

## Continual learning by subnetwork creation and selection

Dekhovich, A.

**DOI**

[10.4233/uuid:d67da1e3-5792-4516-9bf0-f795633152bf](https://doi.org/10.4233/uuid:d67da1e3-5792-4516-9bf0-f795633152bf)

**Publication date**

2024

**Document Version**

Final published version

**Citation (APA)**

Dekhovich, A. (2024). *Continual learning by subnetwork creation and selection*. [Dissertation (TU Delft), Delft University of Technology]. <https://doi.org/10.4233/uuid:d67da1e3-5792-4516-9bf0-f795633152bf>

**Important note**

To cite this publication, please use the final published version (if applicable).  
Please check the document version above.

**Copyright**

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

**Takedown policy**

Please contact us and provide details if you believe this document breaches copyrights.  
We will remove access to the work immediately and investigate your claim.

# Continual learning by subnetwork creation and selection



Aleksandr Dekhovich



# **CONTINUAL LEARNING BY SUBNETWORK CREATION AND SELECTION**



# **CONTINUAL LEARNING BY SUBNETWORK CREATION AND SELECTION**

## **Proefschrift**

ter verkrijging van de graad van doctor  
aan de Technische Universiteit Delft,  
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,  
voorzitter van het College voor Promoties,  
in het openbaar te verdedigen  
op dinsdag 25 juni 2024 om 12:30 uur

door

**Aleksandr DEKHOVICH**

Master of Science in Fundamental Mathematics and Mechanics  
Lomonosov Moscow State University, Rusland,  
geboren te Podolsk, Rusland.

Dit proefschrift is goedgekeurd door de

promotor: Dr. ir. M.H.F. Sluiter	Technische Universiteit Delft
copromotor: Dr. D.M.J. Tax	Technische Universiteit Delft

Samenstelling promotiecommissie bestaat uit:

Rector Magnificus	voorzitter
Dr. ir. M.H.F. Sluiter	Technische Universiteit Delft, promotor
Dr. D.M.J. Tax	Technische Universiteit Delft, copromotor

*Onafhankelijke leden:*

Prof. dr. ir. C. Vuik	Technische Universiteit Delft
Prof. dr. M. Biehl	Rijksuniversiteit Groningen
Dr. E. Gavves	Universiteit van Amsterdam
Dr. J.F.P. Kooij	Technische Universiteit Delft
Prof. dr. ir. M.H.G. Verhaegen	Technische Universiteit Delft, reservelid

*Overig lid:*

Dr. M.A. Bessa	Brown University, USA / Technische Universiteit Delft
----------------	---



*Keywords:* deep learning, continual learning, catastrophic forgetting, scientific machine learning, cooperative modeling

*Cover image:* generated by the author using Midjourney

*Cover design:* Proefschriftmaken.nl

*Printed by:* Proefschriftmaken.nl

*Style:* TU Delft House Style, with modifications by Moritz Beller  
<https://github.com/Inventitech/phd-thesis-template>

The author set this thesis in  $\text{\LaTeX}$  using the Libertinus and Inconsolata fonts.

Copyright © 2024 by A. Dekhovich  
ISBN 978-94-6469-983-8

An electronic version of this dissertation is available at  
<http://repository.tudelft.nl/>.

# CONTENTS

<b>Summary</b>	<b>ix</b>
<b>Samenvatting</b>	<b>xi</b>
<b>Acknowledgments</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Continual learning for deep neural networks . . . . .	2
1.1.1 Different continual learning scenarios . . . . .	3
1.2 General paradigm . . . . .	5
1.2.1 Research Goal and Research Questions . . . . .	5
1.3 Thesis Outline . . . . .	6
<b>Part I: Methodology</b>	<b>9</b>
<b>2 Creation of task-specific subnetworks</b>	<b>11</b>
2.1 Introduction . . . . .	12
2.2 Related work . . . . .	12
2.3 Proposed method. . . . .	13
2.4 Numerical experiments. . . . .	16
2.4.1 LeNet architecture . . . . .	16
2.4.2 VGG architecture. . . . .	17
2.4.3 ResNet architecture . . . . .	19
2.5 Discussion . . . . .	22
2.5.1 Comparison with Magnitude-based approach . . . . .	22
2.5.2 Error estimation . . . . .	24
2.6 Conclusion. . . . .	27
<b>3 Continual learning with specialized subnetworks</b>	<b>29</b>
3.1 Introduction . . . . .	30
3.2 Related work . . . . .	30
3.3 Proposed method. . . . .	34
3.4 Numerical experiments. . . . .	36
3.5 Further analysis . . . . .	42
3.6 Conclusion. . . . .	44
<b>4 Subnetwork selection in surface defect segmentation problem</b>	<b>47</b>
4.1 Introduction . . . . .	48
4.2 Related work . . . . .	49
4.3 Proposed method. . . . .	50
4.3.1 Subnetwork creation . . . . .	52
4.3.2 Subnetwork selection . . . . .	53



4.4	Numerical experiments. . . . .	54
4.4.1	SD-saliency-900 dataset . . . . .	55
4.4.2	Magnetic tile defects dataset . . . . .	56
4.4.3	Hyperparameters choice . . . . .	59
4.5	Conclusion. . . . .	61
<b>Part II: Application</b>		<b>63</b>
<b>5</b>	<b>Incremental learning for physics-informed neural networks</b>	<b>65</b>
5.1	Introduction . . . . .	66
5.2	Related work. . . . .	68
5.3	Problem formulation . . . . .	69
5.4	Proposed method. . . . .	70
5.5	Numerical experiments. . . . .	72
5.5.1	Results . . . . .	72
5.6	Additional study . . . . .	76
5.6.1	Sensitivity to hyperparameters. . . . .	76
5.6.2	Subnetworks analysis . . . . .	77
5.7	Conclusion. . . . .	78
<b>6</b>	<b>Coopertative modeling via continual learning of different material behavior</b>	<b>79</b>
6.1	Introduction . . . . .	80
6.2	Related work. . . . .	81
6.3	Proposed method. . . . .	82
6.4	First case study: learning plasticity laws for different microstructures . . . . .	85
6.4.1	Problem description . . . . .	85
6.4.2	Results . . . . .	88
6.5	Second case study: RVEs with periodic boundary conditions . . . . .	90
6.5.1	Problem description . . . . .	90
6.5.2	Results . . . . .	91
6.6	Discussion . . . . .	95
6.6.1	Architectures comparison . . . . .	95
6.6.2	Knowledge transfer . . . . .	96
6.7	Conclusion. . . . .	96
<b>Conclusions</b>		<b>99</b>
<b>7</b>	<b>Conclusion and Future Work</b>	<b>101</b>
7.1	Conclusion. . . . .	102
7.2	Future Work . . . . .	103
<b>Appendices</b>		<b>105</b>
<b>A</b>	<b>Appendix to Chapter 2</b>	<b>107</b>
A.1	Pruning setups . . . . .	107
A.2	FLOPs computation . . . . .	108

---

<b>B</b>	<b>Appendix to Chapter 3</b>	<b>109</b>
B.1	Additional information on CIFAR-100 experiments . . . . .	109
B.2	ImageNet-100/1000 results . . . . .	112
B.3	CUB-200-2011 additional comparison . . . . .	112
<b>C</b>	<b>Appendix to Chapter 5</b>	<b>113</b>
C.1	Additional information about 1-D reaction PDE . . . . .	113
C.2	Additional information about 1-D convection PDE . . . . .	114
<b>D</b>	<b>Appendix to Chapter 6</b>	<b>115</b>
D.1	Additional information on the First case study . . . . .	115
	<b>Bibliography</b>	<b>117</b>
	<b>Glossary</b>	<b>139</b>
	<b>List of Publications</b>	<b>141</b>



## SUMMARY

Deep learning models have made enormous strides over the past decade. However, they still have some disadvantages when dealing with changing data streams. One of these flaws is the phenomenon called catastrophic forgetting. It occurs when a model learns multiple tasks sequentially, having access only to the data of the current task. However, this scenario has strong implications for real-world machine learning and engineering problems where new information is introduced into the system over time. Continual learning is a subfield of deep learning that aims to work in this scenario. Therefore, this thesis presents a general continual learning paradigm to tackle the catastrophic forgetting issue in deep learning models, regardless of architecture.

Following ideas from the neuroscience literature, we create task-specific regions in the network, i.e. subnetworks, to encode information there. Thus, some parameters are responsible for solving this task, which mitigates forgetting compared to conventional training where the trainable parameters are simultaneously assigned to all tasks. A proper subnetwork should be then selected by the algorithm to make a prediction or information about the correct subnetwork must be given by the user. The subnetworks can share some connections to transfer knowledge between each other and facilitate future learning.

In the first part of the thesis, we describe the proposed methodology: task-specific subnetwork creation during training and the proper subnetwork selection during inference stages. We examine different subnetwork prediction strategies outlining their advantages and disadvantages. We validate the proposed algorithms on a series of well-known image datasets in computer vision in classification and semantic segmentation tasks. The proposed solution significantly outperforms current state-of-the-art methods by 10-20% of accuracy.

The second part of the thesis illustrates the benefits of cooperative learning via continual learning in physical sciences and solid mechanic examples. We demonstrate that by sharing parameters, the following subnetwork can be trained either with lower prediction error, requiring fewer training data points, or both, compared to conventional training with one network per task. Importantly, the model does not forget any of the acquired knowledge since once a parameter is assigned to a subnetwork, it is not changed when training new tasks. We would like to highlight the potential importance of further development of continual learning methods in engineering to improve the generalization capabilities of the models.

The thesis concludes by discussing the main results and findings. We also outline the main limitations of the work and directions for improvement. Further development of continual learning models will lead to more advanced artificial intelligence systems that should contribute to solving a wider range of problems.



