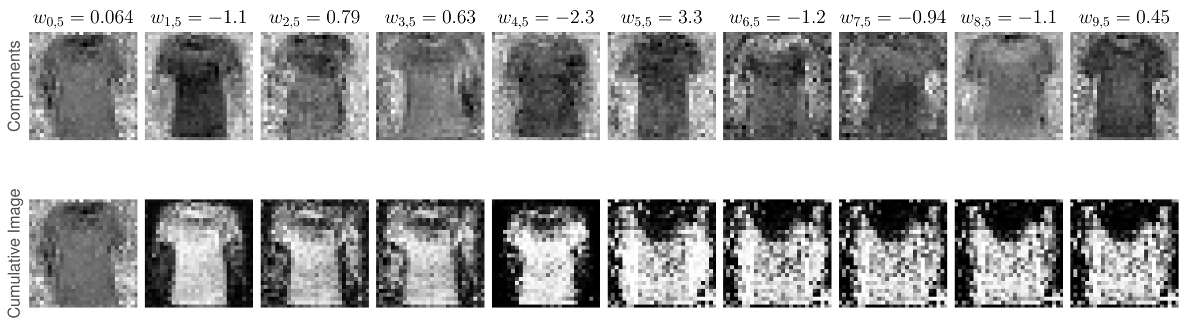


Figure 6.5.: Example of composition from the fifth image of the first class of the FashionMNIST dataset with top row showing components with corresponding weight in title and cumulative image according to (6.3) in bottom row.

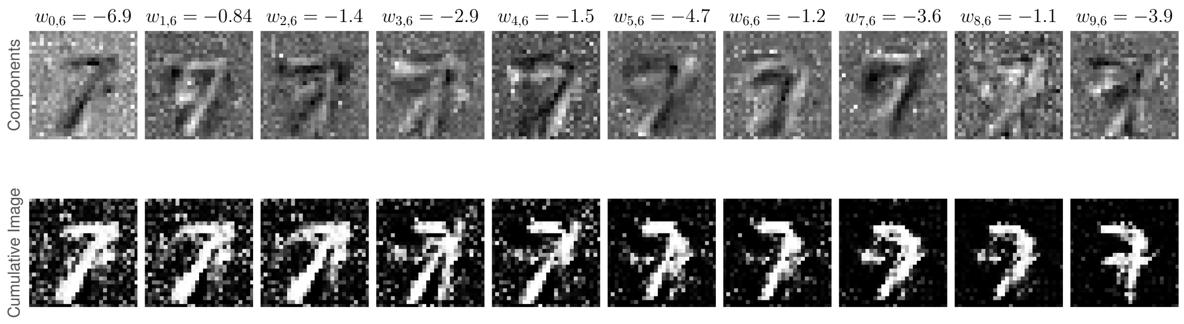


Figure 6.6.: Example of composition for the digit 7 of the MNIST dataset.

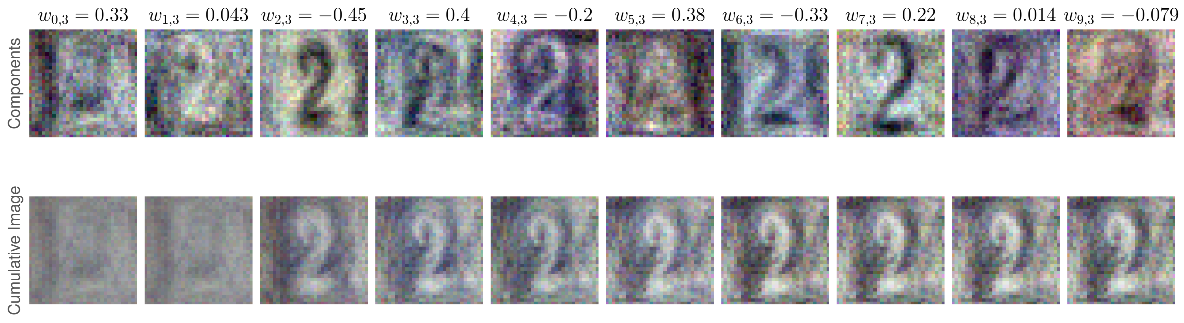


Figure 6.7.: Example of composition for the digit 2 of the SVHN dataset.

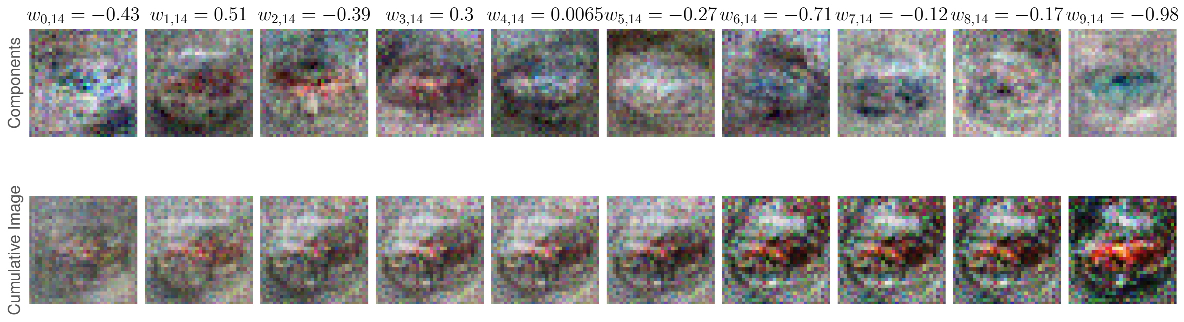


Figure 6.8.: Example of composition for the class car of theCIFAR10 dataset.

that is subjectively more similar to original data. Interestingly, this does not transfer directly to the CIFAR10 dataset. While on this experiment CCMCL outperforms all other compared methods by a wide margin in the objective comparison, the synthetic data generated by it in some instances suffers from a very high brightness and saturation. These two effects lead to pixel artifacts which are especially noticeable for images with a white background.

6.4. Conclusion

Related work has shown the potential of synthetic data for applications like NAS [129] and even to a limited extend for continual learning itself. In this chapter an improvement of such methods is motivated by the introductory research question:

How can synthetic data be learned for rehearsal-based continual learning while reducing the redundancy of data stored in memory?

This question arises from the observation that several methods [79, 128, 129], which generate a small set of synthetic examples intended to summarize a much larger dataset, learn each example of synthetic data individually and hence store potentially redundant information.

Based on the work of Wiewel and Yang [125] a novel method to improving upon these approaches, called CCMCL, is presented in this chapter. It is based on DCGM proposed by Zhao, Mopuri, and Bilen [129] and uses a simple but effective principal of representing and storing synthetic data as a weighted combination of shared components as described in section 6.2.3. This allows for a more memory efficient rehearsal as components do not need to be learned and stored repeatedly for semantically similar data but can instead be shared and stored only once. The weighting coefficients, which are unique to a synthetic example, can not be shared and instead need to be stored individually.

Experimental are presented in section 6.3. CCMCL is compared against the naive rehearsal baseline and related methods, e.g. BiC and DCGM, on ICL sequences with 5 steps for varying rehearsal memory sizes. Throughout this comparison it outperforms all compared methods by a significant margin on almost all datasets with the only exception being the MNIST dataset, where its performance is only marginally better than that of DCGM. On the SVHN and CIFAR10 datasets a large advantage of CCMCL over all other methods is observed, especially for the smaller memory sizes. For an increasing rehearsal memory size this advantage decreases as all methods including the baseline achieve increasingly similar results but with CCMCL still performing best.

In addition to this quantitative comparison, a qualitative one is presented in section 6.3.3. The data stored in the rehearsal memory at the end of an ICL sequence is compared between naive rehearsal/BiC, which store original data, and DCGM/ CCMCL, which both store synthetic data. In the case of CCMCL shared components are shown in addition to the synthetic data itself. In order to illustrate the process of weighting and summing components used by CCMCL, an example of this process is shown in Figure 6.5. It is also observed that DCGM suffers from a higher level of noise when compared to original data. But although CCMCL generates less noisy data, it still shows noticeable levels of it when compared to original data.

Based on these results it can be concluded that there is potential in using synthetic data for rehearsal-based continual learning through a more efficient way of representing and storing it. Especially for smaller memory sizes and more complex datasets it can improve the performance on rehearsal. The method and experiments presented throughout this chapter show that memory efficiency and consequently performance can indeed be improved in this way and can therefore be interpreted as an answer to the motivating question. Naturally a deeper investigation on what is the most efficient way in representing synthetic data is required. Potential directions of improvement over CCMCL are among others: Sharing components not just between one but many classes, introducing multiple levels of components that are combined on different scales or determining the optimal number of components to learn from a dataset. Due to the limited scope of this thesis these topics are left open for further research.

Chapter 7.

Dirichlet Prior Networks for Continual Learning

Although rehearsal is a simple and effective method for continual learning, its performance depends strongly on how exactly it is performed. Depending on how much and what data is replayed, a shift in the data generating distribution can occur. In this chapter these shifts are studied and a novel method first published by Wiewel, Bartler, and Yang in [133] to mitigate their negative effects is presented. The basic idea of this method is the incorporation of known and controllable rehearsal parameters into the prediction of a DNN. These model the otherwise unaccounted distribution shift by a simple modification of the output layer. Using this approach, the mentioned strong dependence of rehearsal-based continual learning on its implementation is drastically reduced. Experimental results on commonly used datasets show that this method can improve the performance of rehearsal significantly.

7.1. Motivation

Despite its simplicity rehearsal using a small memory that stores training examples from previously learned tasks is an effective and commonly used method for mitigating catastrophic forgetting. Even used entirely on its own it is a strong baseline. Given its effectiveness it is no surprise that methods improve upon it in different ways. These improvements are targeting various aspects of rehearsal, namely data selection, storage efficiency and replay mechanisms.

Approaches that try to improve rehearsal performance by means of improved data selection strategies are, for example, GSS and ESS presented in chapter 5. Although these approaches were originally proposed and evaluated in the setting of online continual learning, the same question of what data to store is also relevant in the offline setting. In the latter, however, data selection is simplified since task boundaries are known and the complete dataset of a task is accessible at once. This allows for simple heuristics like uniform random selection or Herding [29] but also more complex approaches such as bi-level optimization proposed by Borsos, Mutny, and Krause in [97].

Since the performance of rehearsal scales with the amount of data that can be stored in memory, some methods use a compression mechanism to store more data. These approaches exploit redundancy in the original training examples to store compressed representations that can later be reconstructed and then used in rehearsal. Hayes et al., for example, propose to use VQ [3] for compressing and storing learned latent representations of inputs in [103]. A similar approach that also stores learned latent representations but without an additional VQ is proposed by Pellegrini et al. in [106]. Related but distinctly different from this are methods that use synthetic or learned data in order to store more information about previous tasks like HAL [114] or DCGM [129] and our method CCMCL presented in chapter 6. Rather than compressing original training examples, entirely synthetic ones are learned in order to capture as much information as possible about the original dataset.

Lastly, there is the aspect of how stored data is used during training on new tasks. While most methods randomly sample examples from the rehearsal memory and mix it with data of the current task, the exact procedure used for sampling and mixing can have a significant influence on the final performance. Especially the ratio of rehearsal and new data in each mini batch needs to be controlled in order to avoid distribution shifts and performance degradation. A study of these effects in the context of ICL for image classification is presented by Wu et al. in [95], where the authors find empirical evidence that the last layer of a DNN is biased towards the most recently learned classes. As a remedy for this the authors propose an additional correction layer that is trained on a balanced validation dataset stored separately and never used during rehearsal. Similarly, Jodelet, Liu, and Murata [116] analyze different strategies like oversampling of the rehearsal memory or loss scaling to overcome the negative effects of the imbalance between rehearsal and new data in ICL. The authors further propose to use a balanced softmax activation function introduced initially by Ren et al. in [108] for the final layer of a DNN resulting in their method Balanced Softmax Incremental Learning (BSIL) and its meta-learning variant Meta Balanced Softmax Incremental Learning (MBSIL). Hou et al. [87] also conclude that the last layer of a DNN used for ICL in image classification is biased towards the most recently learned classes and propose their method Learning a Unified Classifier Incrementally via Rebalancing (LUCIR) that is based on cosine similarity classification in conjunction with fine tuning on a balanced validation dataset. Lastly, our approach for localizing catastrophic forgetting presented in chapter 4 also indicates an increased contribution of the last layers in a DNN to catastrophic forgetting.

Parameters like the ratio of new and stored data in each mini batch and the algorithm used for selecting data from the memory for rehearsal are known in advance and can be controlled during continual learning. In conjunction with the empirically found evidence for the importance of avoiding biases introduced by rehearsal in the final layer of a DNN we therefore try to answer the following research question:

What types of distribution shifts can be introduced by rehearsal and how can their negative effects be avoided?

In this chapter, we first analyze different types of distribution shifts due to rehearsal with a small memory in section 7.2.1. We then proceed with introducing a novel method for modeling prior knowledge about rehearsal parameters using a Dirichlet Prior Network (DPN) in sections 7.2.2 and 7.2.3. Experimental results on common benchmark datasets are presented and discussed in section 7.3. Afterwards the chapter is concluded by a brief summary in section 7.4.

7.2. Method

Our method is based on modeling prior knowledge about the process of rehearsal such that the negative effects of using a small memory, i.e. distribution shift, are reduced. It uses three components to form a prediction for one input image: Statistics of the current task and rehearsal memory, the mixing ratios of the rehearsal process and the predictions of a DNN given an input image. The distribution shifts due to rehearsal are first analyzed in section 7.2.1. To combine these three components the theoretical framework of DPNs, which is described in section 7.2.2, is used. Finally, all three components are combined into a novel method for continual learning in section 7.2.3. Following most rehearsal-based methods, knowledge distillation is used to transfer previously learned knowledge to the current task. Similar to the previous chapters, we focus on ICL for supervised image classification in order to compare our approach to related work.