---

## SAMENVATTING

Modellen voor diep leren hebben de afgelopen tien jaar enorme vooruitgang geboekt. Ze hebben echter nog steeds enkele tekortkomingen als het gaat om veranderende datastromen. Een van deze tekortkomingen is het fenomeen dat “catastrofaal vergeten” wordt genoemd. Het komt voor wanneer een model meerdere taken achter elkaar leert en alleen toegang heeft tot de gegevens van de huidige taak. Dit scenario heeft grote gevolgen voor echte machine learning- en engineeringproblemen waarbij in de loop van de tijd nieuwe informatie in het systeem wordt geïntroduceerd. Continu leren is een deelgebied van diepgaand leren dat zich met dit scenario bezig houdt. Daarom presenteert dit proefschrift een algemeen continu leerparadigma om het probleem van “catastrofaal vergeten” in diep leren modellen aan te pakken, ongeacht de architectuur.

Op basis van ideeën uit de neurowetenschappelijke literatuur creëren we taakspecifieke regio's in het netwerk, d.w.z. subnetwerken, om daar informatie te coderen. Sommige parameters zijn dus verantwoordelijk voor het oplossen van deze taak, wat het vergeten vermindert in vergelijking met conventionele training waarbij de trainbare parameters tegelijkertijd aan alle taken worden toegewezen. Vervolgens moet door het algoritme een juist subnetwerk worden geselecteerd om een voorspelling te doen, of de gebruiker moet informatie over het juiste subnetwerk geven. De subnetwerken kunnen een aantal verbindingen delen om kennis onderling over te dragen en toekomstig leren te vergemakkelijken.

In het eerste deel van het proefschrift beschrijven we de voorgestelde methodologie: taakspecifieke subnetwerkcreatie tijdens training en de juiste subnetwerkselectie tijdens inferentiefasen. We onderzoeken verschillende voorspellingstrategieën voor subnetwerken en schetsen hun voor- en nadelen. We valideren de voorgestelde algoritmen op een reeks bekende beelddatasets in computer vision in classificatie- en semantische segmentatietaken. De voorgestelde oplossing presteert aanzienlijk beter dan de huidige state-of-the-art methoden met een verbetering van 10-20% van de nauwkeurigheid.

Het tweede deel van het proefschrift illustreert de voordelen van coöperatief leren via continu leren in de natuurwetenschappen en met voorbeelden uit de mechanica van de vaste stof. We laten zien dat door het delen van parameters opeenvolgende subnetwerken kunnen worden getraind met een lagere voorspellingsfout, of met minder trainingsdatapunten, of beide, vergeleken met conventionele training met één netwerk per taak. Belangrijk is dat het model niets van de opgedane kennis vergeet, aangezien een parameter die eenmaal aan een subnetwerk is toegewezen, niet meer wordt gewijzigd bij het trainen van nieuwe taken. We willen graag het potentiële belang benadrukken van de verdere ontwikkeling van continue leermethoden in de techniek om de generalisatiemogelijkheden van de modellen te verbeteren.

Het proefschrift wordt afgesloten met een bespreking van de belangrijkste resultaten en bevindingen. Ook schetsen we de belangrijkste beperkingen van het werk en de verbeterpunten. Verdere ontwikkeling van modellen voor continu leren zal leiden tot

geavanceerdere kunstmatige-intelligentiesystemen die zouden moeten bijdragen aan het oplossen van een breder scala aan problemen.

# ACKNOWLEDGMENTS

First of all, I want to thank my parents Eduard and Elizaveta. Your support throughout these four years allowed me to complete my PhD program without going crazy. Your confidence in me remains unchanged throughout my life and helps me overcome life's obstacles. I know I can always rely on you in difficult moments and you will support me. Thank you for the life guidelines you endowed me with, which led me to academia. I also want to mention our cat Simba, who never forgets me even though our time together is infrequent and short, and our cat Iris, who was an important member of the family while I was growing up.

I am grateful to my supervisor Miguel Bessa for the opportunity to work in his laboratory on a fascinating topic. Thank you for the high level of flexibility in my work, which has allowed me to solve a wide range of problems. We have chosen a very ambitious path but your optimism helped me not despair during difficult times throughout the program, especially in its first half. Of course, thanks to my promotor Marcel Sluiter, both for your scientific guidance and for sharing your personal life experience and thoughts. You have always advised me not only about my PhD but also about my future career. I want to thank David Tax, who also supervised me for these four years. I highly appreciate your feedback and the expertise in machine learning that you shared with me.

I want to thank all the Bessa Research Group members for the useful and interesting discussion of new ideas and joint activities outside of work. Our group brings together people with very different backgrounds and knowledge, which makes it especially unique. I also wish you good luck in your scientific and other paths. I especially want to thank my colleagues Jiaxiang Yi and Taylan Turan for an important experience of joint work. It was a productive collaboration that ultimately proved successful.

Last but not least, I want to thank the people who supported my application to the PhD program: my former supervisor Lev V. Lokutsievskiy and lecturers Maria I. Ronzhina, Alsu R. Sayapova and Anton V. Shokurov. Thank you!

*Aleksandr Dekhovich*  
*Delft, May 2024*





# 1

## INTRODUCTION

*This chapter introduces the catastrophic forgetting problem in deep neural networks. We describe continual learning as a field that tackles this problem under different scenarios. A general paradigm for the continual learning algorithm is described in this chapter. Then we formulate the main research goal of the thesis and research questions that help to achieve the goal. Finally, the outline of the thesis is presented.*

Artificial Intelligence (AI) aims to create algorithms and software that imitate the intelligence of humans. There are various subfields in AI aiming to build smart systems that can learn and behave as natural intelligence. One is machine learning (ML), which incorporates observed data to train complex statistical models [1]. Machine learning has become a powerful tool to solve a variety of real-life problems in engineering, finance, medicine and physics. In the last few years, there has been a particular interest in deep neural networks (DNNs), i.e. deep learning (DL) models [2–4].

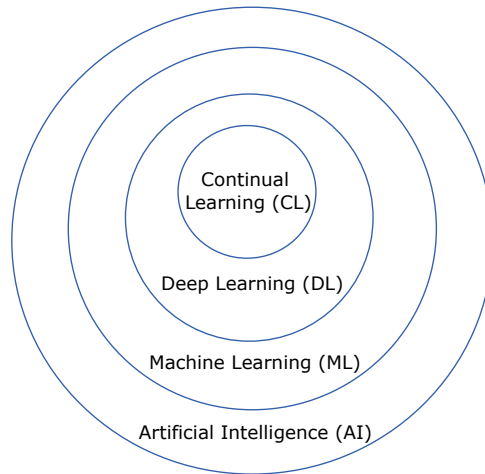


Figure 1.1: Artificial Intelligence and its sub-areas.

Despite the popularity of DNNs, they fail to learn tasks sequentially [5]. This occurs because DNNs tend to forget old information while learning new one. This phenomenon is called *catastrophic forgetting* [6], and *continual learning* (CL) is the subfield of deep learning that focuses on this problem. Figure 1.1 summarizes the relation between AI, ML, DL and CL.

The development of continual learning models improves their robustness to the real-world non-stationary environments. For instance, an autonomous vehicle needs to constantly perform scene recognition incrementally adapting to new conditions without forgetting previous knowledge [7]. Another example is a medical system that continuously learns to segment different regions of the human brain from MRI scans without retraining because past data can be too sensitive to store on a server [8, 9]. In the following sections, we will discuss the reasons for catastrophic forgetting and different problem scenarios. This thesis aims to propose a new paradigm for overcoming catastrophic forgetting in DNNs by taking inspiration from neuroscience literature and encoding different tasks in separate subregions, activating a proper one at inference.

## 1.1 CONTINUAL LEARNING FOR DEEP NEURAL NETWORKS

Machine learning systems, as well as biological ones, suffer from forgetting [6, 10]. However, DNNs-based models may suffer from catastrophic forgetting even if there are only two tasks [11]. In a scenario where the network is trained sequentially, the parameters trained

for the first task will be updated relative to the second, significantly losing the ability to perform the previous one [10]. This is the outcome of the gradient-based weights update rule, where every parameter of the neural network is only updated based on the gradient of the loss on the current task without considering previous tasks [11], in contrast to multi-task learning [12]. In neuroscience, it is also known as *the plasticity-stability dilemma* [13, 14]. The illustration of this process is shown in Figure 1.2.

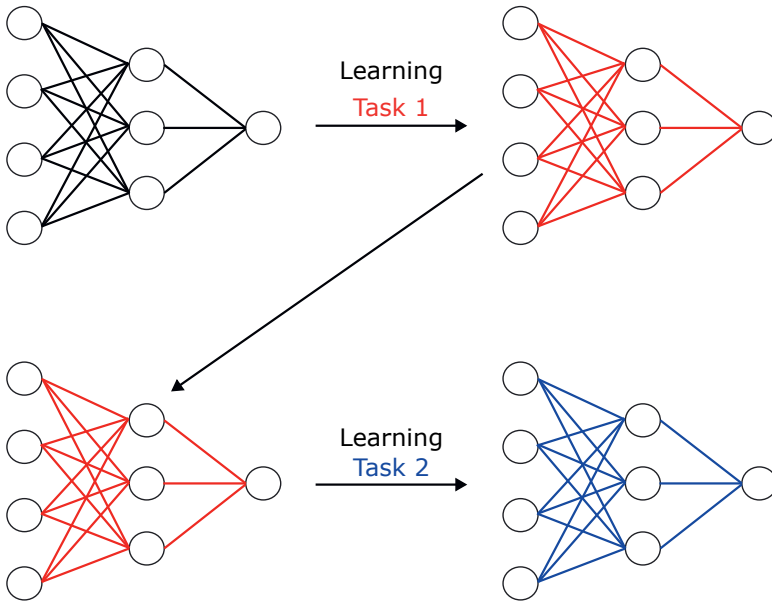


Figure 1.2: An illustration of parameters overwriting in sequential training.

In continual (or lifelong) learning, data comes in batches  $\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_t, \dots$  sequentially, which are called tasks. In the moment  $t$ , only data  $\mathcal{D}_t$  for task  $t$  is available, while all previous batches  $\mathcal{D}_0, \mathcal{D}_1, \dots, \mathcal{D}_{t-1}$  are not. Continual learning aims at creating deep learning models that do not forget previously learned tasks while being able to learn new ones. Addressing current challenges in continual learning and mitigating catastrophic forgetting brings the deep learning community one step closer to mimicking the capabilities of the human brain.