7.2.1. Distribution Shifts due to Rehearsal

First a clear definition of what is meant by a distribution shift in the context of rehearsal for continual learning is required. A typical distribution shift in the context of machine learning arises when training data comes from a distribution that is different from that of test data [26, 28, 33, 100].

In this case, the distribution $p(\mathbf{x}, \mathbf{y})$ of the data generating process is not accurately represented by the training dataset. Given a set of samples, e.g. a task $\mathcal{T}_k$ according to Definition 1, selecting a sample randomly from it can be viewed as sampling from the conditional distribution $p(\mathbf{x}, \mathbf{y}|\mathcal{T}_k)$. If not enough samples are available in the training dataset, $p(\mathbf{x}, \mathbf{y}|\mathcal{T}_k)$ might differ from $p(\mathbf{x}, \mathbf{y})$ and hence there will be a distribution shift between training and testing. Especially for long-tailed distributions characterized by regions of considerable probability density far away from their central part, a large number of samples might be needed to accurately capture $p(\mathbf{x}, \mathbf{y})$ completely.

But while this type of shift poses an important challenge to machine learning in general, it is not what we want to focus on in this chapter. We assume that there is no such distribution shift between training and testing data in the whole sequence of tasks during ICL. Instead, we focus solely on the distribution shifts caused by rehearsal with a memory that is limited in size. Note that since rehearsal is only applied during training, the testing data is unaffected and a distribution shift can only occur for the training data.

For this, we remind ourselves what the goal of continual learning according to Definition 3 is: We want to find a sequence of optimal parameters for a DNN which minimize the chosen loss function on a sequence of M tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_M$ while only the dataset associated with the most recent task is accessible. A naive solution to this problem is storing all previously encountered data followed by joint training, possibly with repeated re-initialization of all weights, on all data for every task. While this approach does solve the continual learning problem and can be considered an upper bound, it does not scale due to an ever increasing computational and memory complexity. But despite its lacking scalability, this approach does not introduce any distribution shift since all training data are stored in the rehearsal memory and can be mixed in the right proportion with new training data. The corresponding distribution

$p(\mathbf{x}, \mathbf{y}|\mathcal{T}_{1:k}) = p(\mathbf{x}, \mathbf{y}|\mathcal{T}_1 \cup \mathcal{T}_2 \cup \ldots \cup \mathcal{T}_k)$ of training data resulting from this approach is therefore the reference for all distribution shifts discussed in the following.

Selecting Samples to Store In contrast to the solution presented above, typically not all training data can be stored during ICL but only a subset $\mathcal{M}_k \subset \mathcal{T}_k$. This leads to the problem of deciding which data to store in the rehearsal memory such that

$$p(\mathbf{x}, \mathbf{y}|\mathcal{M}_k) = p(\mathbf{x}|\mathbf{y}, \mathcal{M}_k)p(\mathbf{y}|\mathcal{M}_k) \tag{7.1}$$

matches

$$p(\mathbf{x}, \mathbf{y}|\mathcal{T}_k) = p(\mathbf{x}|\mathbf{y}, \mathcal{T}_k)p(\mathbf{y}|\mathcal{T}_k) \tag{7.2}$$

as closely as possible. In the case of ICL for supervised image classification it is known that the labels $\mathbf{y}$ follow a categorical distribution

$$p(\mathbf{y}|\boldsymbol{\mu}) = \prod_{c=1}^{C} \mu_c^{y_c}, \tag{7.3}$$

where the parameter vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \ldots, \mu_C]^T$ contains the class probabilities for the C classes with $0 \leq \mu_c \leq 1$, $\sum_{c=1}^{C} \mu_c = 1$ and $\mathbf{y} = [y_1, y_2, \ldots, y_C]^T$ being a one-hot encoded vector. These class probabilities depend on the task $\mathcal{T}_k$ and therefore the notation $p(\mathbf{y}|\mathcal{T}_k)$ is used to make this dependence clear. Hence the parameters of $p(\mathbf{y}|\mathcal{M}_k)$ can easily be matched with those of $p(\mathbf{y}|\mathcal{T}_k)$ when selecting samples to store.

For many benchmark datasets, like those presented in section 3.6, the number of examples per class is exactly or at least approximately the same, i.e. they are balanced datasets. Matching the parameters of these categorical distributions is therefore ensured by selecting an equal number of samples from each class to store in the memory. If this rehearsal buffer is allowed to grow an equal number of samples per class can be stored on every task without the need to remove any previously stored data. A fixed size memory, on the other hand, is usually kept completely filled on every task such that the maximum amount of data is stored right from the beginning of an ICL sequence. This necessitates a slightly more complex memory management algorithm that not only stores new but also removes previously stored data in order to match the class balance. It is therefore not surprising that many methods for continual learning use some mechanisms that ensure an equal number of samples per class are stored in the rehearsal buffer.

Matching the conditional distributions, i.e. $p(\mathbf{x}|\mathbf{y}, \mathcal{M}_k) \approx p(\mathbf{x}|\mathbf{y}, \mathcal{T}_k)$, on the other hand is much more challenging since the input space $\mathcal{X}$ is typically high dimensional and no analytical expression for a distribution of the input data is known. Most methods therefore use an alternative approach for selecting which samples of a class to store in memory. While some use simple heuristics like random selection or Herding [29][1], Aljundi et al. treat the process of sample selection as a constraint reduction problem [83]. Borsos, Mutny, and Krause [97] on the other hand use bi-level optimization in combination with a NTK to select samples for storage in memory. A different approach based on entropy maximization is proposed by Wiewel and Yang in [126] and presented in chapter 5.

Sampling From Rehearsal Memory In addition to the aspects of selecting samples to store in memory, the process of sampling and mixing data from it with data of the current task is also of great importance since an improper mixing ratio or sampling strategy can introduce biases [87, 95, 116]. Suppose that during an ICL on task $\mathcal{T}_k$ enough diverse samples from the previously encountered tasks $\mathcal{T}_{1:k-1}$ are stored in the rehearsal memory $\mathcal{M}_{1:k-1}$ and therefore $p(\mathbf{x}, \mathbf{y}|\mathcal{M}_{1:k-1}) \approx p(\mathbf{x}, \mathbf{y}|\mathcal{T}_{1:k-1})$ holds. Sampling a mini batch with data from both the rehearsal memory $\mathcal{M}_{1:k-1}$ and the new task $\mathcal{T}_k$ can then be interpreted as observing samples from

$$p(\mathbf{x}, \mathbf{y}|\mathcal{T}_{k^\star}) = p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{k^\star})p(\mathbf{y}|\mathcal{T}_{k^\star}), \tag{7.4}$$

where $\mathcal{T}_{k^\star}$ is an effective dataset consisting of samples from the rehearsal memory $M_{1:k-1}$ and the new task T_k. From this dataset, samples are selected uniformly at random to form a mini batch for the continual learning task $\mathcal{T}_k$. The size and composition of $\mathcal{T}_{k^\star}$ depend on the fraction $r \in [0, 1]$ of data selected uniformly at random from the rehearsal memory $M_{1:k-1}$ and the complementary fraction $s = 1 - r$ of data from task $\mathcal{T}_k$ in each mini batch. The parameters r and s are user chosen and have an impact to the class imbalance in $\mathcal{T}_{k^\star}$. Following the same basic idea of avoiding distribution shifts during training on an ICL sequence, $p(\mathbf{x}, \mathbf{y}|\mathcal{T}_{k^\star})$ should match in the ideal case the true distribution of the data generating process

$$p(\mathbf{x}, \mathbf{y}|\mathcal{T}_{1:k}) = p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{1:k})p(\mathbf{y}|\mathcal{T}_{1:k}) \tag{7.5}$$

for all tasks encountered so far as if the model was trained on all tasks jointly. This is considered later as the performance Upper Bound (UB). Here, similar to the problem of selecting samples

[1]Herding iteratively selects a subset of data such that the moments of this subset closely match those of the complete set. In ICaRL, for example, this is the mean feature vector obtained before the final layer in a DNN.

to store, it is again difficult to match the conditional distributions $p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{1:k})$ and $p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{k^\star})$ for the same reasons mentioned before. In the case of ICL for supervised image classification, the marginals $p(\mathbf{y}|\mathcal{T}_{1:k})$ and $p(\mathbf{y}|\mathcal{T}_{k^\star})$ are categorical distributions and have a parameterized analytical expression. Given that $\mathbf{y}$ is a one-hot encoded vector with $\mathbf{y} = [y_1, y_2, \ldots, y_C]^T$, the parameters of $p(\mathbf{y}|\mathcal{T}_{1:k})$ can be estimated by

$$\boldsymbol{\mu}_{\mathcal{T}_{1:k}} = \mathbb{E}_{p(\mathbf{y}|\mathcal{T}_{1:k})}[\mathbf{y}] \approx \frac{\sum_{\mathbf{y}\in\mathcal{T}_{1:k}} \mathbf{y}}{|\mathcal{T}_{1:k}|}, \tag{7.6}$$

which depends only on the complete training data $\mathcal{T}_{1:k}$. The parameters of $p(\mathbf{y}|\mathcal{T}_{k^\star})$

$$\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} = \mathbb{E}_{p(\mathbf{y}|\mathcal{T}_{k^\star})}[\mathbf{y}] = r\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}} + s\boldsymbol{\mu}_{\mathcal{T}_k} \approx [N_1, N_2, \ldots, N_C]^T / |\mathcal{T}_{k^\star}, \tag{7.7}$$

where N_c is the number of examples from class c in $\mathcal{T}_{k^\star}$, depend on the contents of $\mathcal{M}_{1:k-1}$, the training data of the current task $\mathcal{T}_k$ and the parameters r and s. This illustrates that the composition of a mini batch and therefore the choice of r and s has a direct influence on the parameters of $p(\mathbf{y}|\mathcal{T}_{k^\star})$. The distributions $p(\mathbf{y}|\mathcal{T}_{k^\star})$ and $p(\mathbf{y}|\mathcal{T}_{1:k})$ can therefore be matched simply by adjusting the rehearsal parameters r and s accordingly for every task. Although r and s also influence the conditional distribution $p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{k^\star})$, it can not be matched to $p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{1:k})$ in such a simple way. The reason for this is the lack of an analytical expression for $p(\mathbf{x}|\mathbf{y}, \mathcal{T}_{k^\star})$. We therefore focus only on matching the analytical form of the priors for $\mathbf{y}$ in this chapter.

Different Rehearsal Strategies There are many different strategies how a mini batch can be constructed during rehearsal. Potentially the simplest is to select samples uniformly at random from the union of the rehearsal memory $\mathcal{M}_{1:k-1}$ with the complete dataset of the current task $\mathcal{T}_k$. Given that $\mathcal{M}_{1:k-1} \bigcap \mathcal{T}_k = \emptyset$ follows $\mathcal{T}_{k^\star} = \mathcal{M}_{1:k-1} \bigcup \mathcal{T}_k$. This is the strategy of No Oversampling (NoOS) and the ratio of rehearsal data in a mini batch is given by $r = |\mathcal{M}_{1:k-1}|/|\mathcal{T}_{k^\star}| = |\mathcal{M}_{1:k-1}|/(|\mathcal{M}_{1:k-1}| + |\mathcal{T}_k|)$. This ratio might change with every task depending on the amount of data in the current task and whether the rehearsal memory is growing or fixed in size. Another common strategy is Buffer Oversampling (BOS), where each mini batch contains an equal amount of data sampled uniformly at random from the rehearsal memory $\mathcal{M}_{1:k-1}$ and the current task $\mathcal{T}_k$. This leads to a mixing ratio of $r = s = 1/2$ which remains constant for every task. Usually this requires oversampling the pretty small rehearsal memory, i.e. the same sample from the rehearsal memory may be selected multiple times, hence its name. Class Oversampling (COS) on the other hand ensures that the probability of observing a particular class in a mini batch is equal for all classes on average. While COS also requires oversampling the buffer similar to BOS, it leads to a mixing ratio r that changes with each task. The reason for this is that during ICL the number of classes stored in the rehearsal memory increases with every task.

Figure 7.1 shows the class probabilities of these three different strategies during an ICL sequence for five tasks with two distinct classes each. This is a typical setup used in the literature on continual learning that arises when a dataset with ten classes is split into five equally sized tasks. The first row (a) shows the class probabilities of the dataset from task $\mathcal{T}_k$ in red while the second row (b) shows in blue the class probabilities of all data stored in a rehearsal memory with a fixed size that is nine times smaller than the training dataset of the current task, i.e. $|\mathcal{M}_{1:k-1}| = |\mathcal{T}_k|/9$. It is further assumed that the all tasks of the ICL sequence feature an equal number of examples per class and hence this class balance is maintained when filling the rehearsal memory. Rows three to five then show the class probabilities of $p(\mathbf{y}|\mathcal{T}_{k^\star})$ for the three rehearsal strategies NoOS, BOS and COS introduced above. In these rows the color red shows the component of class probabilities caused by data of the new task $\mathcal{T}_k$ and blue the component due to data stored in the rehearsal buffer. When comparing those with the class probabilities of a setup where the training data of all tasks can be stored shown as purple in row (f), a significant discrepancy for NoOS and BOS becomes obvious. Given that the class probabilities fully describe a categorical distribution, it can therefore be concluded that these two strategies cause a distribution shift between $p(\mathbf{y}|\mathcal{T}_{k^\star})$ and $p(\mathbf{y}|\mathcal{T}_{1:k})$. It is, however, also obvious that BOS leads to class probabilities that are closer to the ideal case shown in the last row (f). It should therefore induce less biases than NoOS during rehearsal. Only for COS are the class probabilities identical to those of a training where all data can be stored. Based on our reasoning it should therefore result in the best performance of all these strategies which we will verify in section 7.3.

But despite this theoretical advantage of COS over BOS, in practice it has some serious disadvantages. When using BOS a constant amount of data from the current task $\mathcal{T}_k$ is present in each mini batch. The number of iterations to complete one epoch over the training data of $\mathcal{T}_k$ with fixed size remains constant no matter how many classes are stored in the rehearsal memory. This is in contrast to COS which processes increasingly less data of the current task $\mathcal{T}_k$ in each mini batch as the number of classes stored in the rehearsal memory $\mathcal{M}_{1:k-1}$ increases. In practice this leads to an ever increasing number of iterations that are required to complete one epoch over the training data of the current task. Especially if many classes are stored in the rehearsal buffer and the current task only features a small number of classes, a large amount of data in each mini batch is sampled from the rehearsal memory. Given that it is

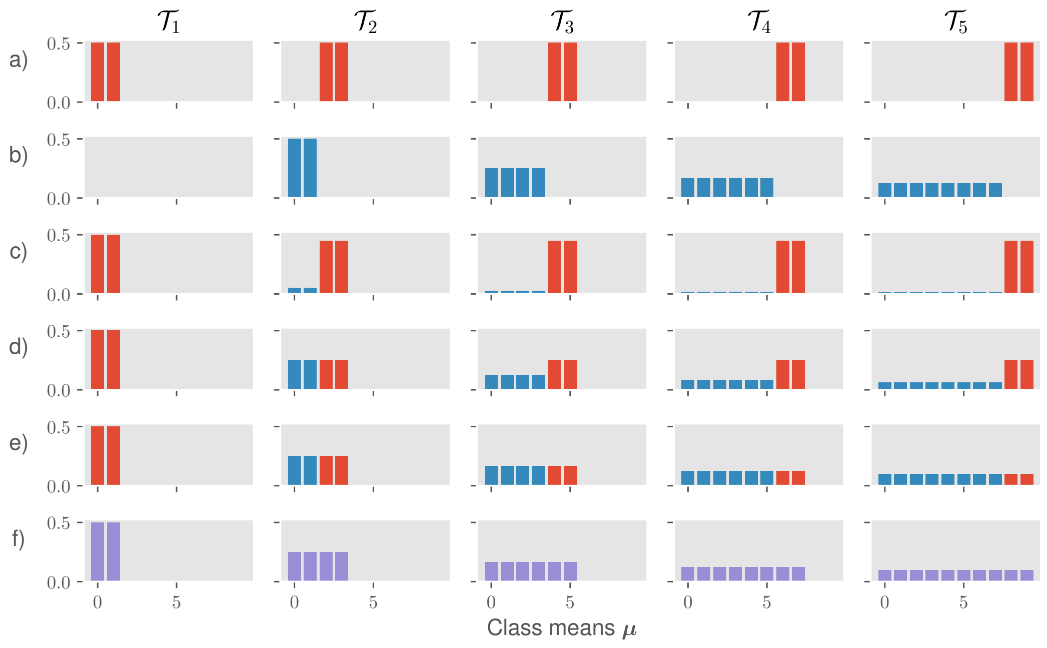


Figure 7.1.: Class probabilities during an ICL sequence for supervised image classification with five tasks each containing two distinct classes. Depicted are : a) $\boldsymbol{\mu}_{\mathcal{T}_k}$, b) $\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}}$, c) $\boldsymbol{\mu}_{\mathcal{T}_k\star}$ using NoOS with $r = 1/10$, d) $\boldsymbol{\mu}_{\mathcal{T}_k\star}$ using BOS, e) $\boldsymbol{\mu}_{\mathcal{T}_k\star}$ using COS and f) $\boldsymbol{\mu}_{\mathcal{T}_{1:k}}$.

generally desirable to use a small rehearsal memory this also increases the risk of overfitting to the stored data significantly.

This means, COS eliminates the discussed distribution shift and leads to better results than BOS but comes not only with an increased computational cost but also a greater risk of overfitting. It would therefore be desirable to find an approach that similarly to COS prevents distribution shifts but also avoids excessive repetitions of the relatively few data stored in the rehearsal memory. As a potential solution to this we propose an approach that uses the prior knowledge about the class probabilities of both $p(\mathbf{y}|\mathcal{M}_{1:k-1})$ and $p(\mathbf{y}|\mathcal{T}_k)$ in addition to the parameters r and s in the prediction of a DNN such that potential distribution shifts are accounted for by the prior knowledge and therefore have reduced impact on the weights learned during training.

7.2.2. Dirichlet Prior Networks

Definition 8: Conjugate Prior

In Bayesian probability theory the prior distribution $p(\boldsymbol{\mu})$ is said to be the conjugate of a likelihood function $p(\mathbf{y}|\boldsymbol{\mu})$ if the posterior

$$p(\boldsymbol{\mu}|\mathbf{y}) = \frac{p(\mathbf{y}|\boldsymbol{\mu})p(\boldsymbol{\mu})}{p(\mathbf{y})} \tag{7.8}$$

is of the same family as the prior. This means that the form of $p(\mathbf{y}|\boldsymbol{\mu})$ and $p(\boldsymbol{\mu}|\mathbf{y})$ is identical.

In the setting of supervised learning DNNs are typically trained to predict a point estimate for the parameters of a distribution $p(\mathbf{y}|\mathbf{x})$ over a target $\mathbf{y} \in \mathcal{Y}$ conditioned on an input $\mathbf{x} \in \mathcal{X}$. In the case of supervised image classification with a one-hot encoding $\mathbf{y} = [y_1, y_2, \dots, y_C]^T$,

$$p(\mathbf{y}|\boldsymbol{\mu}) = \prod_{c=1}^{C} \mu_c^{y_c} \tag{7.9}$$

is a categorical distribution over C different class labels whose associated class probabilities are predicted by a standard DNN $f_\theta(\mathbf{x})$, i.e. $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_C]^T = f_\theta(\mathbf{x})$. The dependence of $\boldsymbol{\mu}$ on $\mathbf{x}$ is omitted for a cleaner notation.

A Dirichlet Prior Network (DPN) directly predicts parameters $\boldsymbol{\alpha}$ of the conjugate prior to the categorical [23, 76 f.], namely the Dirichlet distribution

$$\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x}) = \frac{\Gamma(\alpha_0)}{\prod_{c=1}^{C}\Gamma(\alpha_c)} \prod_{c=1}^{C} \mu_c^{\alpha_c - 1} \tag{7.10}$$

with

$$\alpha_c > 0,\ \alpha_0 = \sum_{c=1}^{C} \alpha_c, \tag{7.11}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_C]^T = g_{\boldsymbol{\theta}}(\mathbf{x})$ are the so called concentration parameters of the Dirichlet distribution and $g_{\boldsymbol{\theta}}(\mathbf{x})$ is the DPN with input $\mathbf{x}$. The sum of all concentration parameters α_0 is defined as the precision and $\Gamma(z) = \int_0^\infty x^{z-1}\mathrm{e}^{-x}\mathrm{d}x$ is the gamma function. Note that $\boldsymbol{\alpha}$ represents the parameters of the Dirichlet prior $\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x})$ while $\boldsymbol{\mu}$ in (7.9) represents the parameters of the categorical distribution $p(\mathbf{y}|\boldsymbol{\mu})$. Again the dependence of $\boldsymbol{\alpha}$ on $\mathbf{x}$ is omitted for a cleaner notation. The concentration parameters completely define the Dirichlet distribution and are predicted by the DPN. A proof that the Dirichlet distribution is indeed the conjugate prior of the categorical in (7.9) is provided in appendix B.

DPNs were initially proposed by Malinin and Gales [71] in the context of predictive uncertainty estimation, where they are intended to model three types of uncertainty in the prediction of a DNN: distributional, data and model uncertainty.

Distributional uncertainty is caused by a mismatch or shift between the distributions of data from training and testing datasets. It is usually not explicitly modeled but rather part of model uncertainty in Bayesian approaches.

Model or *epistemic* uncertainty measures how well a model, e.g. DNN, is matched to given training data, i.e. the uncertainty in estimated model parameters. It can be reduced by increasing the amount of training data and is therefore more pronounced for small datasets.

Data or *aleatoric* uncertainty on the other hand can not be reduced by increasing the size of a training dataset as it is inherent to the data itself. Examples for this are overlapping classes or noise in the ground truth labels.

According to Malinin and Gales, data uncertainty is a *known-unknown* since the DNN is capable of confident predictions after training and can predict whether an input is difficult to classify or not. Distributional uncertainty on the other hand is stated to be an *unknown-unknown* as a confident prediction is impossible since the model is exposed to unseen data under the distribution shift and therefore can only predict with a low confidence. While this short and simple introduction to uncertainty suffices for our purposes, we refer the reader to [71] and [88] for a detailed discussion on uncertainty and training methods for DPNs. In this chapter, we use the DPN in another context, namely to derive a novel solution to mitigate class prior distribution shifts during continual learning.

Making Predictions using DPNs Given parameters $\boldsymbol{\alpha}$, the distribution over class labels according to Malinin and Gales [71] can be determined through marginalization as

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} = \prod_{c=1}^{C} \left(\frac{\alpha_c}{\alpha_0}\right)^{y_c} . \tag{7.12}$$

A detailed derivation of this equivalence is provided in Appendix B. Note that in this equation the dependence of $\boldsymbol{\alpha}$ on $\mathbf{x}$ is again omitted. Comparing this with the categorical distribution obtained using a standard DNN shown in (7.9) reveals that α_c/α_0 is equivalent to the class probability μ_c of the c-th class. Importantly, these are unaffected by a scaling of $\boldsymbol{\alpha}$ and therefore the information carried by the precision α_0 is lost in the process of marginalization. But unlike the normalized class probabilities μ_c with $0 \leq \mu_c \leq 1$ and $\sum_c \mu_c = 1$, predicted concentration parameters α_c are not normalized and lower bounded, i.e. $0 < \alpha_c$, and therefore offer an additional degree of freedom.

This additional freedom is used by Malinin and Gales to model distributional uncertainty using DPNs. For this, the DPN is trained to predict with a high precision $\alpha_0 = \sum_c \alpha_c$ on In-Distribution (ID) and a low one on Out-Of-Distribution (OOD) data. This training procedure not only requires additional OOD data that is close but still different in distribution from the ID data, but also determines an explicit target precision for ID data as well. For a more detailed discussion about training losses and DPN calibration we refer an interested reader to the original publications [71, 88].

In our case, for a fair comparison with related methods on continual learning, no additional OOD data can be used and therefore the corresponding training methods from [71, 88] can not be applied. But fortunately, if a calibrated precision is not required, a DPN can simply be trained using the commonly used cross entropy loss as we do in this chapter.

Activation Functions Many different activation functions can be used in the construction of a DPN $g_{\boldsymbol{\theta}}(\mathbf{x})$, which is used to predict the parameters $\boldsymbol{\alpha}$. While there are no constraints for hidden layers, the activation function in the final layer of the DPN has to be chosen carefully. In order to fulfill the constraints stated in Eq.7.11, a strictly positive output for each neuron is required. This allows for many popular activation functions like the Softplus or exponential function to be used. Even the frequently used ReLU activation can be used if a small offset is added to its output such that it is strictly positive for all inputs. Using an activation function that is not only lower but also upper bounded like the sigmoid is also possible but leads to an upper bounded maximum precision α_0. To still enable the DPN to predict with a high precision, a scaling is necessary in this case.

Among these activation functions the exponential is of particular theoretical importance. Given an output of the final layer $\mathbf{z} = [z_1, z_2, \dots, z_C]^T$ in a DPN it is defined as $\alpha_c = \mathrm{e}^{z_c}$ with z_c being the output of a neuron corresponding to the c-th class. Using the exponential activation function in combination with (7.12) yields

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha})\mathrm{d}\boldsymbol{\mu} = \prod_{c=1}^{C} \left(\frac{\alpha_c}{\alpha_0}\right)^{y_c} = \prod_{c=1}^{C} \left(\frac{\mathrm{e}^{z_c}}{\sum_j \mathrm{e}^{z_j}}\right)^{y_c} . \tag{7.13}$$

As Malinin and Gales point out, this is equivalent to the prediction of a DNN with Softmax activation, i.e. $\hat{\mu}_c = \mathrm{e}^{z_c} / \sum_j \mathrm{e}^{z_j}$. Such a DNN can therefore be interpreted as a special case of a DPN with exponential activation in its final layer.

7.2.3. Continual Learning with DPNs

As discussed in section 7.2.1 there can be distribution shifts caused by using a small rehearsal memory. These have an effect on the weights of a DNN when not accounted for by appropriate counter measures. Since during each task $\mathcal{T}_k$ the network is trained by sampling from $p(\mathbf{y}|\mathbf{x}, \mathcal{T}_{k^\star})$, a potential shift between it and $p(\mathbf{y}|\mathbf{x}, \mathcal{T}_{1:k})$ is learned as well and may lead to biases in the prediction.