Continual learning literature mostly focuses on algorithms for computer vision or natural language processing tasks. Yet, developing a general method not limited to one data type is essential, albeit less common in the current literature. Also, modern approaches do not focus much on the influence of one task in learning the following ones. However, the human brain can take advantage of already-acquired information facilitating the learning process and requiring fewer training examples to learn a pattern [15].

### 1.1.1 DIFFERENT CONTINUAL LEARNING SCENARIOS

The notation of the task refers to a separate batch of training data  $\mathcal{D}_t = (X^{(t)}, Y^{(t)})$  available at the current step  $t$ , where  $X^{(t)}$  is input data and  $Y^{(t)}$  are corresponding labels. This

can be a set of new classes, domains, or output space [16]. Continual learning literature distinguishes several scenarios for classification problem [16, 17], which are formalized with the use of marginal distributions  $P(X^{(t)})$ ,  $P(X^{(t)}) \neq P(X^{(t+1)})$ , and the availability of task-ID:

- Task-incremental learning (Task-IL): the easiest CL scenario, where task-ID is given during training and inference. Tasks have non-overlapping class labels and  $\{Y^t\} \neq \{Y^{t+1}\}$ . A real-life example is learning mechanical law, e.g. stress-strain relation, for different material types. This is a task-IL problem since at the inference stage we know for which material we want to predict stress with the given strain (see also Chapter 6).
- Domain-incremental learning (Domain-IL): task-ID is unavailable during inference. However, the model does not need to infer what task input data belongs to and has a fixed set of output neurons, meaning  $\{Y^t\} = \{Y^{t+1}\}$ , but the input distribution changes over time. In other words, tasks always have the same structure, but there is a distribution shift in input space. One example of domain-incremental learning is an autonomous agent driving under different weather conditions [18].
- Class-incremental learning (Class-IL): the most challenging scenario, where task-ID is not given at the inference stage. The output layer consists of separate task-specific classification heads  $\{Y^t\} \subset \{Y^{t+1}\}$ . A typical example of the class-IL problem is a monitoring system that learns to classify different types of defects while they occur.

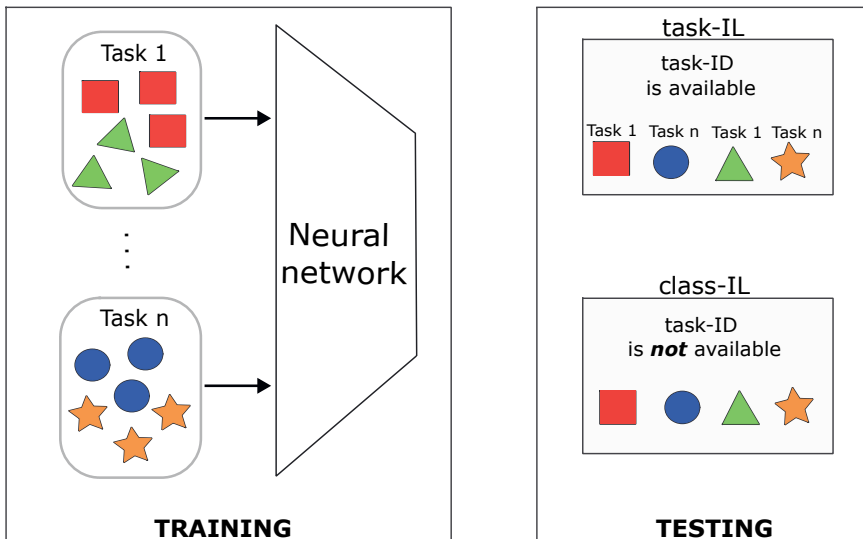


Figure 1.3: An illustration of the difference between task-IL and class-IL scenarios.

However, in this thesis, we will discuss only task-IL and class-IL scenarios in case of classification problems, following the categorization of Masana et al. [19], and scenarios

with or without provided task-ID for regression and segmentation problems. In Figure 1.3, we show a schematic with the difference between task-IL and class-IL scenarios for classification.

## 1.2 GENERAL PARADIGM

In this section, we present the basic paradigm proposed in the thesis. Drawing inspiration from the neuroscience literature and some evidence about how the human brain works [20, 21], we propose dividing the network into task-related regions, which in this case are task-specific subnetworks. Thus, during training, the algorithm strives to find these subnetworks for specific tasks, and during testing, it is necessary to select a suitable subnetwork and make a prediction using it.

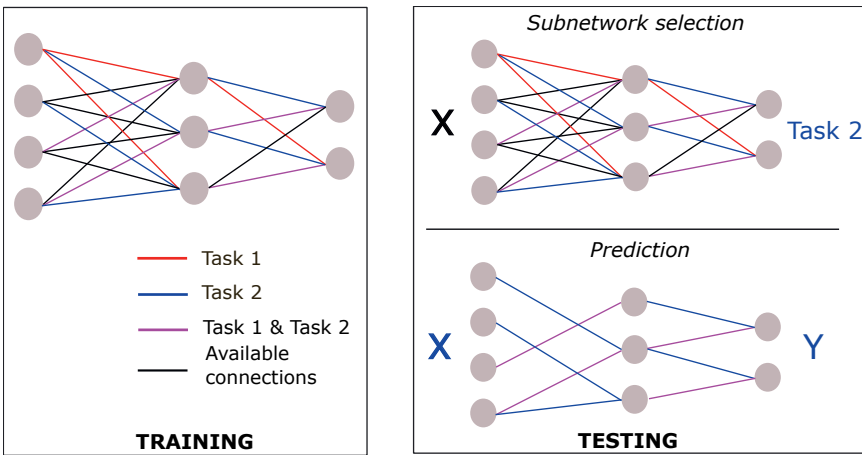


Figure 1.4: An overview of the proposed CL paradigm.

Figure 1.4 summarizes this paradigm using the example with two tasks. We demonstrate it in the most general case of the class-IL scenario. Here the model first has to predict the task from which  $X$  came, and then make a prediction  $Y$  with the corresponding subnetwork.

### 1.2.1 RESEARCH GOAL AND RESEARCH QUESTIONS

The research goal is to build a DNN without catastrophic forgetting that can be applied to a variety of problems and data types, e.g., images and time series. As described previously, we create specialized regions in a network (task-specific subnetworks) that are responsible for solving a particular task. Moreover, if tasks are similar, we want previously acquired knowledge to help the model master new tasks with better performance. Overall, the research goal (RG) can be formulated as follows:

Develop a neural network model that can solve several tasks incrementally, promoting different regions and training the model to use learned concepts for solving new ones.

We formulate several research questions (**RQ**) that should help us to achieve the **RG** and discuss the importance of these questions:

- **RQ1:** How to create a task-specific subnetwork?

First, we should understand which connections and neurons are the most significant in DNN for solving a given task. By estimating the importance of each parameter, we will be able to prune the connections that do not contribute much to the final output and retain only the useful ones.

- **RQ2:** How to train subnetworks sequentially without forgetting?

Once we can build these task-specific subnetworks separately, we need to figure out how to build them sequentially. The problem here is that subsequent subnetworks can interfere with all previous ones, causing forgetting.

- **RQ3:** What is the mechanism for activating the correct subnetwork during inference?

At the inference stage, DNN receives the test data point. In task-IL scenario, the model also knows the task-ID for the data point and, therefore, which subnetwork to use. However, in a more general class-IL case, task-ID should be inferred by the algorithm to use the correct subnetwork.

- **RQ4:** Can knowledge transfer between the regions improve generalization and/or reduce training data needs?

The created subnetworks make parts of the DNN pretrained on previous tasks. This may facilitate the learning of new tasks if the model reuses the connections pretrained on similar tasks. As a result, this transfer learning effect may improve the performance of the model on new tasks without forgetting old ones or reduce the need for training data for future tasks.

### 1.3 THESIS OUTLINE

The thesis consists of two parts: Methodology (Chapters 2, 3 and 4) and Application (Chapters 5, 6). In the first part, we develop the general approach to tackle continual learning problems. In Chapter 2, we present a data-driven neural network pruning algorithm [22] that creates a task-specific subnetwork (region) for the given dataset. We validate this pruning strategy on various image-based datasets such as MNIST and CIFAR-10/100 outperforming other state-of-the-art methods. In Chapter 3, we utilize the developed pruning algorithm to create sequentially task-specific subnetworks in the class-IL scenario. Then we propose two strategies to infer task-ID to select the associated subnetwork. However, these strategies require access to a batch of test data for a more accurate determination of task-ID. Chapter 5 presents an incremental learning approach for the defect segmentation problem, where we improve the task prediction strategy and overcome the limitation of the test batch that is raised in Chapter 3.

The second part of the thesis explores opportunities for continual learning in the physical sciences and engineering. In both cases, we assume that task-ID is provided at the training and inference stages and there is no need to predict it. In Chapter 5, we propose an incremental learning approach for physics-informed neural networks (PINNs) [23].

In this algorithm, we take advantage of task-specific subnetworks to learn every partial differential equation (PDE) with a separate subnetwork. This way, part of the network is already pretrained with previous PDEs, which improves the generalization capabilities of PINNs. However, due to the specifics of PINNs, training data is always available for every PDE since one can easily sample training points of the domain. Chapter 6 describes the application of the developed in the first part methodology to the stress-strain constitutive modeling problem. In this chapter, we consider the scenario where task-ID is provided at inference, meaning we know which model we need to use for the given strain values.





# PART I: METHODOLOGY



## 2

2

## CREATION OF TASK-SPECIFIC SUBNETWORKS

*Deep neural networks are highly overparameterized in conventional training and use all of their parameters for learning a task. However, the human brain activates only a small portion of all connections that process new information for learning. Thus, this chapter proposes an approach to sparsify a neural network for the given task, i.e. identify a task-specific subnetwork that is responsible for solving this task. We propose an iterative pruning strategy introducing a simple importance-score metric that deactivates unimportant connections, tackling overparameterization in DNNs and modulating the firing patterns. The aim is to find the smallest number of connections that is still capable of solving a given task with comparable accuracy, i.e. a simpler subnetwork. This subnetwork is then associated with the task, while the remaining parameters are freed for others. The proposed strategy is validated on various image datasets such as MNIST, CIFAR-10/100 and Tiny-ImageNet outperforming other state-of-the-art pruning strategies by a margin in terms of the number of removed connections. We also explore the influence of different training hyperparameters on the level of produced sparsity.*

## 2.1 INTRODUCTION

Creating task-specific subnetworks requires the understanding of which parameters are useful for the given data and which can be removed. For this purpose, we turn to the literature on optimization of neural network architecture. One way to optimize the architecture is to build it from scratch via Neural Architecture Search (NAS) [24], however, NAS-based approaches are very computationally expensive [25]. Therefore, we follow a different route in this thesis using neural network pruning.

Pruning is a common technique for neural network compression [26], where the main goal is to reduce memory and computational costs of inference. Pruning assumes particular relevance for deep neural networks because modern architectures involve several millions of parameters. Existing pruning methods are based on different strategies, e.g Hessian analysis [27, 28], magnitudes of weights [29], data-driven approaches [30, 31], among others [32, 33]. Pruning can be done in one shot [34] or in an iterative way [35], and it is possible to prune connections [27, 28, 36, 37], neurons [31, 38, 39] or filters for convolutional layers [40, 41]. The typical pruning pipeline includes three stages: training the original network, pruning parameters and fine-tuning. Recently, interesting solutions for the third stage have been suggested that involve weight rewinding [35] and learning rate rewinding [42].

We have developed an algorithm that aims to prune a network without a significant decrease in accuracy after every iteration by keeping the signal in the network at some predefined level close to the original one. This contrasts with strategies that use a predefined ratio of parameters to prune [30, 31, 35] which may lead to a drastic drop in accuracy for relatively high ratios. We look at the local behaviour of a particular connection and its contribution to the neuron, but not at the output or the loss function. Our aim is to deactivate unimportant connections for a given problem in order to free them for other tasks – a crucial step towards novel architectural continual learning strategies [43, 44]. Sparse architectures are also more robust to noisy data [45] and exhibit benefits in the context of adversarial training [46, 47]. Therefore, we focus on reducing the number of parameters of deep learning models targeting these scenarios, although we expect a reduction of the computational complexity (FLOPs) as well.

Our iterative pruning algorithm is based on an importance score metric proposed herein that quantifies the relevance of each connection to the local neuron behaviour. We show compressions of more than 50 times for VGG [3] architectures on CIFAR-10 and Tiny-ImageNet datasets with a marginal drop of accuracy. We also apply our method to ResNets [4], achieving better parameter compression than state-of-the-art algorithms with a comparable decrease of accuracy on CIFAR-10 dataset. In addition, we visualise the effects of our pruning strategy on the information propagation through the network, and we observe a significant homogenization of the importance of the pruned neurons. We associate this homogenization with the notion of neural network relief: using fewer neuronal connections and distributing importance among them.

## 2.2 RELATED WORK

One of the first works eliminating unimportant connections in relatively small networks proposed analyzing the Hessian of the loss function [27, 28] without network retraining. This idea was further developed for convolutional networks [36, 48]. However, computing

second-order derivatives by calculating the Hessian is costly.

The magnitude-based approach [29] is simple and fast because it only involves pruning the weights with the smallest magnitude. This assumes that parameters with small magnitudes do not contribute significantly to the resulting performance. For convolutional layers, the sum of kernels' elements in the filters is considered and filters with the smallest sum are pruned [49]. Iterative magnitude pruning is applied for finding winning tickets [35] – the minimal subnetwork that can be trained at least as well as the original one with the same hyperparameters.

Other algorithms use input data to reduce the number of parameters. In [30] pruning depends on the ratio of zero activations using ReLU function. So, the neurons that do not fire frequently enough are eliminated. A greedy approach is used for ThiNet [50], where channels that do not affect the resulting sum in convolutional layers are removed by solving an optimization problem. A similar algorithm is used in [40], but the optimization problem is solved with LASSO regression for determining ineffective channels. The property of convexity and sparsity that ReLU produces provides analytical boundaries for Net-trim [51] by solving a convex optimization problem. The game-theoretic approach with Shapley values [31] demonstrates good performance in a low-data regime, i.e. one-shot pruning without retraining, where neurons within one layer are considered as players in a cooperative game. NISP [52] finds the contribution of neurons to the last layer before classification, while iSparse framework [53] trains sparse networks eliminating the connections that do not contribute to the output. A pruning strategy where the connections' contribution is based on computing the loss function derivatives was also suggested in [54].

Additional methods of interest include Bayesian weight pruning [33] and Bayesian compression [39] to prune neurons and aim at computational efficiency. SSS [55] introduces a scaling factor and sparsity constraints on this factor to scale neurons' or blocks' outputs. Similarly, SNLI [56] uses ISTA [57] to update the scaling parameter in Batch Normalization of convolutional layers to obtain a sparse representation. HRank [58] calculates the average rank of feature maps and prunes filters with the lowest ones. FSABP [59] finds filters that extract similar information and prunes them. As a result, filters that provide diversity in feature maps are retained. In the same way, CHIP [60] measures correlations between feature maps truncating feature maps with the lowest independence scores.

## 2.3 PROPOSED METHOD

Our goal is to eliminate connections in a layer that, on average, provide a weak contribution to the next layer. We believe that a low magnitude of a weight does not mean that information passing through the connection has to have a negligible contribution. The converse is also assumed to be true: if a high weight magnitude is multiplied by a weak signal from the neuron (or even zero) then the contribution of that product is relatively insignificant in comparison with other input signals in the neuron, as long as this holds for most data. Therefore, our strategy contrasts significantly with magnitude-based pruning [29].

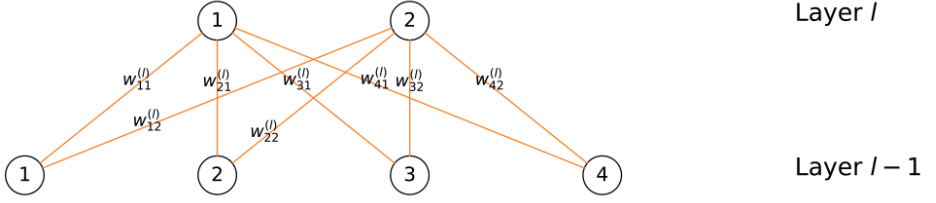


Figure 2.1: Neural network layers  $l-1$  and  $l$  with  $m_{l-1} = 4$  and  $m_l = 2$  neurons, respectively. The weights associated to a connection between neurons  $i$  and  $j$  in layer  $l-1$  to  $l$  are  $w_{ij}^{(l)}$ .

**Fully connected layers** Assume that we have a pruning set  $\mathbf{X}^{(l-1)} = \{\mathbf{x}_1^{(l-1)}, \dots, \mathbf{x}_N^{(l-1)}\}$  with  $N$  samples, where each datapoint  $\mathbf{x}_n^{(l-1)} = (x_{n1}^{(l-1)}, \dots, x_{nm_{l-1}}^{(l-1)}) \in \mathbb{R}^{m_{l-1}}$  is the input for layer  $l-1$  with dimension  $m_{l-1}$  and where  $1 \leq l \leq L$ . We define the *importance* of the connection between neuron  $i$  of layer  $l-1$  and neuron  $j$  of layer  $l$  as:

$$s_{ij}^{(l)} = \frac{\overline{w_{ij}^{(l)} x_i^{(l-1)}}}{\sum_{k=1}^{m_{l-1}} \overline{w_{kj}^{(l)} x_k^{(l-1)}} + |b_j^{(l)}|}, \quad (2.1)$$

where  $\overline{w_{ij}^{(l)} x_i^{(l-1)}} = \frac{1}{N} \sum_{n=1}^N |w_{ij}^{(l)} x_{ni}^{(l-1)}|$ , and  $w_{ij}^{(l)}$  is the corresponding weight between neurons  $i$  and  $j$  (see Figure 2.1) and  $b_j^{(l)}$  is the bias associated to neuron  $j$ . The importance score for the bias of neuron  $j$  is  $s_{m_{l-1}+1,j}^{(l)} = \frac{|b_j^{(l)}|}{\sum_{k=1}^{m_{l-1}} \overline{w_{kj}^{(l)} x_k^{(l-1)}} + |b_j^{(l)}|}$ . The denominator corresponds to the total importance in the neuron  $j$  of layer  $l$  that we denote as  $S_j^{(l)} = \sum_{k=1}^{m_{l-1}} \overline{w_{kj}^{(l)} x_k^{(l-1)}} + |b_j^{(l)}|$ ,  $1 \leq j \leq m_l$ . Algorithm 1 summarizes the procedure.

---

**Algorithm 1** Fully connected layers pruning

---

```

1: function FC_PRUNING(network, X,  $\alpha$ )
2:    $\mathbf{X}^{(0)} \leftarrow \mathbf{X}$ 
3:   for every fc_layer  $l$  in FC_Layers do
4:      $\mathbf{X}^{(l)} \leftarrow \text{fc\_layer}(\mathbf{X}^{(l-1)})$ 
5:     for every neuron  $j$  in fc_layer  $l$  do
6:       compute importance scores  $s_{ij}^{(l)}$  for every incoming connection  $i$  using (2.1).
7:        $\hat{s}_{ij}^{(l)} = \text{Sort}(s_{ij}^{(l)}, \text{order} = \text{descending})$ 
8:        $p_0 = \min\{p : \sum_{i=1}^p \hat{s}_{ij}^{(l)} \geq \alpha\}$ 
9:       prune connections with importance score  $s_{ij}^{(l)} < \hat{s}_{p_0 j}^{(l)}$ 
10:    end for
11:  end for
12:  return pruned network
13: end function

```

---

The idea behind our approach is to deactivate the connections  $i = 1 \dots, m_{l-1}$  that on average do not carry important information to the neuron  $j$  of layer  $l$  in comparison with other connections. As a result of Eq.1, we simplify each neuron by using a smaller number of input connections, while causing minimal effective changes to the input signal of that neuron and subsequent minimal impact on the network.