Wiewel, Bartler, and Yang therefore propose for their method Dirichlet Prior Networks for Continual Learning (DPNCL) in [133] to use a priori information about such shifts and incorporate this knowledge into the prediction of a DPN. In order to motivate how the prior information can be incorporated into the prediction of a DPN, Bayesian inference is considered. Given a dataset, e.g. the effective dataset $\mathcal{T}_{k^\star}$ from section 7.2.1, containing N_c examples of class c, the likelihood function of the parameter vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_C]^T$ of the categorical distribution is given by

$$p(\mathcal{T}_{k^\star}|\boldsymbol{\mu}) = \prod_{c=1}^{C} \mu_c^{N_c}. \tag{7.14}$$

Note that $\mathcal{T}_{k^\star}$ depends not only on the current task $\mathcal{T}_k$ and the rehearsal memory $\mathcal{M}_{1:k-1}$ but also on the mixing ratios r and s. Before observing this dataset the concentration parameters of a Dirichlet prior $\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha})$ are chosen to be uninformative, i.e. $\boldsymbol{\alpha} = [1, 1, \ldots, 1]^T$. This expresses that nothing about the distribution of $\boldsymbol{\mu}$ is known at this point. After $\mathcal{T}_{k^\star}$ is observed, our knowledge about it is updated and by combining (7.10) with (7.14) and utilizing $\alpha_c = 1$, the result is a posterior

$$p(\boldsymbol{\mu}|\mathcal{T}_{k^\star}, \boldsymbol{\alpha}) \propto p(\mathcal{T}_{k^\star}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) \propto \prod_{c=1}^{C} \mu_c^{N_c+\alpha_c-1} = \prod_{c=1}^{C} \mu_c^{N_c}, \tag{7.15}$$

where N_c is the number of occurrence for class c in task $\mathcal{T}_{k^\star}$. When comparing (7.15) with (7.10) it is evident that this posterior is again a Dirichlet distribution $\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha})$ with updated concentration parameters $\boldsymbol{\alpha} = [N_1 + 1, N_2 + 1, \ldots, N_C + 1]^T$, which can be used as an updated prior for a categorical distribution again. Since the concentration parameters $\boldsymbol{\alpha}$ of the Dirichlet distribution $p(\boldsymbol{\mu}|\mathcal{T}_{k^\star}, \boldsymbol{\alpha})$ contain class counts N_c of the effective dataset $\mathcal{T}_{k^\star}$ from section 7.2.1, it can model the distribution shift introduced by rehearsal. The reason for this is that the vector of class probabilities $\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$ in (7.7) is directly proportional to the class counts, i.e. $\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} = [N_1, N_2, \ldots, N_C]^T / |\mathcal{T}_{k^\star}|$.

In contrast to the DPN $g_{\boldsymbol{\theta}}(\mathbf{x})$, which predicts the parameters $\boldsymbol{\alpha}$ solely based on individual inputs $\mathbf{x}$, the posterior in (7.15) is independent of any input $\mathbf{x}$ and can be viewed as a summary of knowledge about the dataset $\mathcal{T}_{k^\star}$. The main idea of Wiewel, Bartler, and Yang [133] is to combine the posterior $p(\boldsymbol{\mu}|\mathcal{T}_{k^\star}, \boldsymbol{\alpha})$ with the prediction of a DNN $h_{\boldsymbol{\theta}}(\mathbf{x})$. This has the advantage that a prediction not only depends on an input $\mathbf{x}$ but also on knowledge of the observed dataset $\mathcal{T}_{k^\star}$ through the choice of a prior $\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha})$. The distribution shifts due to rehearsal as described in 7.2.1 can therefore be modeled by choosing corresponding concentration parameters $\boldsymbol{\alpha}$. This effectively avoids the need of changing the trainable parameters $\boldsymbol{\theta}$ of $h_{\boldsymbol{\theta}}(\mathbf{x})$ just to follow the distribution shift and the biasing of a DNN $h_{\boldsymbol{\theta}}(\mathbf{x})$ towards the most recently learned classes due to rehearsal [95].

For this, the parameters r and s are used in conjunction with the estimates of $\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}}$ and $\boldsymbol{\mu}_{\mathcal{T}_k}$ as prior information in the prediction. This results in

$$\boldsymbol{\alpha} = \hat{\alpha}\left(r\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}} + s\boldsymbol{\mu}_{\mathcal{T}_k}\right) \odot h_{\boldsymbol{\theta}}(\mathbf{x}) = \hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x}), \tag{7.16}$$

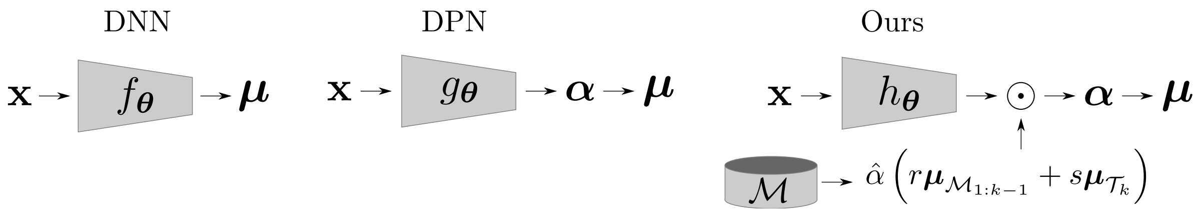

DNN
DPN
Ours
$\mathbf{x} \rightarrow f_{\boldsymbol{\theta}} \rightarrow \boldsymbol{\mu}$
$\mathbf{x} \rightarrow g_{\boldsymbol{\theta}} \rightarrow \boldsymbol{\alpha} \rightarrow \boldsymbol{\mu}$
$\mathbf{x} \rightarrow h_{\boldsymbol{\theta}} \rightarrow \odot \rightarrow \boldsymbol{\alpha} \rightarrow \boldsymbol{\mu}$
$\mathcal{M} \rightarrow \hat{\alpha}\left(r\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}} + s\boldsymbol{\mu}_{\mathcal{T}_k}\right)$

Figure 7.2.: Comparison of different architectures mentioned in this chapter. A DNN directly predicts the class probabilities $\boldsymbol{\mu}$ while a DPN outputs the parameters $\boldsymbol{\alpha}$ of a Dirichlet distribution. The approach proposed by Wiewel, Bartler, and Yang [133], on the other hand, uses prior knowledge of rehearsal parameters, the memory $\mathcal{M}_{1:k-1}$ and task $\mathcal{T}_k$ in the prediction of concentration parameters $\boldsymbol{\alpha}$.

where $\odot$ is the Hadamard product, $\hat{\alpha}$ is a hyperparameter and $h_{\boldsymbol{\theta}}(\mathbf{x})$ is a DNN with trainable parameters $\boldsymbol{\theta}$. A connection to the discussion of Bayesian inference and (7.15) above can be made if the parameter $\hat{\alpha}$ is chosen as the total number of training examples from $\mathcal{T}_{k^\star}$. In this particular case, $\hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$ contains the number of examples for every class in the effective dataset $\mathcal{T}_{k^\star}$. This allows for modifying the known prior information $\hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$ based on individual inputs $\mathbf{x}$ and decouples the known shift in $p(\mathbf{y}|\mathbf{x}, \mathcal{T}_{k^\star})$ during rehearsal from the learned weights of $h_{\boldsymbol{\theta}}$. As a result biases towards the most recently encountered classes are reduced during ICL. A comparison of how the method proposed by Wiewel, Bartler, and Yang differs from a standard DNN $f_{\boldsymbol{\theta}}$ and DPN $g_{\boldsymbol{\theta}}$ is shown in Figure 7.2.

The predictive categorical distribution that results from this modification is given by

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} = \prod_{c=1}^{C} \left(\frac{\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x})\right]_c}{\sum_j \left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x})\right]_j} \right)^{y_c} . \tag{7.17}$$

Compared with prediction of a standard DPN shown in (7.12), (7.17) contains a scaling for each output of $h_{\boldsymbol{\theta}}(\mathbf{x})$ that is dependent on the rehearsal parameters and estimated class probabilities of data in the memory and current task. While training on an ICL sequence this accounts for a potential shift in $p(\mathbf{y}|\mathbf{x}, \mathcal{T}_{k^\star})$. During testing, however, no prior information is available. In common benchmarks each class is typically equally important and therefore all components of $\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$ are chosen to have an equal magnitude. If only a prediction over class labels is of interest as in our case, the parameter $\hat{\alpha}$ can be chosen as an arbitrary positive constant since it has no influence on the predictive distribution (7.17). Similar to a standard DNN, the common cross entropy loss

$$\mathcal{L}_{CE}(\boldsymbol{\theta},\mathcal{T}) = \frac{-1}{|\mathcal{T}|}\sum_{\mathbf{x},\mathbf{y}\in\mathcal{T}}\sum_{c=1}^{C} y_c \ln \frac{\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x})\right]_c}{\sum_{j=1}^{C}\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x})\right]_j} \tag{7.18}$$

is used during training.

As an activation function of the last layer in $h_{\boldsymbol{\theta}}(\mathbf{x})$, Wiewel, Bartler, and Yang propose to use the sigmoid function, i.e. $\alpha_c = \sigma(z_c) = 1/(1+\mathrm{e}^{-z_c})$, as it is not only lower but also upper bounded. Since it approaches zero in the limit as its input goes to negative infinity, it satisfies the constraints on $\boldsymbol{\alpha}$ stated in (7.11). The upper bound of one, which is achieved in limit as the input approaches infinity, ensures that the maximum precision is limited by the prior knowledge encoded in $\hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$. Additionally, it also prevents the model from predicting for a class by simply outputting a very large activation for that particular class. Instead it requires not only predicting a large activation for this class but also a low one for all other classes.

7.2.4. Knowledge Distillation for DPNCL

Many other methods for continual learning that use a rehearsal memory also use knowledge distillation as proposed by Hinton, Vinyals, and Dean in [39]. Therefore it is also used in DPNCL as an additional component that is added to the loss in (7.18). For this, a copy of the weights learned during task $\mathcal{T}_{k-1}$ is saved as $\boldsymbol{\theta}_{Dist}$ and used to predict distillation targets $\boldsymbol{\alpha}_{Dist}$ when task $\mathcal{T}_k$ is trained. The standard formulation of knowledge distillation for classification uses the categorical cross entropy loss

$$\mathcal{L}_{Dist}(\boldsymbol{\theta}_{Dist},\boldsymbol{\theta},\mathcal{T},T) = \frac{-1}{|\mathcal{T}|}\sum_{\mathbf{x}\in\mathcal{T}}\sum_{i=1}^{C} \frac{\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot \sigma(\mathbf{z}_{Dist}/T)\right]_i}{\sum_{j=1}^{C}\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot \sigma(\mathbf{z}_{Dist}/T)\right]_i} \ln \frac{\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot \sigma(\mathbf{z}/T)\right]_i}{\sum_{j=1}^{C}\left[\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot \sigma(\mathbf{z}/T)\right]_j}, \tag{7.19}$$

where T is called the distillation temperature, $\sigma(.)$ is the sigmoid activation function and $\mathbf{z}/\mathbf{z}_{Dist}$ is the output of the final layer of $h_{\boldsymbol{\theta}}(\mathbf{x})/h_{\boldsymbol{\theta}_{Dist}}(\mathbf{x})$. Note that since $h_{\boldsymbol{\theta}_{Dist}}$ was only trained on the classes encountered in tasks $\mathcal{T}_{1:k-1}$, the distillation loss on task $\mathcal{T}_k$ is only computed for the output neurons corresponding to those previously learned classes.

The distillation loss in (7.19) is independent of $\hat{\alpha}$ which prevents it from being distilled into the currently trained model. This means that the prior information encoded in $\hat{\alpha}$ would be lost if

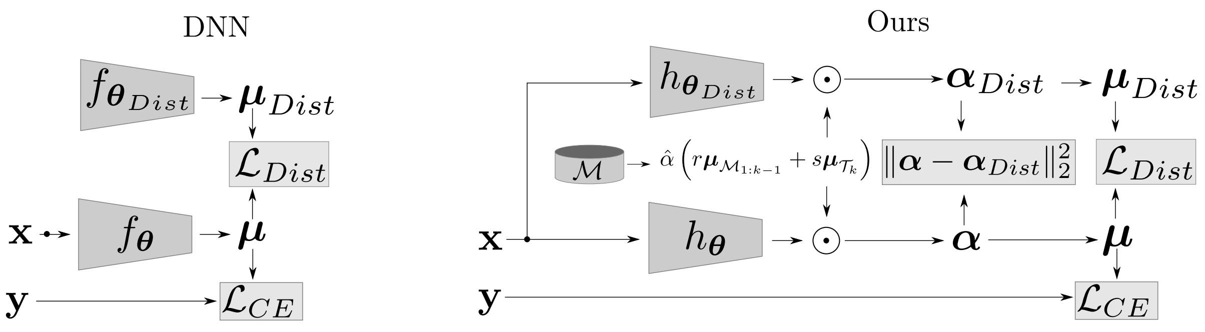


Figure 7.3.: Comparison between standard knowledge distillation and DPNCL.

only the standard knowledge distillation from [39] would be used. In order to avoid this, Wiewel, Bartler, and Yang propose to add the squared euclidean distance between the concentration parameters predicted by the distillation model α_{Dist} and α to $\mathcal{L}_{Dist}$. A comparison between standard knowledge distillation and the one used by DPNCL is shown in Figure 7.3.

The complete loss used for training in DPNCL is therefore given by

$$\mathcal{L}(\theta_{Dist}, \theta, \mathcal{T}, T) = \mathcal{L}_{CE}(\theta, \mathcal{T}) + T \cdot \left[\mathcal{L}_{Dist}(\theta_{Dist}, \theta, \mathcal{T}, T) + \sum_{\mathbf{x} \in \mathcal{T}} \frac{\|\alpha(\mathbf{x}) - \alpha_{Dist}(\mathbf{x})\|_2^2}{|\mathcal{T}|} \right], \tag{7.20}$$

where $\alpha(\mathbf{x})/\alpha_{Dist}(\mathbf{x})$ are the concentration parameters predicted on the input image $\mathbf{x}$ by $h_\theta/h_{\theta_{Dist}}$. Algorithm 4 summarizes the complete DPNCL method in pseudo code.

7.3. Experiments and Results

This section covers experiments that not only compare the method proposed by Wiewel, Bartler, and Yang to related work in section 7.3.1 but also verifies the motivation of it through comparisons to baselines in section 7.3.2. Similar to the chapters before, the experiments are restricted to the ICL scenario of continual learning in order to allow for a fair comparison with related methods. For this, the CIFAR100 and ImageNet datasets discussed in section 3.6 form the basis from which sequences with five or ten incremental steps are constructed. While the complete CIFAR100 dataset is used for this, only a subset of 100 classes selected randomly from the ImageNet dataset are used. This subset is referred to as ImageNet-100 and is build in accordance to the many other publications on continual learning [58, 95, 99, 109, 116, 120]. The selection of those 100 classes is performed using the Numpy [102] function `random.shuffle` with the seed 1993.

Algorithm 4: Pseudo code of DPNCL

```
Input: Number of tasks $N_T$, Number of samples per task to $N_{Store}$, rehearsal mixing
       ratios $r = 1 - s$, hyperparameter $\hat{\alpha}$, batch size $N_B$, learning rate $\gamma$, distillation
       temperature $T$
1  Randomly initialize weights $\boldsymbol{\theta}$ of $h_{\boldsymbol{\theta}}$;
2  Initialize empty rehearsal memory $\mathcal{M}$;
3  for $1 \leq k \leq N_T$ do
4      Determine $\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} = r\boldsymbol{\mu}_{\mathcal{M}_{1:k-1}} + s\boldsymbol{\mu}_{\mathcal{T}_k}$;
5      while Not converged do
6          Sample mini batch $\mathcal{B} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N_B}$ from $\mathcal{T}_{k^\star}$;
7          Predict $\boldsymbol{\alpha}_i = \hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta}}(\mathbf{x}_i)$;
8          Compute loss $\mathcal{L}_{CE}(\boldsymbol{\theta}, \mathcal{B})$ given in (7.18);
9          if $k > 1$ then
10             Predict $\boldsymbol{\alpha}_{i,Dist} = \hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}} \odot h_{\boldsymbol{\theta_{Dist}}}(\mathbf{x}_i)$;
11             Compute $\mathcal{L}_{Dist}(\boldsymbol{\theta}_{Dist}, \boldsymbol{\theta}, \mathcal{B}, T)$ given in (7.19);
12             Loss $\mathcal{L}(\boldsymbol{\theta}_{Dist}, \boldsymbol{\theta}, \mathcal{B}, T) =$
                 $\mathcal{L}_{CE}(\boldsymbol{\theta}, \mathcal{B}) + T \cdot \left[\mathcal{L}_{Dist}(\boldsymbol{\theta}_{Dist}, \boldsymbol{\theta}, \mathcal{B}, T) + \sum_{\mathbf{x} \in \mathcal{B}} \frac{\|\boldsymbol{\alpha}(\mathbf{x}) - \boldsymbol{\alpha}_{Dist}(\mathbf{x})\|_2^2}{|\mathcal{B}|}\right]$;
13         else
14             Loss $\mathcal{L}(\boldsymbol{\theta}, \mathcal{B}) = \mathcal{L}_{CE}(\boldsymbol{\theta}, \mathcal{B})$;
15         end
16         Update weights $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma\nabla_{\boldsymbol{\theta}}\mathcal{L}$;
17     end
18     Select $N_{store}$ samples randomly from $\mathcal{T}_k$ and store them in $\mathcal{M}$;
19     Save weights for distillation $\boldsymbol{\theta_{Dist}} \leftarrow \boldsymbol{\theta}$;
20 end
```

For constructing ICL sequences, the datasets are split in two subsets containing exactly half of all classes. While the first half is used as a base task, the remaining half is split into five or ten subsets again. These are then used as the incremental steps, where during each step only data of the corresponding subset is available to the model under test. Following related work, the order of classes $0, 1, \ldots, 99$ is shuffled using `random.shuffle` with the seed 1993. This setup with a base task containing a large number of classes is intended to mimic a more realistic continual learning scenario, where a DNN is trained on a large dataset, e.g. ImageNet, and only a small number of new classes is learned with each step.

For the experiments on CIFAR10 and CIFAR100, the ResNet32 architecture detailed in Table 3.6 is used. All experiments on ImageNet-100 are performed with the ResNet18 architecture shown in Table 3.4. These architectures are used since they are commonly used throughout the literature for many different benchmarks and therefore allow for a simple and fair comparison between methods.

The hyperparameters used for training are not optimized extensively and are identical for both datasets used in the experiments. Given that the ImageNet-100 dataset is much more computationally demanding than CIFAR100 due to the much higher resolution images, a small-scale manual hyperparameter optimization was performed only on the latter. Similar to many related methods, we use the SGD optimizer with a batch size of 256 and a momentum of 0.9. The DNNs are trained for 250 on CIFAR100 and 150 epochs on ImageNet-100. During training on CIFAR100 the learning rate follows a schedule that starts at 0.1 and decays the learning rate through dividing it by ten after 70% of the iterations on a task are completed. On ImageNet-100 this schedule is only applied during training of the base task and all subsequent tasks are trained using a learning rate of 0.01. Knowledge distillation as described in section 7.2.3 and shown in Algorithm 4 is used with a temperature of $T = 2.0$. The parameter $\hat{\alpha}$ is selected on each experiment in such a way that $\hat{\alpha}\boldsymbol{\mu}_{\mathcal{T}_{k^\star}}$ is a vector that contains the number of examples per class in the equivalent dataset $\mathcal{T}_{k^\star}$.

Only those data augmentations that are also used by related work are employed. For CIFAR100 this is a random horizontal flipping and padding by four pixel with value zero for each of their RGB components on each side of the images followed by a random crop down to the original image size of 32×32 pixel. On ImageNet-100 a smart image resize function from Tensorflow [42] that prevents aspect ratio distortion is used to create 256×256 pixel images from which random crops of 224×224 pixels are extracted. This is again followed by a random horizontal flipping. During testing only one center crop of each image is used instead of a multiple random ones. Following the literature, all results are reported for a growing rehearsal

Table 7.1.: AIA in % on CIFAR100 and ImageNet-100 with a growing buffer of 20 examples per class for different number of steps.

	CIFAR100		ImageNet-100	
Number of tasks	5	10	5	10
Baseline (NoOS)	29.34	18.89	37.00	25.52
Baseline (BOS)	52.06	43.70	54.51	46.09
Baseline (COS)	55.21	51.14	N/A	N/A
ICaRL [58]	57.17	52.57	65.04	59.53
BiC [95]	59.36	54.20	70.07	64.96
BSIL [116]	62.22	58.32	72.57	68.25
DPNCL (Ours, NoOS) [133]	62.33	56.63	71.29	64.60
LUCIR [87]	63.42	60.18	70.47	69.09
DPNCL (Ours, COS) [133]	63.64	58.29	N/A	N/A
MBSIL [116]	64.11	60.08	72.88	69.26
PODNet [99]	64.83	63.19	75.54	74.33
TPCIL [109]	65.34	**63.58**	**76.27**	**74.81**
DPNCL (Ours, BOS) [133]	**65.74**	62.83	73.91	70.65
Upper Bound (UB)	67.82		79.76	

memory that stores 20 images per class as the average over five independent training runs. Random selection is used as strategy for selecting examples to store in memory.

7.3.1. Comparison to Related Work

The comparison with related work on methods for continual learning is of great interest and therefore presented first. But while there are many methods proposed in the literature, not all share the same experimental setup or scenario and can therefore not be compared directly. Table 7.1 shows many recently proposed methods that are evaluated for the ICL scenario using an identical DNN architecture, i.e. ResNet, and also use a growing buffer that stores 20 examples per class. Additionally, the performance of a baseline that uses naive rehearsal with different types of oversampling is included and serves as a baseline. Naive rehearsal selects uniformly at random which data to store in and replay from its rehearsal memory. The AIA as discussed in section 3.5 is used since it allows for a quick comparison of the performance of a method throughout the whole continual learning sequence with just one scalar.

Beginning with the baseline, it is evident that using NoOS results in poor performance, not only compared with other more advanced methods but also other oversampling strategies of the same baseline as well. Even using BOS increases the performance significantly while COS yields even better results. Especially on the longer sequence with ten tasks there is a significant advantage of the latter. Unfortunately due to the drastically higher computational cost of COS no results on ImageNet-100 are available. But a better performance of COS when compared with BOS can be expected on this dataset as well.

Other methods like ICaRL or BiC achieve better results than the baseline on both CIFAR100 and ImageNet-100 for every sequence length. The BSIL and its variant using meta learning MBSIL proposed by Jodelet, Liu, and Murata [116] are closely related to DPNCL. In fact, they are equivalent in structure if an exponential activation function is used for the last layer of $h_{\boldsymbol{\theta}}(\mathbf{x})$. But despite their similarity on both datasets they fall behind DPNCL with BOS for both sequence lengths. This might be due to DPNCL using a sigmoid activation for the last layer of $h_{\boldsymbol{\theta}}(\mathbf{x})$, better optimized hyperparameters or the additional distillation loss. Similarly, LUCIR achieves lower results than DPNCL with BOS in all experiments.

When comparing the best results for every dataset and sequence length, DPNCL with BOS achieves the best results on CIFAR100 with a sequence length of five and outperforms PODNet significantly while TPCIL only marginally. On the ten step sequence and on ImageNet-100 TPCIL achieves the best results, closely followed by PODNet and DPNCL with BOS. But it is important to note that TPCIL stores not only training examples in its buffer but also additional information about the relation of those in an elastic Hebbian graph. Comparing the performance of DPNCL with the best performing method on the longer sequences reveals that especially on ImageNet-100 it falls behind. A possible reason for this might be the distillation that is only applied on the last layer in contrast to PODNet, where it is performed on all layers and the additional information stored by TPCIL.

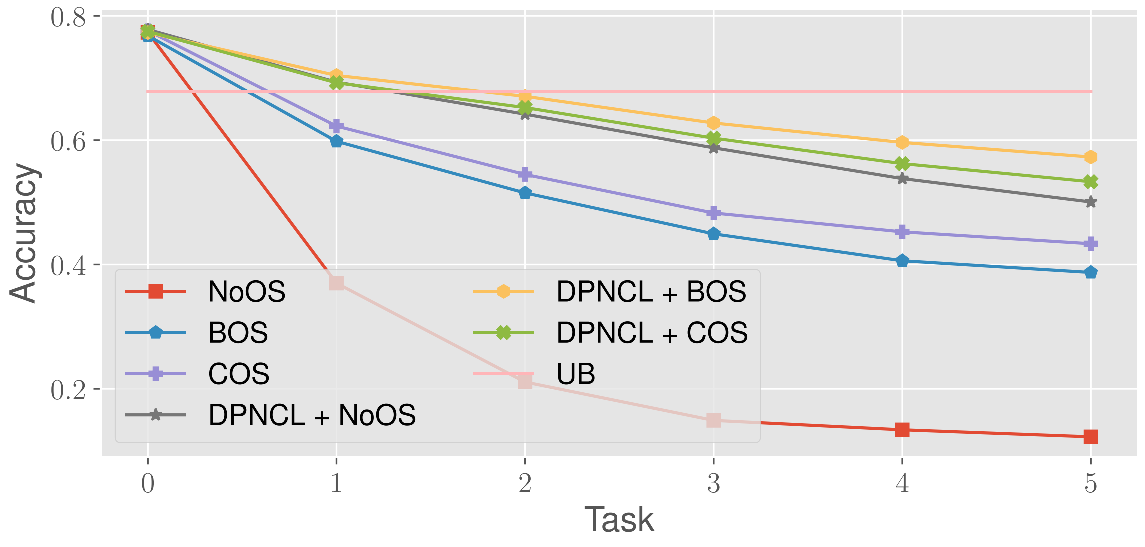
Comparing the above mentioned methods with the Upper Bound (UB) in Table 7.1 it seems that the best performing methods, i.e. DPNCL for CIFAR100 with five steps and TPCIL for all other cases, achieve similar results as the UB of training on all data simultaneously. But it has to be noted that the AIA is equivalent to the standard classification accuracy A if a sequence has only one task as it is the case for UB. If, however, there is more than one task in the sequence as it is the case for all results in Table 7.1 except the UB, they are not equivalent and the AIA is generally higher than the standard classification accuracy A determined on the last task. This means that although it might seem that a continual learning methods might achieve results close to the UB in terms of AIA, it might achieve a significantly lower classification accuracy on the final task as shown in the next chapter.