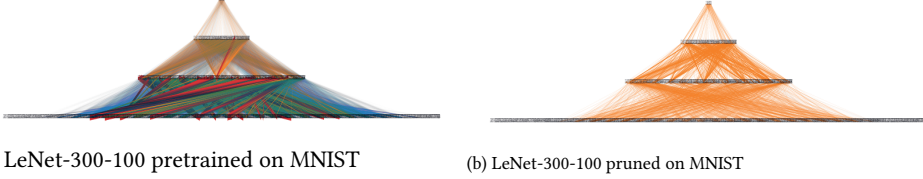


Figure 2.2: LeNet-300-100 architecture on MNIST before and after pruning, where connections are coloured with respect to importance score: blue (least important)  $\rightarrow$  red (most important).

Figure 2.2 shows the importance score of every connection in the LeNet-300-100 [61] architecture before and after pruning when applied to the MNIST dataset. Connections are represented by thin blue lines when they have the lowest importance score  $s_{ij}^{(l)}$ , and go up to red thick lines when they are the most important. The figure demonstrates an interesting phenomenon besides the fact that there are much fewer connections after applying our pruning strategy: the importance scores of the connections become much closer to each other and there are no longer highly important and highly unimportant connections. We eliminate connections (contributors to the neuron’s signal) that on average do not contribute in terms of the strength of the signal that they bring to the neuron for a given dataset.

**Convolutional layers** Our pruning approach for convolutional layers is similar to the one conducted on fully connected layers. We consider kernels and a bias in a particular filter as contributors to the signal produced by this filter.

Assume we have  $m_{l-1}$ -channelled input samples  $\mathbf{X}^{(l-1)} = \{\mathbf{x}_1^{(l-1)}, \dots, \mathbf{x}_N^{(l-1)}\}$ , where  $\mathbf{x}_k^{(l-1)} = (x_{k1}^{(l-1)}, \dots, x_{km_{l-1}}^{(l-1)}) \in \mathbb{R}^{m_{l-1} \times h_{l-1}^1 \times h_{l-1}^2}$ , where  $h_{l-1}^1$  and  $h_{l-1}^2$  are the height and width of input images (or feature maps) for convolutional layer  $l$ . For every kernel  $\mathbf{K}_{1j}^{(l)}, \mathbf{K}_{2j}^{(l)}, \dots, \mathbf{K}_{m_{lj}}^{(l)}$ ,  $\mathbf{K}_{ij}^{(l)} = (k_{ijqt}^{(l)}) \in \mathbb{R}^{\eta \times \eta}$ ,  $1 \leq q, t \leq \eta$ ,  $\eta$  is a kernel size, and a bias  $b_j^{(l)}$  in filter  $\mathbf{F}_j^{(l)}$ , we define  $\hat{\mathbf{K}}_{ij}^{(l)} = \left( |k_{ijqt}^{(l)}| \right)$  as a matrix consisting of the absolute values of the matrix  $\mathbf{K}_{ij}^{(l)}$ .

Then we compute importance scores  $s_{ij}^{(l)}, i \in \{1, 2, \dots, m_l\}$  of kernels  $\mathbf{K}_{ij}^{(l)}$  as follows:

$$s_{ij}^{(l)} = \frac{\frac{1}{N} \sum_{n=1}^N \left\| \hat{\mathbf{K}}_{ij}^{(l)} * \left| x_{ni}^{(l-1)} \right| \right\|_F}{S_j^{(l)}}, \quad (2.2)$$

$$s_{m_l+1, j}^{(l)} = \frac{|b_j^{(l)}| \sqrt{h_l^1 h_l^2}}{S_j^{(l)}}. \quad (2.3)$$



where  $S_j^{(l)} = \sum_{i=1}^{m_{l-1}} \left( \frac{1}{N} \sum_{n=1}^N \left\| \hat{\mathbf{K}}_{ij}^{(l)} * \mathbf{x}_{ni}^{(l-1)} \right\|_F \right) + |b_j^{(l)}| \sqrt{h_i^1 h_l^2}$  is the total importance score in filter  $\mathbf{F}_j^{(l)}$  of layer  $l$ , and where  $*$  indicates a convolution operation, and  $\|\cdot\|_F$  the Frobenius norm.

In Eq. 2.2, we compute the amount of information that every kernel produces on average, analogously to what we do in fully connected layers. Algorithm 2 summarizes the approach for convolutional layers.

---

**Algorithm 2** Convolutional layers pruning
 

---

```

function CONV_PRUNING(network, X,  $\alpha$ )
2:    $\mathbf{X}^{(0)} \leftarrow \mathbf{X}$ 
   for conv layer  $l$  in CONV_Layers do
4:    $\mathbf{X}^{(l)} \leftarrow \text{conv\_layer}(\mathbf{X}^{(l-1)})$ 
   for every filter  $\mathbf{F}_j$  in conv_layer  $l$  do
6:     compute importance scores  $s_{ij} \forall$  kernel  $\mathbf{K}_{ij}^{(l)}$  and bias  $b_j^{(l)}$  in filter  $\mathbf{F}_j^{(l)}$ 
       using (2.2), (2.3).
        $\hat{s}_{ij}^{(l)} = \text{Sort}(s_{ij}^{(l)}, \text{order} = \text{descending})$ 
8:      $p_0 = \min\{p : \sum_{i=1}^p \hat{s}_{ij}^{(l)} \geq \alpha\}$ 
       prune kernel  $\mathbf{K}_{ij}^{(l)}$  with importance score  $s_{ij}^{(l)} < \hat{s}_{p_0 j}^{(l)}$ 
10:    end for
   end for
12:  return pruned network
end function

```

---

## 2.4 NUMERICAL EXPERIMENTS

We test our pruning method for LeNet-300-100 and LeNet-5 [61] on MNIST, VGG-19 [3] and VGG-like [62] on CIFAR-10/100 [63] and Tiny-ImageNet datasets. We evaluate our method on classification error, the percentage of pruned parameters  $\left( \frac{|w=0|}{|w|} (\%) \right)$  or the percentage of remaining parameters  $\left( \frac{|w \neq 0|}{|w|} (\%) \right)$  or compression rate  $\left( \frac{|w|}{|w \neq 0|} (\%) \right)$  and pruned FLOPs (floating-point operations), where  $|w \neq 0|$  refers to the number of unpruned connections. For the details about training and pruning hyperparameters, and FLOPs computation see Section A.1 and Section A.2. We use the initialization method introduced in [64] to initialize parameters. We run our experiments multiple times with different random initializations of the parameters. In our tables and figures, we present mean and standard deviation for the results averaged over these random initializations.

### 2.4.1 LENET ARCHITECTURE

We experiment with LeNets on the MNIST dataset. LeNet-300-100 is a fully connected network with 300 neurons and 100 neurons in two hidden layers respectively, and LeNet-5 Caffe, which is a modified version of [61], has two convolutional layers followed by one hidden layer and an output layer. We perform iterative pruning by retraining the network starting from initial random parameters, instead of fine-tuning. Table 2.1 shows that our approach is among the best for both LeNets in terms of pruned parameters.

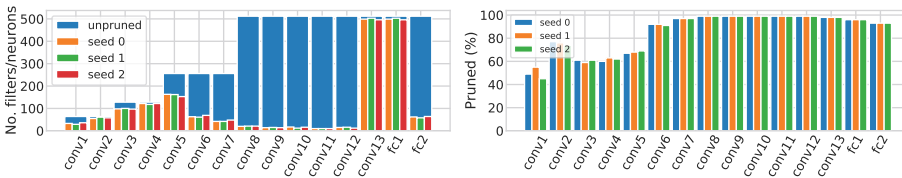
Table 2.1: Results for LeNet-300-100 and LeNet-5 trained and pruned on MNIST. For pruning, 1000 random training samples are chosen,  $\alpha_{fc} = 0.95$ ,  $\alpha_{conv} = 0.9$ .

Network	Method	Error (%)	Parameters retained (%)
LeNet-300-100	DNS [65]	1.99	1.79
	L-OBS [36]	1.96	1.5
	SWS [32]	1.94	4.3
	Sparse VD [33]	<b>1.92</b>	<b>1.47</b>
	<b>NNrelief (ours)</b>	$1.98 \pm 0.07$	$1.51 \pm 0.07$
LeNet-5	DNS [65]	0.91	0.93
	L-OBS [36]	1.66	0.9
	SWS [32]	0.97	0.5
	Sparse VD [33]	<b>0.75</b>	<b>0.36</b>
	<b>NNrelief (ours)</b>	$0.97 \pm 0.05$	$0.65 \pm 0.02$

## 2.4.2 VGG ARCHITECTURE

We perform our experiments on VGG-13 and VGG-like (adapted version of VGG-16 for CIFAR-10 dataset that has one fully connected layer less) on CIFAR-10, CIFAR-100 and Tiny-ImageNet datasets. We also show in this section the effect of two optimizers – Adam [66] and SGD and two weight decay values on our pruning technique.

**CIFAR-10/100** Table 2.2 shows that after the reduction of the weights to less than 2% of the original number, the accuracy only drops by 0.1%, and that our approach outperforms others both in terms of pruned parameters and pruned FLOPs. In Figure 2.3, we show the final architecture after pruning and the level of sparsity by layer. It can be observed that NNrelief with Adam optimizer achieves a high level of filter sparsity even though it prunes kernels.



(a) An architecture before and after pruning by seed.

(b) Sparsity by layer after pruning using different seeds.

Figure 2.3: Architecture structure for VGG-like on CIFAR-10 with Adam optimizer considering three random initializations.

We also perform pruning for standard VGG-13 on CIFAR-100 using 5 iterations, and considered three randomization seeds to provide reasonable statistical significance. The results from the different optimizers are presented in Figure 2.4. The results show that Adam seems to perform the compression more aggressively, resulting in a higher compression rate, and also a slightly lower accuracy. The compression of about 14.5 $\times$  after the fifth

Table 2.2: Results for VGG-like trained on CIFAR-10 with Adam optimizer. During retraining 60 epochs are used; the learning rate is decreased by 10 every 20 epochs. For NNrelief (ours)  $\alpha_{\text{conv}} = \alpha_{\text{fc}} = 0.95$  and 1000 samples are chosen for the importance scores computation. Sign ‘-’ means that accuracy has increased after pruning.