7.3.2. Comparison to Baselines and Upper Bound

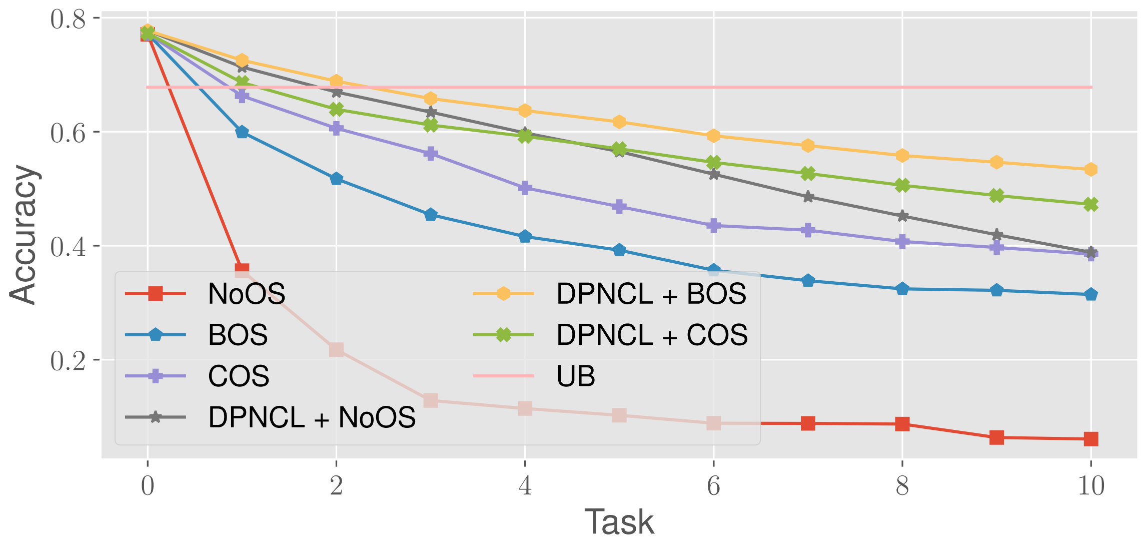
Apart from comparing the AIA, which summarizes the performance of a model for the whole continual learning sequence, it can also be of interest to compare its accuracy A on the test dataset throughout the ICL sequence. This metric is of more use for a practical application since it reflects the actual classification accuracy on a cumulative test dataset for every task. In contrast to this, the AIA is generally higher than that since it includes the test accuracy on previous tasks, which contain less classes and are therefore in general simpler to solve.

For this, Figure 7.4 shows the accuracy A of baselines, DPNCL and the UB over all tasks for the five and ten step sequences on CIFAR100. While all methods start with an identical performance on the base task, it is obvious that the baseline without oversampling (NoOS) suffers from severe catastrophic forgetting even after the first task, where its accuracy is already half of that achieved on the base task. During the next tasks its performance deteriorates exponentially until it reaches roughly 10% which is approximately the performance a model trained only on the last task with ten classes would achieve. The baseline with BOS on the other hand retains much of its initial performance but also experiences catastrophic forgetting. On the final task it achieves almost a four times higher accuracy than the baseline without oversampling. A slightly better result is obtained by the baseline using COS with a consistently higher accuracy. This advantage gets more pronounced for later tasks which makes it a much better choice for longer sequences if its increased computational cost is acceptable.

The motivation for incorporating the prior knowledge into the predictions as shown in (7.16) is to reduce the effects of distribution shifts caused by oversampling or a lack of it. Consequently, when using DPNCL the performance of a model during an ICL sequence should be independent of the chosen oversampling method. To some extend this is indeed the case as indicated by the results of DPNCL shown in Figure 7.4. All of these show significantly less catastrophic forgetting when compared with the baselines. But more importantly the dependence of their performance on the oversampling strategy is greatly reduced, especially on the five step sequence. For ten steps, the difference between oversampling strategies is reduced for DPNCL but still significant. Even NoOS achieves almost the same accuracy on every task when compared with BOS or COS. It is further interesting to note that BOS achieves even better results than COS for both sequence lengths with DPNCL. In combination with a much smaller computational cost when compared with COS, BOS is not only achieving a better accuracy but is also faster to train. But although on the first tasks there is virtually no difference

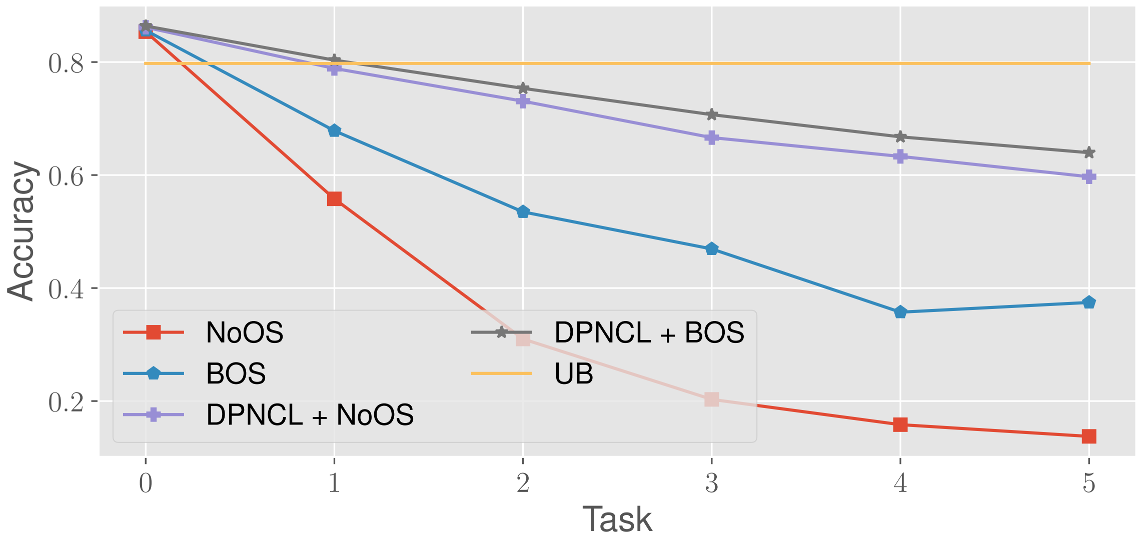


(a) 5 Tasks

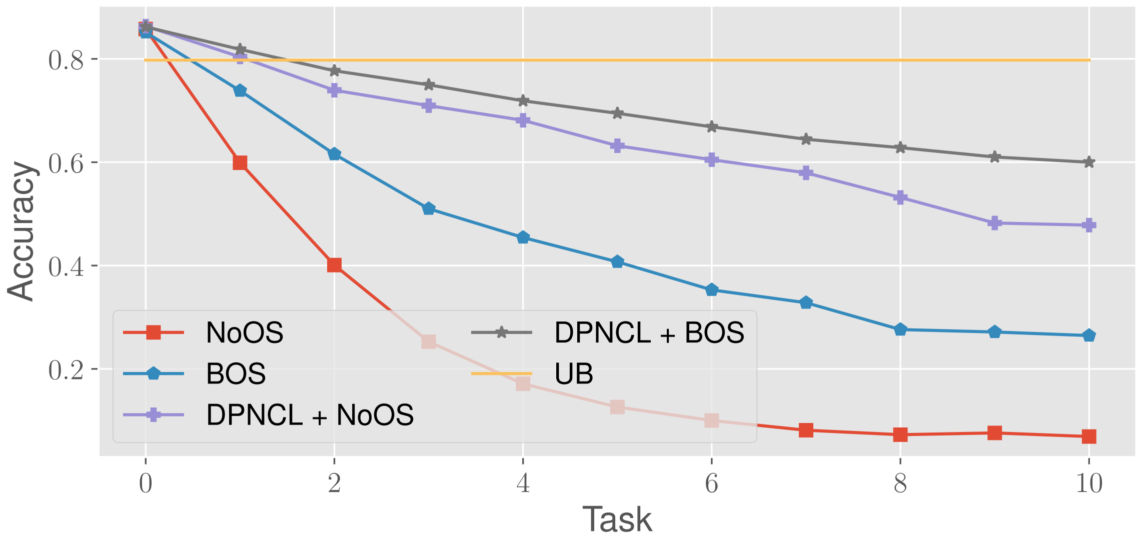


(b) 10 Tasks

Figure 7.4.: ICL sequence on CIFAR100 comparing the baseline and our method with different oversampling strategies.



(a) 5 Tasks



(b) 10 Tasks

Figure 7.5.: ICL sequence on ImageNet-100 comparing the baseline and our method with different oversampling strategies.

in performance, a growing discrepancy between BOS and NoOS with DPNCL can also be observed.

Similar results are obtained on ImageNet-100. The baselines again deteriorate rapidly in their performance but not as drastic when compared with CIFAR100. The difference between NoOS and BOS is also less severe but still very significant. A reason for this could be the much higher resolution of ImageNet-100 images when compared with CIFAR100 which might not only lead to more information stored in the buffer but also easier discrimination between classes. Overall these results and those shown in Table 7.1 indicate that the ICL sequence on ImageNet-100 is less difficult than the one on CIFAR100. Due to limited computational resources and the observation that BOS performed better on CIFAR100 when combined with DPNCL, an experiment using the baseline with COS is not performed on ImageNet-100.

Comparing the accuracy of all baselines and DPNCL on the final task with the UB confirms that indeed even the highest performing continual learning method still achieves significantly lower results than the UB. Since this accuracy on the final task is most relevant for a practical application, comparisons with an UB in terms of the AIA as they are common in the literature can be misleading.

7.4. Conclusion

As a closing remark of this chapter the motivating research question is considered again:

What types of distribution shifts can be introduced by rehearsal and how can their negative effects be avoided?

In order to determine what types of distribution shifts are possible, in section 7.2.1 several different types and their causes are discussed. Aspects like selecting examples to store and different oversampling strategies applied during rehearsal in the case of ICL are discussed. While distribution shifts are possible in both the high-dimensional continuous input and low-dimensional discrete output space, only for the latter an analytical approach is feasible. The very high-dimensional input space and the unknown form of distribution over it prevent an analytical approach for addressing distribution shifts in the input space.

Although the COS oversampling strategy prevents a distribution shift between rehearsal with a small buffer and an ideal scenario where all data can be stored in the rehearsal memory, it comes with serious limitations and drawbacks. Compared to NoOS and BOS it leads to an increasingly larger number of examples in a mini batch being sampled from the rehearsal memory and therefore slows down training. In addition to this, it also increases the risk of overfitting on the stored data since a large number of training iterations are performed on a rather small amount of training examples stored in the rehearsal memory.

As a way to incorporate prior knowledge about the oversampling strategy and rehearsal memory content into the prediction of a DNN, the concept of a DPN is introduced in section 7.2.2. Using this as a basis, an approach referred to as DPNCL by Wiewel, Bartler, and Yang [133] is introduced in section 7.2.3. It uses estimates of class probabilities from both the rehearsal memory and current task in combination with the prediction of a DNN in order to model a distribution shift due to rehearsal with a small memory. This reduces the effect of such shifts on the parameters that are learned and therefore reduces catastrophic forgetting. Through this modeling of known shifts the use of extensive oversampling and therefore a prolonged training including an increased risk of overfitting can be avoided.

The direct comparison of experimental results on the CIFAR100 and ImageNet-100 datasets presented in section 7.3.1 shows that this method not only achieves results that are competitive with but on CIFAR100 even outperforms recently proposed related methods. But on ImageNet-100 and for a ten step ICL sequence it is outperformed by two competing methods which use a more complex distillation loss and in one case store additional data in form of a graph that models the relation between individual training examples. Another interesting observation is that, in contrast to the baseline, DPNCL with BOS outperforms DPNCL with COS. Given that the former requires much less training iterations to complete an epoch when compared with the latter, this is a surprising and interesting result. A possible reason for this could be that during COS overfitting to the small number of samples stored in the rehearsal buffer is much more likely than during BOS. Without using DPNCL this overfitting might be unnoticeable since the performance degradation due to the rehearsal induced distribution shift dominates. But when using DPNCL overfitting to the rehearsal data is a more noticeable. An additional comparison to a baseline with different oversampling strategies presented in section 7.3.2 provides further evidence for the claim that incorporating the prior knowledge in this way reduces the effects of distribution shifts. As expected, the influence of an oversampling strategy used during rehearsal is greatly reduced when the prior knowledge is included compared with the baseline that does not include it.

In summary, it can be concluded that there are distribution shifts caused by using only a small rehearsal memory that can not store all data. While oversampling can to some extend overcome the negative effects of such shifts, it on its own introduces new problems like a significant slow down and risk of overfitting in the case of COS or a rather weak performance when compared with related methods. Using the theory of DPNs to incorporate prior knowledge about the oversampling strategy and memory contents into the prediction of a model is able to overcome these. It not only prevents the negative effects of a distribution shift in the output space but also enables the use of computationally much less demanding oversampling strategies like BOS. Extending the approach to settings other than ICL for supervised image classification or to also model shifts in the input space is left open for further research.

Chapter 8.

Conclusion and Future Work

In this thesis, different aspects of rehearsal-based continual learning for image classification are addressed in an attempt to contribute a better understanding of the current main challenge, i.e. catastrophic forgetting, and novel methods for overcoming it. The main goal of this thesis is to improve the capability of DNNs to learn on long sequences of tasks without requiring to store all previous training data. This would close an important gap between human and machine learning and move today's weak AI closer to AGI [90]. There are many different ways to approach continual learning as presented in chapter 3. This thesis focuses on the most promising family of rehearsal-based methods. Despite their rather simple approach of storing previously encountered training data for rehearsal during later tasks, they provide a strong baseline and basis for further improvements.