Network	Method	Acc (%) (base- line)	Acc drop (%)	Parameters retained (%)	FLOPs pruned (%)
VGG-like	Pruning [49]	93.25	-0.15	36	34.2
	Sparse VD [33]	92.3	0.0	2.1	N/A
	BC-GNJ [39]	91.6	0.2	6.7	55.6
	BC-GHS [39]	91.6	0.6	5.5	61.7
	SNIP [54]	91.7	<b>-0.3</b>	3.0	N/A
	HRank [58]	93.96	1.62	17.9	65.3
	CHIP [60]	93.96	0.24	16.7	66.6
	<b>NNrelief (ours)</b>	92.5	0.1	<b>1.92 ± 0.02</b>	<b>75.5 ± 0.4</b>

iteration without loss in accuracy is presented for Adam and weight decay  $5 \cdot 10^{-4}$ , while SGD allows to train a network with higher accuracy, but after 5 iterations we obtain smaller compression (about 4.5 $\times$ ) and the decrease of accuracy by 0.5% with the same weight decay. We obtain a lower compression rate for both optimizers when the weight decay is lower ( $10^{-4}$ ). Figure 2.5 shows the final architecture and sparsity that we achieve with Adam optimizer at the end of the pruning procedure. We observe a high level of filter sparsity with Adam, as also reported in [67], but good compression is also obtained for the SGD optimizer.

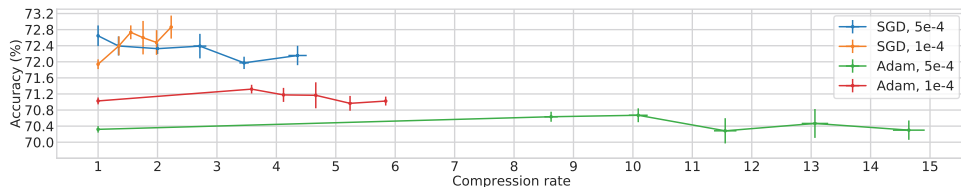


Figure 2.4: Results for VGG-13 over three seeds; mean values are used to compute dots and standard deviation are shown with error bars. We compare two optimizers, SGD and Adam, and two different values of weight decay for evaluation after 5 pruning iterations.

**Tiny-ImageNet** We also apply our approach to Tiny-ImageNet dataset that is a subset of the ImageNet dataset with 200 classes and an image spatial resolution of  $64 \times 64$ . Following SNIP strategy, we use strides [2,2] in the first convolutional layer to reduce the size of images. The results are reported in Table 2.3. We observe a compression by more than 40 times almost without any loss of accuracy ( $-0.03\%$ ) using Adam optimizer.

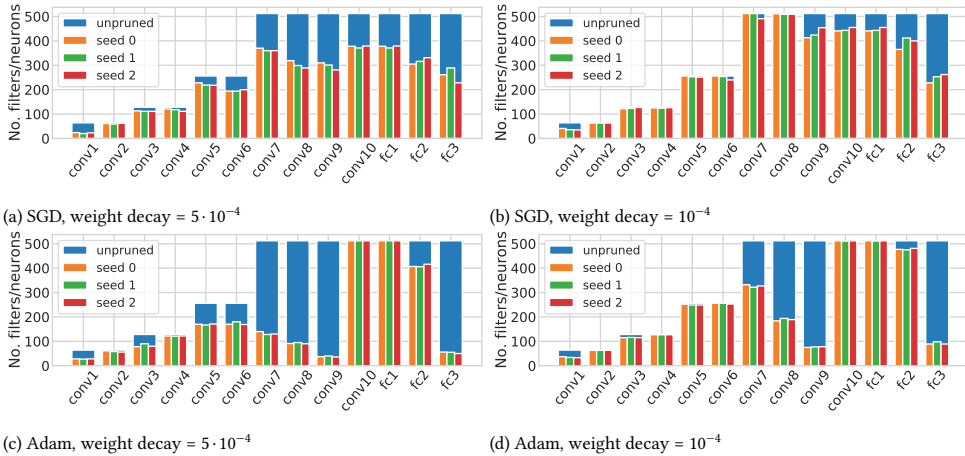


Figure 2.5: VGG-13 architecture on CIFAR-100 trained with SGD (top) and Adam (bottom), and weight decay equal to  $5 \cdot 10^{-4}$  (left) and  $10^{-4}$  (right) and pruned with 5 iterations. The results for three different seeds are presented.

Table 2.3: Results for VGG-like trained on Tiny-ImageNet with Adam optimizer. During retraining 50 epochs are used; the learning rate is decreased by 10 every 20 epochs. For NNrelief (ours)  $\alpha_{\text{conv}} = \alpha_{\text{fc}} = 0.95$  and 2000 samples are chosen for the importance scores computation.

Network	Method	Acc (%) (base- line)	Acc drop (%)	Parameters retained (%)	FLOPs pruned (%)
VGG- like	SNIP [54]	45.14	0.87	5.0	N/A
	<b>NNrelief (ours)</b>	45.63	0.03	<b>2.32 ± 0.06</b>	<b>75 ± 0.32</b>

### 2.4.3 RESNET ARCHITECTURE

In addition, we test our approach on ResNet architectures for CIFAR-10/100 and Tiny-ImageNet datasets. Our main objective is to prune as many parameters as we can without significant loss of accuracy. For ResNet-20/56 on CIFAR-10, we perform iterative pruning over 10 iterations, training the model with SGD and Adam. In Table 2.4 we compare our results with other approaches, and Figure 2.6 displays the pruning history. We achieve a higher percentage of pruned parameters for ResNet-20 and ResNet-56, using both optimizers with comparable final accuracy.

A notable observation for some seeds when training with SGD is that separate convolutional blocks are pruned and only residual blocks remain, i.e. the signal propagates through skip connections and not through main convolutional layers since they do not contribute to the sum (Figure 2.8).

For ResNet-20/56 on CIFAR-100 the results are presented in Table 2.5 and by iteration in Figure 2.7.

Table 2.4: Results for ResNets trained on CIFAR-10 with SGD and Adam optimizers. During retraining 60 epochs are used; the learning rate is decreased by 10 every 20 epochs. For NNrelief (ours)  $\alpha_{\text{conv}} = 0.95, \alpha_{\text{fc}} = 0.99$ .

ResNet	Method	Acc (%) (baseline)	Acc drop (%)	Parameters pruned (%)	FLOPs pruned (%)
20	SFP [41]	92.2	1.37	30	42.2
	SNLI [56]	92.0	1.1	37.2	N/A
	SSS [55]	92.8	2	45	<b>60</b>
	CNN-CFC [68]	92.2	1.07	42.75	41.6
	<b>NNrelief (SGD)</b>	$92.25 \pm 0.12$	$1.15 \pm 0.13$	<b><math>63.68 \pm 1.52</math></b>	$25.85 \pm 1.57$
	<b>NNrelief (Adam)</b>	$91.83 \pm 0.16$	<b><math>0.39 \pm 0.27</math></b>	<b><math>68.75 \pm 1.26</math></b>	$13.81 \pm 2.6$
56	Pruning-B [49]	93.04	-0.02	13.7	27.6
	SFP [41]	93.59	-0.19	30	41.1
	NISP [52]	N/A	0.03	42.6	43.61
	CNN-CFC [68]	93.14	<b>-0.24</b>	43.09	42.78
	HRank [58]	93.36	0.09	42.4	50
	CHIP [60]	93.26	1.21	71.8	72.3
	<b>NNrelief (SGD)</b>	$93.6 \pm 0.08$	$1.14 \pm 0.22$	<b><math>76.2 \pm 0.44</math></b>	$35.8 \pm 2.69$
	<b>NNrelief (Adam)</b>	$92.8 \pm 0.08$	$0.03 \pm 0.23$	<b><math>75.95 \pm 0.22</math></b>	$32.5 \pm 0.31$

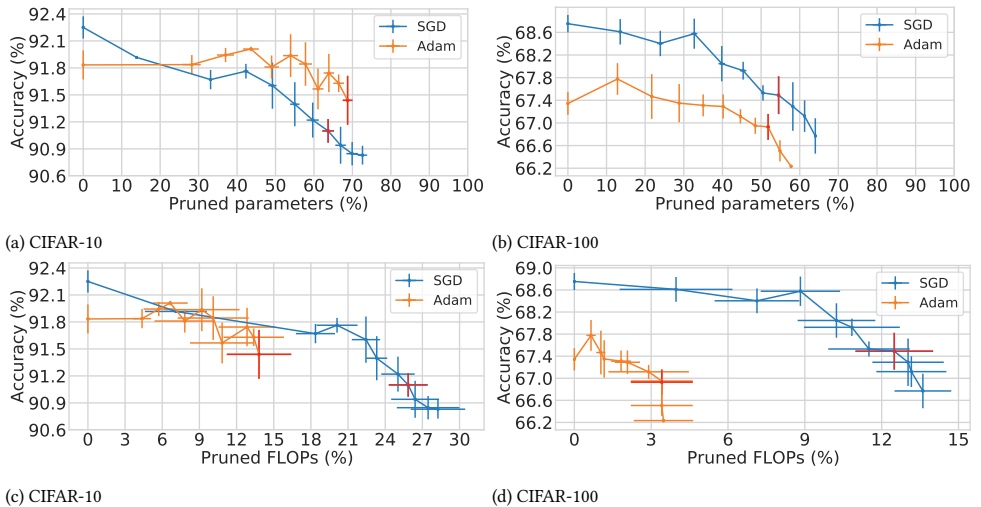


Figure 2.6: ResNet-20 on CIFAR-10 and CIFAR-100 by iteration averaged over three random initializations, the error bars represent standard deviation for these three runs. The red dot corresponds to the selected best iteration.

**Tiny-ImageNet** Training of a modified ResNet-18 with 16, 32, 64 and 128 output channels indicates that we can prune more than 50% of the parameters with both optimizers (see

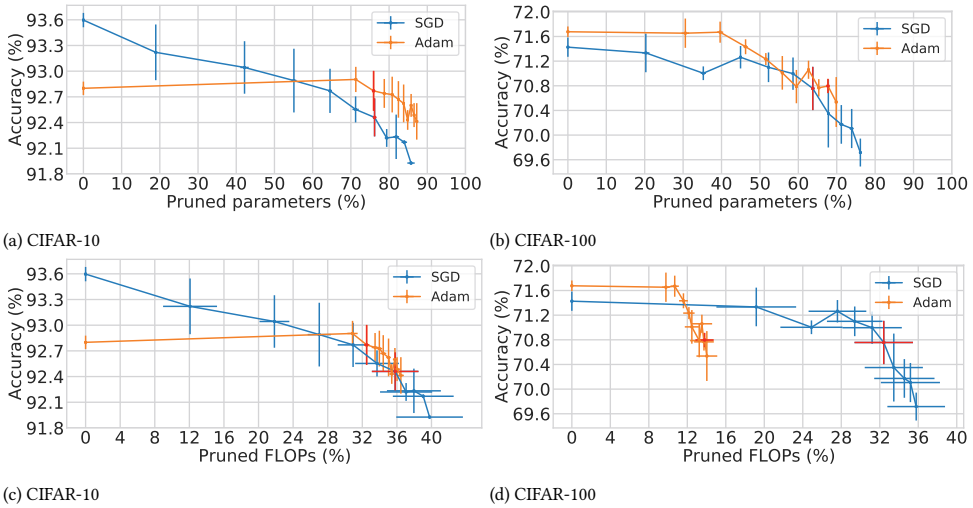


Figure 2.7: ResNet-56 on CIFAR-10 and CIFAR-100 by iteration averaged over three seeds, the error bars represent standard deviation for these three runs. The red dot corresponds to the selected best iteration.

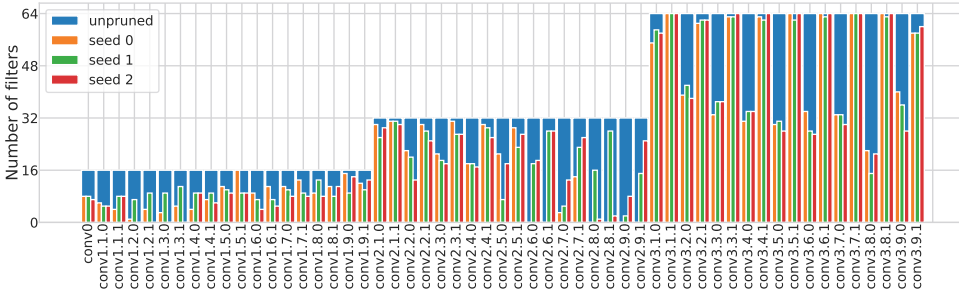


Figure 2.8: ResNet-56 architecture on CIFAR-10 by seed with SGD optimizer.

Table 2.5: Results for ResNets trained on CIFAR-100 with SGD and Adam optimizers. During retraining 80 epochs are used; the learning rate is decreased by 10 every 30 epochs. For NNrelief (ours)  $\alpha_{\text{conv}} = 0.95, \alpha_{\text{fc}} = 0.99$ .

ResNet	Optimizer	Acc (%) (baseline)	Acc drop (%)	Parameters pruned (%)	FLOPs pruned (%)
20	SGD	$68.75 \pm 0.15$	$1.26 \pm 0.33$	$54.54 \pm 0.24$	$12.49 \pm 1.52$
	Adam	$67.34 \pm 0.2$	$0.41 \pm 0.23$	$51.82 \pm 0.23$	$3.42 \pm 1.21$
56	SGD	$71.43 \pm 0.16$	$0.43 \pm 0.26$	$58.62 \pm 0.26$	$31.2 \pm 3.09$
	Adam	$71.68 \pm 0.09$	$0.88 \pm 0.12$	$67.72 \pm 0.29$	$13.79 \pm 0.94$

Figure 2.9). Adam, however, maintains a higher level of accuracy during retraining than SGD. We use 2000 samples to evaluate importance scores, and weight decay for (re-)training

equal to  $5 \cdot 10^{-4}$ .

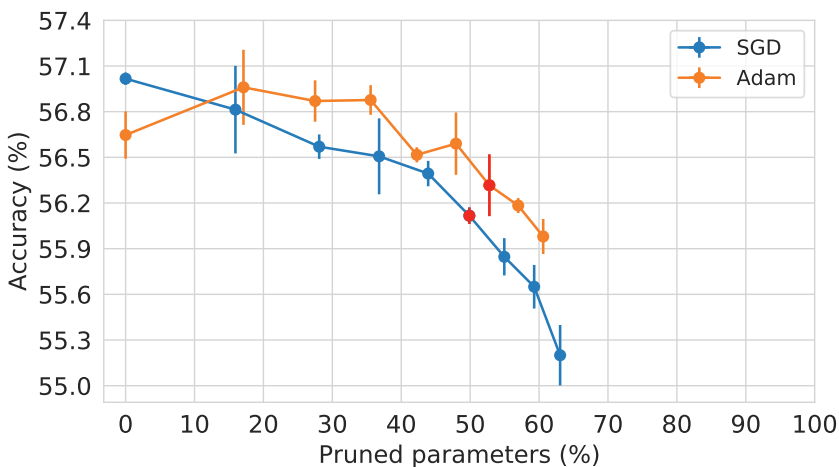


Figure 2.9: ResNet-18 on Tiny-ImageNet trained with SGD and Adam optimizers. The results are averaged over three random initializations. The bars represent standard deviations considering three runs.

**Sparsity** In contrast to VGG results, we do not observe a significant difference between optimizers in terms of produced sparsity for CIFAR-10/100 and Tiny-ImageNet. In the works that explore the question of sparsity [67], ResNet architecture was not explored. It seems that the skip connections allow the signal to pass unhindered to the next layers, making it harder for the Adam optimizer to find which activations are significant, and aggressively prune the less significant connections.

## 2.5 DISCUSSION

To illustrate the meaning of the importance scores we compare our approach with the magnitude-based one. Also, we present error bounds for the difference of a signal in the layer before and after pruning.

### 2.5.1 COMPARISON WITH MAGNITUDE-BASED APPROACH

In order to show the difference between our approach and the magnitude-based approach [29], we build the heatmaps with the location of top 15.5% of the most significant connections in the LeNet-300-100 pretrained on MNIST for both our proposed method and the magnitude-based pruning, since this number corresponds to the percentage of parameters that remain after pruning with our proposed rule (importance scores) using  $\alpha = 0.95$ . The importance scores (IS) are computed from Eq. 2.1, while for the magnitude-based rule, we consider both structured and unstructured pruning. In the structured case, we select the top 15.5% in each layer independently, and in the unstructured, the top 15.5% from all layers are selected, meaning that some layers may contain more or less than 15.5% of the parameters.

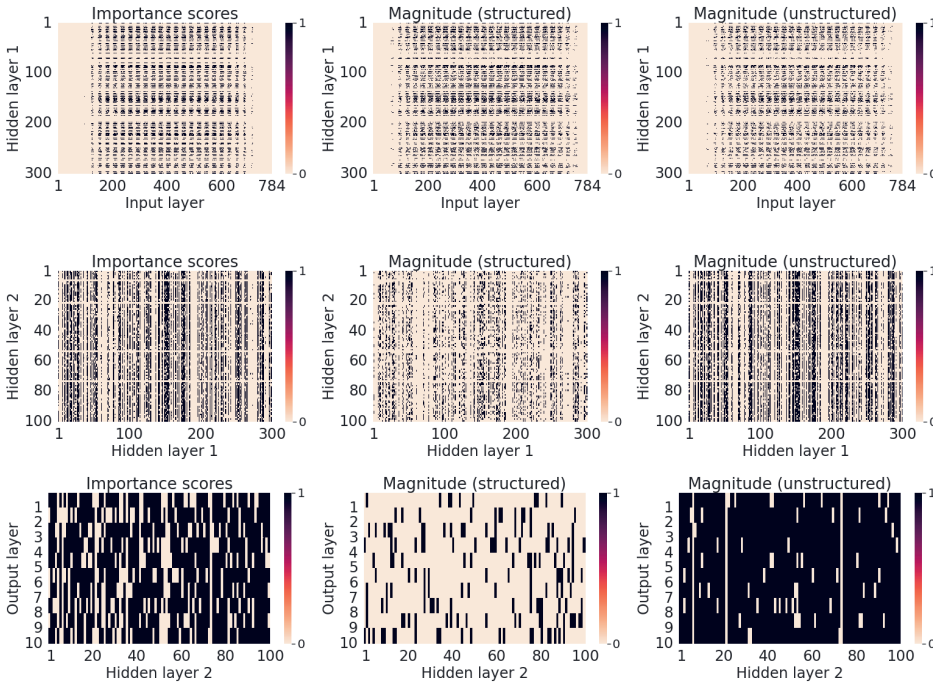


Figure 2.10: The remaining 15.5% connections according to Importance scores and magnitude-based rule (structured and unstructured pruning) for trained LeNet-300-100 on MNIST.

Figure 2.10 displays the difference between our approach and the magnitude-based (structured and unstructured) one when determining the significance of network connections. According to the figure, we can see different patterns for the remaining connection, especially for the unstructured magnitude-based rule, where more connections are retained in the last layer compared to the IS rule. Overall, we present the Jaccard index [69] between IS and both magnitude-based rules in Table 2.6. The Jaccard index (or Intersection over Union, IoU) for two sets  $A$  and  $B$  is defined as follows:  $\text{IoU} = \frac{|A \cap B|}{|A \cup B|}$ .

Table 2.6: Jaccard index (similarity) between IS rule ( $\alpha = 0.95$ ) and magnitude-based one (structured and unstructured) for each layer if 84.5% of parameters are pruned in LeNet-300-100 pretrained on MNIST.