8.1. Summary of Contributions

The first contribution of this thesis is intended to study the phenomenon of catastrophic forgetting in more detail when compared with simply using scalar metrics. i.e. loss or accuracy. To gain more insight into which parts of a DNN are affected to what extend by catastrophic forgetting, a novel method for attributing an increase in loss to its weights is proposed. The process of training a DNN is interpreted as moving along a trajectory in the parameter space. The individual contributions are determined by integrating the gradient of a loss w.r.t. to the weights on the previous task along the trajectory of the new task for every weight separately. Afterwards these individual contributions can be aggregated and visualized at different levels, i.e. individual weights, layers or groups of layers. This is in contrast to the commonly used scalar quantities, i.e. losses or metrics, which can only indicate an increase in loss but not localize its cause. An application on this method to multiple datasets in three different continual learning scenarios, i.e. ICL, IDL and ITL, confirms experimental observations from previous work and provides new insights.

This study of catastrophic forgetting is followed by the second contribution of this thesis: a novel method for selecting which data to store in a rehearsal-memory for online continual learning. This less studied problem assumes that data becomes available in a stream of individual samples rather than tasks with clearly defined boundaries and large datasets. It is often associated with autonomous systems that experience and interact with the world. These systems are typically limited by their storage capacity and computational power. To overcome catastrophic forgetting in this setting, a novel method is derived from information theoretic principles. It maximizes an approximation of the entropy of a distribution associated with sampling uniformly at random from the rehearsal memory. It is shown through experimental evaluation that this method outperforms its closest competitor GSS on all tested datasets. Furthermore, it is also competitive with more complex related work that not only utilizes improved sample selection techniques but also employs more sophisticated rehearsal methods.

Next the use of synthetic data in the context of continual learning is addressed as the third contribution of this thesis. Based on previous work, which demonstrated a certain potential of synthetic data in rehearsal-based continual learning, a novel way of representing the synthetic data is presented. Its main idea is to avoid learning redundant information and storing synthetic data not individually. It rather learns components with associated weights that form synthetic data. This allows for sharing the same components across many different samples and therefore leads to a much more memory-efficient rehearsal. Experiments show a significant performance increase when compared to related work on several datasets, especially for small rehearsal memories.

Finally, the rehearsal process itself is analyzed with respect to distribution shifts caused by a limited memory size. It is shown theoretically that a rehearsal with limited memory size leads to a distribution shift which degrades the performance when compared to the ideal case of training on all data. This shift depends on the specific memory size, its content and the oversampling strategy. In order to mitigate the effects of such a distribution shift, a novel method based on DPNs is used. It incorporates prior knowledge about the rehearsal process and the current training dataset into the prediction of a DNN. This reduces the influence of the distribution shift on the learnable weights. Experimental results show that this method is competitive on several datasets and sequence lengths when compared with related work and even outperforms it in some cases. An ablation study further shows that the final continual learning performance is largely unaffected by the type of oversampling during rehearsal and therefore validates the main idea of this method.

8.2. Outlook and Open Questions

The results of this thesis show that recent methods for continual learning with DNNs for image classification can reduce the performance degradation due to catastrophic forgetting significantly. But there are still open questions which could not be answered in this thesis and are left open for potential future work.

Given that learning without forgetting requires some sort of memory, be it explicitly or implicitly through generative models or regularization terms, and there are no practical systems with unlimited resources, a continual learning system will at some point reach its maximum storage capacity. If learning is to be continued beyond this point, it has to be decided how to make resources available for new tasks. This inevitably means that some information about previously encountered tasks will be lost. Selecting what and how much to forget is a trade-off between freeing resources for new tasks while at the same time keeping the most relevant information about previously encountered tasks. This is an aspect that humans experience repeatedly throughout their life but it is rarely discussed in the literature. Usually a growing memory without a maximum size is used. A notable exception to this are online continual learning methods, similar to the one presented in chapter 5, where a fixed size memory is assumed and an algorithm is used to store only the most relevant data in memory.

In chapter 6 it is shown that sharing components of synthetic data for rehearsal has the potential to improve the memory efficiency of rehearsal-based methods significantly. The linear combination of components with a final non-linear function is a first attempt but more complicated compositions of synthetic data are possible. It might, for example, not be necessary for individual components to have the same size as the input data but rather could span only a partition of it. Sharing not only a fixed number of components between data of the same class but across all classes and reusing those learned on previous tasks cumulatively is a another direction that can be explored in order to further increase memory efficiency.

The main chapters of this thesis study different aspects like sample selection in chapter 5, learning synthetic rehearsal data in chapter 6 and improving the rehearsal process itself in chapter 7. But a natural next step is to combine these different approaches into one method. This is possible at least for the combination of ESS with DPNCL for online continual learning and CCMCL with DPNCL in offline continual learning. Such combinations might improve continual learning performance even more than one method individually can. But due to the limited scope of this thesis, such combinations were not studied. The simultaneous use of ESS and CCMCL in one method is not possible, since the former selects data from the dataset to store in the rehearsal memory while the latter learns completely synthetic data.

It is implicitly assumed in most of the literature that Backpropagation (BP) is a suitable algorithm for training DNNs in the context of continual learning. This assumption is probably driven by the overwhelming success that it achieved in recent years. But additional measures are required to achieve continual learning with DNNs that are trained using BP. In addition, there is also longstanding evidence from neuroscience that BP is biologically implausible as discussed in section 2.5. Given the success of biological systems in continual learning, it might therefore be worthwhile to investigate if more biologically plausible training algorithms like Direct Feedback Alignment (DFA) [47] are inherently more suitable for continual learning than BP.

Finally, there is more to continual learning than simply avoiding catastrophic forgetting. If its goal is to achieve human-like learning on a sequence of tasks, other aspects like the ability to quickly learn from only a few examples, improving on previously learned tasks by learning new ones, adapting to changing domains and utilizing multiple input modalities at the same time are also important. Incorporating more methods and insights from other fields like transfer, few-shot and self-supervised learning for building a system that is truly capable of continual learning is therefore required. After all, each of these aspects alone is probably not enough to achieve true human-like sequential learning without forgetting. Only a suitable combination of them will enable true continual learning with DNNs.

Appendix A.

Online Continual Learning and Buffer Management

This section presents the process of deriving the approximate solution to the optimization problem (5.7). Starting with the KDE from (5.5) we have

$$\begin{aligned} H(X|Y,\mathcal{M}_l) &= -\mathbb{E}_{p(\mathbf{x}|\mathbf{y},\mathcal{M}_l)}\left[\log p(X|Y,\mathcal{M}_l)\right] \\ &\approx -\mathbb{E}_{p(\mathbf{x}|\mathbf{y},\mathcal{M}_l)}\left[\log \frac{1}{M_y\sqrt{2\pi}}\sum_{l=1}^{M_y} \mathrm{e}^{\frac{-\|\mathbf{x}-\mathbf{x}_y[l]\|_2^2}{2}}\right]. \end{aligned} \tag{A.1}$$

Maximization over the index l of all possible memories $\mathcal{M}_l$ then yields

$$l^\star = \arg\max_l H(X|Y,\mathcal{M}_l) \approx \arg\max_l -\mathbb{E}_{p(\mathbf{x}|\mathbf{y},\mathcal{M}_l)}\left[\log \frac{1}{M_y\sqrt{2\pi}}\sum_{m=1}^{M_y} \mathrm{e}^{\frac{-\|\mathbf{x}-\mathbf{x}_y[m]\|_2^2}{2}}\right]. \tag{A.2}$$

Approximating the expectation operator by sampling at the modes $\mathbf{x}_y[m]$ then gives

$$l^\star = \arg\max_l H(X|Y,\mathcal{M}_l) \approx \arg\max_l \frac{-1}{M_y}\sum_{m=1}^{M_y} \log \frac{1}{M_y\sqrt{2\pi}}\sum_{n=1}^{M_y} \mathrm{e}^{\frac{-\|\mathbf{x}_y[m]-\mathbf{x}_y[n]\|_2^2}{2}}. \tag{A.3}$$

Using Jensen's inequality $-\log\frac{1}{n}\sum_{i=1}^{n} x_i \le -\frac{1}{n}\sum_{i=1}^{n}\log x_i$, the objective function can be lower bounded. This allows for maximizing the bound instead of the objective function and permits further simplification as

$$l^\star = \arg\max_l H(X|Y,\mathcal{M}_l) \approx \arg\max_l -\log \frac{1}{M_y^2\sqrt{2\pi}} \sum_{m=1}^{M_y}\sum_{n=1}^{M_y} \mathrm{e}^{\dfrac{-\|\mathbf{x}_y\,[m] - \mathbf{x}_y\,[n]\,\|_2^2}{2}} \tag{A.4}$$
$$= \arg\min_l \sum_{m=1}^{M_y}\sum_{n=1}^{M_y} \mathrm{e}^{\dfrac{-\|\mathbf{x}_y\,[m] - \mathbf{x}_y\,[n]\,\|_2^2}{2}}.$$

Since the exponential is a monotonically increasing function and the summation is performed over all possible distances between all examples stored in $\mathcal{M}_l$, $l^\star$ identifies the memory which has the maximum distance between all examples stored in it. This results in our final approximation

$$l^\star = \arg\max_l H(X|Y,\mathcal{M}_l) \approx \arg\max_l \sum_{m=1}^{M_y}\sum_{n=1}^{M_y} \|\mathbf{x}_y\,[m] - \mathbf{x}_y\,[n]\,\|_2^2. \tag{A.5}$$

Appendix B.

Dirichlet Prior Networks for Continual Learning

Following Bishop and Nasrabadi [23, 76 f.], consider the categorical likelihood function for $N = \sum_{c=1}^{C} N_c$ observations from a dataset $\mathcal{D}$

$$p(\mathcal{D}|\boldsymbol{\mu}) = \prod_{c=1}^{C} \mu_c^{N_c}, \tag{B.1}$$

where N_c is the number of observations with class label c. With the Dirichlet prior

$$\mathrm{Dir}(\boldsymbol{\mu}) = \frac{\Gamma(\alpha_0)}{\prod_{c=1}^{C} \Gamma(\alpha_c)} \prod_{c=1}^{C} \mu_c^{\alpha_c - 1} \tag{B.2}$$

follows for the posterior

$$p(\boldsymbol{\mu}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}) \propto \prod_{c=1}^{C} \mu_c^{N_c + \alpha_c - 1}. \tag{B.3}$$

According to Bishop and Nasrabadi [23, 76 f.] this is again the form of a Dirichlet distribution whose normalization coefficient can be obtained by comparison with (B.2). This leads to

$$p(\boldsymbol{\mu}|\mathcal{D}) = \frac{\Gamma(\alpha_0 + N)}{\prod_{c=1}^{C} \Gamma(\alpha_c + N_c)} \prod_{c=1}^{C} \mu_c^{\alpha_c + N_c - 1}. \tag{B.4}$$

Hence the posterior $p(\boldsymbol{\mu}|\mathcal{D})$ is of the same form as the Dirichlet prior and according to Definition 8 the Dirichlet distribution is indeed the conjugate prior to the categorical distribution.

In section 7.2.2 it is stated that according to Malinin and Gales [71] the distribution over class labels can be determined through marginalization as

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} = \prod_{c=1}^{C} \left(\frac{\alpha_c}{\alpha_0}\right)^{y_c}. \tag{B.5}$$

In order to show that this is indeed true, the probability $p(y_1 = 1, y_2 = 0, \dots, y_c = 0|\mathbf{x})$ is considered as an example. Note that the dependence of the concentration parameters of the Dirichlet distribution $\boldsymbol{\alpha}$ on $\mathbf{x}$ is omitted for a cleaner notation. For this specific case follows

$$\begin{aligned}
p(y_1 = 1, y_2 = 0, \dots, y_c = 0|\mathbf{x}) &= \int p(y_1 = 1, y_2 = 0, \dots, y_c = 0|\boldsymbol{\mu})\mathrm{Dir}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} \\
&= \underbrace{\int \mu_1 \frac{\Gamma(\alpha_0)}{\prod_{c=1}^{C}\Gamma(\alpha_c)} \prod_{c=1}^{C} \mu_c^{\alpha_c - 1}\mathrm{d}\boldsymbol{\mu}}_{\mathbb{E}_{\mu\sim\mathrm{Dir}(\mu|\mathbf{x})}[\mu_1]} \\
&= \frac{\Gamma(\alpha_0)\Gamma(\alpha_1+1)}{\Gamma(\alpha_0+1)\Gamma(\alpha_1)} \underbrace{\int \frac{\Gamma(\alpha_0+1)}{\Gamma(\alpha_1+1)\prod_{c=2}^{C}\Gamma(\alpha_c)} \mu_1^{\alpha_1+1-1} \prod_{c=2}^{C} \mu_c^{\alpha_c-1}\mathrm{d}\boldsymbol{\mu}}_{\int \mathrm{Dir}(\mu|\tilde{\alpha})\mathrm{d}\mu = 1} \\
&= \frac{(\alpha_0 - 1)!\alpha_1!}{\alpha_0!(\alpha_1 - 1)!} = \frac{\alpha_1}{\alpha_0},
\end{aligned} \tag{B.6}$$

where $\tilde{\boldsymbol{\alpha}} = [\alpha_1 + 1, \alpha_2, \dots, \alpha_C]$ and the property $\Gamma(x + 1) = x\Gamma(x)$ of the gamma function is used. By analogous derivations for all other $C - 1$ values of the one-hot vector $\mathbf{y}$, this results can be generalized to

$$p(\mathbf{y}|\mathbf{x}) = \prod_{c=1}^{C} \left(\frac{\alpha_c}{\alpha_0}\right)^{y_c}, \tag{B.7}$$

which shows that (7.17) is indeed true.

Bibliography

[1] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[2] D. O. Hebb. *The Organization of Behavior*. JOHN WILEY and SONS, 1949.

[3] Robert Gray. “Vector quantization”. In: *IEEE Assp Magazine* 1.2 (1984), pp. 4–29.

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

[5] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[7] Jeffrey C Schlimmer and Douglas Fisher. “A case study of incremental concept induction”. In: *AAAI*. Vol. 86. 1986, pp. 496–501.

[8] Michael C Shewry and Henry P Wynn. “Maximum entropy sampling”. In: *Journal of applied statistics* 14.2 (1987), pp. 165–170.

[9] Evelyn Fix and Joseph Lawson Hodges. “Discriminatory analysis. Nonparametric discrimination: Consistency properties”. In: *International Statistical Review/Revue Internationale de Statistique* 57.3 (1989), pp. 238–247.

[10] 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.

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

[12] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[13] David JC MacKay. “A practical Bayesian framework for backpropagation networks”. In: *Neural computation* 4.3 (1992), pp. 448–472.

[14] Vladimir Vapnik. “Principles of risk minimization for learning theory”. In: *Advances in neural information processing systems*. 1992, pp. 831–838.

[15] Kenneth D. Miller and David J. C. MacKay. “The Role of Constraints in Hebbian Learning”. In: *Neural Computation* 6.1 (Jan. 1994), pp. 100–126.

[16] 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.

[17] Larry F Abbott and Sacha B Nelson. "Synaptic plasticity: taming the beast". In: *Nature neuroscience* 3.11 (2000), pp. 1178–1183.

[18] Carlotta Domeniconi and Dimitrios Gunopulos. "Incremental support vector machine construction". In: *Proceedings 2001 ieee international conference on data mining*. IEEE. 2001, pp. 589–592.

[19] Masa-Aki Sato. "Online model selection based on the variational Bayes". In: *Neural computation* 13.7 (2001), pp. 1649–1681.

[20] Susan M Barnett and Stephen J Ceci. "When and where do we apply what we learn?: A taxonomy for far transfer." In: *Psychological bulletin* 128.4 (2002), p. 612.

[21] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

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

[23] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[24] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 2007.

[25] Marco F Huber, Tim Bailey, Hugh Durrant-Whyte, and Uwe D Hanebeck. "On entropy approximation for Gaussian mixture random vectors". In: *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE. 2008, pp. 181–188.

[26] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. Mit Press, 2008.

[27] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[28] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[29] Max Welling. "Herding dynamical weights to learn". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 1121–1128.

[30] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[31] 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).

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[33] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[34] 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).

[35] 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).

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. “How transferable are features in deep neural networks?” In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[39] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[40] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456.

[41] Durk P Kingma, Tim Salimans, and Max Welling. “Variational dropout and the local reparameterization trick”. In: *Advances in neural information processing systems* 28 (2015), pp. 2575–2583.

[42] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[44] 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.

[45] Simone Kühn and Ulman Lindenberger. "Research on human plasticity in adulthood: A lifespan agenda". In: *Handbook of the psychology of aging*. Elsevier, 2016, pp. 105–123.

[46] Micah M. Murray, David J. Lewkowicz, Amir Amedi, and Mark T. Wallace. "Multisensory Processes: A Balancing Act across the Lifespan". In: *Trends in Neurosciences* 39.8 (2016), pp. 567–579.

[47] Arild Nøkland. "Direct feedback alignment provides learning in deep neural networks". In: *Advances in neural information processing systems* 29 (2016).

[48] 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).

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

[50] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. "Matching networks for one shot learning". In: *Advances in neural information processing systems* 29 (2016).

[51] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of wasserstein gans". In: *Advances in neural information processing systems* 30 (2017).

[52] David J. Heeger. "Theory of cortical function". In: *Proceedings of the National Academy of Sciences* 114.8 (2017), pp. 1773–1782.

[53] 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* 114.13 (2017), pp. 3521–3526.

[54] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. "Overcoming catastrophic forgetting by incremental moment matching". In: *Advances in neural information processing systems* 30 (2017).

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

[56] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in neural information processing systems* 30 (2017), pp. 6467–6476.

[57] Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. “Encoder based lifelong learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1320–1328.

[58] 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.

[59] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. “Continual Learning with Deep Generative Replay”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[60] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical networks for few-shot learning”. In: *Advances in neural information processing systems* 30 (2017).

[61] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. “Full resolution image compression with recurrent neural networks”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 5306–5314.

[62] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *CoRR* abs/1708.07747 (2017).

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

[64] 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.

[65] Xiangrui Chao, Gang Kou, Tie Li, and Yi Peng. “Jie Ke versus AlphaGo: A ranking approach using decision making method for large-scale data with incomplete information”. In: *European Journal of Operational Research* 265.1 (2018), pp. 239–247.

[66] 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.

[67] Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. “Don’t forget, there is more than forgetting: new metrics for Continual Learning”. In: *Continual Learning Workshop at NeuRIPS 2018* (2018).

[68] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. “Re-evaluating continual learning scenarios: A categorization and case for strong baselines”. In: *arXiv preprint arXiv:1810.12488* (2018).

[69] Arthur Jacot, Franck Gabriel, and Clement Hongler. “Neural Tangent Kernel: Convergence and Generalization in Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

[70] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. “Rotate your networks: Better weight consolidation and less catastrophic forgetting”. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE. 2018, pp. 2262–2268.

[71] Andrey Malinin and Mark Gales. “Predictive uncertainty estimation via prior networks”. In: *Advances in neural information processing systems* 31 (2018).

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

[73] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. “Conditional probability models for deep image compression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4394–4402.

[74] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. “Variational Continual Learning”. In: *International Conference on Learning Representations*. 2018.

[75] Hippolyt Ritter, Aleksandar Botev, and David Barber. “Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

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

[77] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. “Progress and Compress: A scalable framework for continual learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 4528–4537.

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

[79] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. “Dataset distillation”. In: *arXiv preprint arXiv:1811.10959* (2018).

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

[81] 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 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 11849–11860.

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

[83] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. “Gradient based sample selection for online continual learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[84] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. “Efficient Lifelong Learning with A-GEM”. In: *International Conference on Learning Representations*. 2019.

[85] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. “Variable rate deep image compression with a conditional autoencoder”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3146–3154.

[86] Koen V Haak and Christian F Beckmann. “Plasticity versus stability across the human cortical visual connectome”. In: *Nature communications* 10.1 (2019), pp. 1–8.

[87] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. “Learning a unified classifier incrementally via rebalancing”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 831–839.

[88] Andrey Malinin and Mark Gales. “Reverse kl-divergence training of prior networks: Improved uncertainty and adversarial robustness”. In: *Advances in Neural Information Processing Systems* 32 (2019).

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

[90] Henry Shevlin, Karina Vold, Matthew Crosby, and Marta Halina. “The limits of machine intelligence: Despite progress in machine intelligence, artificial general intelligence is still a major challenge”. In: *EMBO reports* 20.10 (2019), e49177.

[91] Gido M Van de Ven and Andreas S Tolias. “Three scenarios for continual learning”. In: *arXiv preprint arXiv:1904.07734* (2019).

[92] James CR Whittington and Rafal Bogacz. “Theories of error back-propagation in the brain”. In: *Trends in cognitive sciences* 23.3 (2019), pp. 235–250.

[93] Felix Wiewel and Bin Yang. “Continual Learning for Anomaly Detection with Variational Autoencoder”. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 3837–3841.

[94] Felix Wiewel and Bin Yang. “Localizing Catastrophic Forgetting in Neural Networks”. In: *CoRR* abs/1906.02568 (2019).

[95] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. “Large scale incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 374–382.

[96] Zalán Borsos, Mojmir Mutny, and Andreas Krause. “Coresets via Bilevel Optimization for Continual Learning and Streaming”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 14879–14890.

[97] Zalán Borsos, Mojmir Mutny, and Andreas Krause. “Coresets via bilevel optimization for continual learning and streaming”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 14879–14890.

[98] Arslan Chaudhry, Naeemullah Khan, Puneet Dokania, and Philip Torr. “Continual learning in low-rank orthogonal subspaces”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9900–9911.

[99] 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*. Vol. 12365. Springer. 2020, pp. 86–102.

[100] Tongtong Fang, Nan Lu, Gang Niu, and Masashi Sugiyama. “Rethinking Importance Weighting for Deep Learning under Distribution Shift”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 11996–12007.

[101] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. “Orthogonal gradient descent for continual learning”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 3762–3773.

[102] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[103] Tyler L. Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. “REMIND Your Neural Network to Prevent Catastrophic Forgetting”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Cham: Springer International Publishing, 2020, pp. 466–483.

[104] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. “Federated Learning: Challenges, Methods, and Future Directions”. In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60.

[105] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. “Backpropagation and the brain”. In: *Nature Reviews Neuroscience* 21.6 (2020), pp. 335–346.

[106] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. “Latent replay for real-time continual learning”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10203–10209.

[107] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. “Gdumb: A simple approach that questions our progress in continual learning”. In: *European conference on computer vision*. Springer. 2020, pp. 524–540.

[108] Jiawei Ren, Cunjun Yu, Xiao Ma, Haiyu Zhao, Shuai Yi, et al. “Balanced meta-softmax for long-tailed visual recognition”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4175–4186.

[109] Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. “Topology-preserving class-incremental learning”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 254–270.

[110] Gido M van de Ven, Hava T Siegelmann, and Andreas S Tolias. “Brain-inspired replay for continual learning with artificial neural networks”. In: *Nature communications* 11.1 (2020), pp. 1–14.

[111] Felix Wiewel, Andreas Brendle, and Bin Yang. “Continual Learning Through One-Class Classification Using VAE”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 3307–3311.

[112] 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.

[113] Sawsan AbdulRahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. “A survey on federated learning: The journey from centralized to distributed on-site learning and beyond”. In: *IEEE Internet of Things Journal* 8.7 (2021), pp. 5476–5497.

[114] Arslan Chaudhry, Albert Gordo, Puneet Kumar Dokania, Philip H. S. Torr, and David Lopez-Paz. “Using Hindsight to Anchor Past Knowledge in Continual Learning”. In: *AAAI*. 2021.

[115] Ze Cui, Jing Wang, Shangyin Gao, Tiansheng Guo, Yihui Feng, and Bo Bai. “Asymmetric gained deep image compression with continuous rate adaptation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10532–10541.

[116] Quentin Jodelet, Xin Liu, and Tsuyoshi Murata. “Balanced softmax cross-entropy for incremental learning”. In: *International Conference on Artificial Neural Networks*. Springer. 2021, pp. 385–396.

[117] Ta-Chu Kao, Kristopher Jensen, Gido van de Ven, Alberto Bernacchia, and Guillaume Hennequin. “Natural continual learning: success is a journey, not (just) a destination”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28067–28079.

[118] Wouter M. Kouw and Marco Loog. “A Review of Domain Adaptation without Target Labels”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.3 (2021), pp. 766–785.

[119] Noel Loo, Siddharth Swaroop, and Richard E Turner. “Generalized Variational Continual Learning”. In: *International Conference on Learning Representations*. 2021.

[120] Sudhanshu Mittal, Silvio Galesso, and Thomas Brox. “Essentials for class incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3513–3522.

[121] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. “Dataset Distillation with Infinitely Wide Convolutional Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 5186–5198.

[122] Dongsub Shim, Zheda Mai, Jihwan Jeong, Scott Sanner, Hyunwoo Kim, and Jongseong Jang. “Online Class-Incremental Continual Learning with Adversarial Shapley Value”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 11. 2021, pp. 9630–9638.

[123] Ilia Sucholutsky and Matthias Schonlau. “Soft-label dataset distillation and text dataset distillation”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[124] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. “Multi-Task Learning for Dense Prediction Tasks: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1.

[125] Felix Wiewel and Bin Yang. “Condensed Composite Memory Continual Learning”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[126] Felix Wiewel and Bin Yang. “Entropy-based Sample Selection for Online Continual Learning”. In: *2020 28th European Signal Processing Conference (EUSIPCO)*. 2021, pp. 1477–1481.

[127] Yu Zhang and Qiang Yang. “A Survey on Multi-Task Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1.

[128] Bo Zhao and Hakan Bilen. “Dataset condensation with differentiable siamese augmentation”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12674–12685.

[129] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. “Dataset Condensation with Gradient Matching”. In: *International Conference on Learning Representations*. 2021.

[130] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. “A Comprehensive Survey on Transfer Learning”. In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76.

[131] Robert A. Marsden, Felix Wiewel, Mario Döbler, Yang Yang, and Bin Yang. “Continual Unsupervised Domain Adaptation for Semantic Segmentation using a Class-Specific Transfer”. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8.

[132] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. “Continual test-time domain adaptation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 7201–7211.

[133] Felix Wiewel, Alexander Bartler, and Bin Yang. “Dirichlet Prior Networks for Continual Learning”. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. 2022, pp. 1–8.

[134] Niclas Vödisch, Daniele Cattaneo, Wolfram Burgard, and Abhinav Valada. “Continual SLAM: Beyond lifelong simultaneous localization and mapping through continual learning”. In: *Robotics Research*. Springer, 2023, pp. 19–35.

[135] Niclas Vödisch, Daniele Cattaneo, Wolfram Burgard, and Abhinav Valada. “CoVIO: Online Continual Learning for Visual-Inertial Odometry”. In: *arXiv:2303.10149* (2023).

[136] Niclas Vödisch, Kürsat Petek, Wolfram Burgard, and Abhinav Valada. “CoDEPS: Online Continual Learning for Depth Estimation and Panoptic Segmentation”. In: *arXiv:2303.10147* (2023).