	Layer	Magnitudes (structured)	Magnitudes (unstructured)
Importance scores	784 $\rightarrow$ 300	0.08	0.08
	300 $\rightarrow$ 100	0.11	0.18
	100 $\rightarrow$ 10	0.16	0.66

Both Figure 2.10 and Table 2.6 demonstrate that the IS rule prunes the network differently from both magnitude-based rules. First, the binary heatmap patterns in the figure indicate a clear difference between the methods. In addition, the IoU indicators are low for



every case, meaning that the IS pruning proposed herein determines the most significant connections differently to both magnitude-based rules. From empirical results, we observe that IS-based pruning deactivates more connections in overparameterized layers. In Table 2.7, we present the number of active neurons by layer after one pruning iteration (15.5% of all parameters are retained). As it can be seen, the IS metric deactivates more neurons in every layer compared to unstructured magnitude pruning [29, 35]. This happens because some neurons produce a low signal (or zero signal because of the ReLU activation function).

Table 2.7: Number of active neurons for magnitude-based pruning and with importance scores one (NNrelief) for LeNet-300-100 pretrained on MNIST.

Method	Architecture
Original	784 → 300 → 100 → 10
importance scores	380 → 131 → 96 → 10
magnitude-based (unstructured)	530 → 136 → 97 → 10

In contrast to magnitude-based pruning, NNrelief automatically determines the number of connections to prune when  $\alpha$  is given. However, for magnitude-based pruning, the number of pruned connections is a hyperparameter which is difficult to set. For example, if we prune 80% of the remaining parameters at every iteration as discussed in the Lottery Ticket Hypothesis work [35], then after 11 iterations magnitude-based pruning retains  $100\% \cdot 0.8^{11} \approx 8.6\%$  of the parameters, while NNrelief retains 1.51% (see Table 2.1 and Table A.3). This is achieved due to the usage of the input signal to estimate the importance scores, which helps to determine the number of connections to prune.

### 2.5.2 ERROR ESTIMATION

To complement the analysis of our approach we derive error bounds by considering the difference between a signal in the trained neuron and a reduced one without retraining, before and after the activation function. This analysis is important to make sure NNrelief does not destroy the pretrained structure, and that changes in the output neurons are not dramatic enough to prevent successful retraining. To illustrate this, we compare the derived error bounds with the observed changes in ResNet-20 pretrained on CIFAR-10 if the fully connected layer is pruned. Without loss of generality we may assume the first  $p$  connections in the neuron  $j$  of layer  $l$  are kept and that the bias  $b_j^{(l)}$  as well as weights  $w_{p+1,j}^{(l)}, \dots, w_{m_l,j}^{(l)}$  are pruned in a particular neuron  $j$ . Then, for any input  $x_n^{(l-1)} \in \mathbf{X}^{(l-1)} = \{x_1^{(l-1)}, \dots, x_N^{(l-1)}\} \subset \mathbb{R}^{m_{l-1}}$  we obtain:

$$\begin{aligned} \delta_p(x_n^{(l-1)}) &= \left| \sum_{k=1}^{m_{l-1}} w_{kj}^{(l)} x_{nk}^{(l-1)} + b_j^{(l)} - \sum_{k=1}^p w_{kj}^{(l)} x_{nk}^{(l-1)} \right| = \left| \sum_{k=p+1}^{m_{l-1}} w_{kj}^{(l)} x_{nk}^{(l-1)} + b_j^{(l)} \right| \leq \\ &\leq \left| b_j^{(l)} \right| + \sum_{k=p+1}^{m_{l-1}} \left| w_{kj}^{(l)} x_{nk}^{(l-1)} \right|. \quad (2.4) \end{aligned}$$

By averaging over all samples:

$$\begin{aligned} \overline{\delta}_p &\leq \left| b_j^{(l)} \right| + \sum_{k=p+1}^{m_{l-1}} \left| w_{kj}^{(l)} x_k^{(l-1)} \right| = S_j^{(l)} \cdot \left( \frac{\left| b_j^{(l)} \right|}{S_j^{(l)}} + \frac{\sum_{k=p+1}^{m_{l-1}} \left| w_{kj}^{(l)} x_k^{(l-1)} \right|}{S_j^{(l)}} \right) = \\ &= S_j^{(l)} \cdot \sum_{k=p+1}^{m_{l-1}} s_{kj}^{(l)} = S_j^{(l)} (1 - \alpha). \end{aligned} \quad (2.5)$$

Therefore, from (2.5) we see that the sequence of residuals  $\{\overline{\delta}_p\}_{p=1}^m$  decreases with respect to  $p$  and increases with  $\alpha$ . Also, for any Lipschitz continuous activation function  $\varphi \in \text{Lip}(X)$ ,  $X \subset \mathbb{R}$  with Lipschitz constant  $C$  we obtain:

$$\begin{aligned} \Delta_p(x_n^{(l-1)}) &= \left| \varphi \left( \sum_{k=1}^{m_{l-1}} w_{kj}^{(l)} x_{nk}^{(l-1)} + b_j^{(l)} \right) - \varphi \left( \sum_{k=1}^p w_{kj}^{(l)} x_{nk}^{(l-1)} \right) \right| \leq \\ &\leq C \left| \sum_{k=p+1}^{m_{l-1}} w_{kj}^{(l)} x_{nk}^{(l-1)} + b_j^{(l)} \right| = C \overline{\delta}_p(x_n^{(l-1)}). \end{aligned} \quad (2.6)$$

As a result, from Eq. (2.5) and Eq. (2.6):

$$\overline{\Delta}_p \leq C S_j^{(l)} (1 - \alpha). \quad (2.7)$$

For the point-wise estimation on set  $\mathbf{X}^{(l-1)} = \{x_1^{(l-1)}, \dots, x_N^{(l-1)}\} \subset \mathbb{R}^{m_{l-1}}$ ,  $\forall x_i^{(l-1)}$  we obtain:

$$\Delta_p(x_i^{(l-1)}) \leq \max_{x_n^{(l-1)} \in \mathbf{X}^{(l-1)}} C \overline{\delta}_p(x_n^{(l-1)}). \quad (2.8)$$

Since ReLU, ELU, sigmoid and, tanh are Lipschitz continuous functions then (2.7) and (2.8) hold for them, meaning that the sequence of residuals  $\{\overline{\Delta}_p\}_{p=1}^{m_l}$  decreases as well with the increase of  $\alpha$ . Moreover, in the case of ReLU and ELU, the Lipschitz constant is  $C = 1$ . In summary, by increasing  $\alpha$  we indeed reduce the approximation error of the neuron.

Analogously, for input  $m_{l-1}$ -channeled pruning set  $\mathbf{X}^{(l-1)} = \{x_1^{(l-1)}, \dots, x_N^{(l-1)}\}$ , where  $x_k^{(l-1)} = (x_{k1}^{(l-1)}, \dots, x_{km}^{(l-1)}) \in \mathbb{R}^{m_{l-1} \times h_{l-1}^1 \times h_{l-1}^2}$ , where  $h_{l-1}^1$  and  $h_{l-1}^2$  are height and width of input images (or feature maps) for convolutional layer  $l-1$ , and  $\mathbf{B}_j^{(l)} \in \mathbb{R}^{h_l^1 \times h_l^2}$  is a matrix that consists of the same bias value  $b_j^{(l)}$  at every element we obtain:

$$\begin{aligned} \delta_p(x_k^{(l-1)}) &= \left\| \sum_{i=1}^{m_{l-1}} \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} + \mathbf{B}_j^{(l)} - \sum_{i=1}^p \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} \right\|_F = \\ &= \left\| \sum_{i=p+1}^{m_{l-1}} \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} + \mathbf{B}_j^{(l)} \right\|_F \leq \sum_{i=p+1}^{m_{l-1}} \left\| \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} \right\|_F + \left\| \mathbf{B}_j^{(l)} \right\|_F \leq \\ &\leq \sum_{i=p+1}^{m_{l-1}} \left\| \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} \right\|_F + \sqrt{h_l^1 h_l^2} |b_j^{(l)}|. \end{aligned} \quad (2.9)$$

Then, by averaging over all pruning samples similarly to the fully connected case we obtain:

$$\bar{\delta}_p \leq \sum_{i=p+1}^{m_{l-1}} \frac{1}{N} \sum_{k=1}^N \left\| \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} \right\|_F = S_j^{(l)}(1-\alpha), \quad (2.10)$$

from which for any Lipschitz continuous activation functions  $\varphi$  with Lipschitz constant  $C$ , we obtain:

$$\begin{aligned} \bar{\Delta}_p &= \frac{1}{N} \sum_{k=1}^N \left\| \varphi \left( \sum_{i=1}^{m_{l-1}} \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} + \mathbf{B}_j^{(l)} \right) - \varphi \left( \sum_{i=1}^p \mathbf{K}_i^{(l)} * x_{ki}^{(l-1)} \right) \right\|_F \leq \\ &\leq CS_j^{(l)}(1-\alpha). \end{aligned} \quad (2.11)$$

Inequalities (2.7), (2.11) show how the signal in a neuron or filter, respectively, changes after the layer is pruned. Let us consider how this change influences the network output. We define  $N_{i_l}^{(l)}(x; \mathbf{W}^{(1:l)})$  as a value in the  $i_l^{\text{th}}$  neuron in layer  $l$ , where  $\mathbf{W}^{(1:l)} = (\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(l)})$  are parameters in layers  $1, \dots, l$ . The matrix that we obtain after pruning layer  $l$  we define as  $\hat{\mathbf{W}}^{(l)}$ , and  $m_l$  is the number of neurons in layer  $l$ ,  $1 \leq l \leq L$ . Then, for  $1 \leq l < L$  (for  $l = L$  see (2.5) in Section 5.2):

$$\begin{aligned} \varepsilon(x) &= \left| N_{i_L}^{(L)}(x; \mathbf{W}^{(1:L)}) - N_{i_L}^{(L)}(x; \mathbf{W}^{(1:l-1)}, \hat{\mathbf{W}}^{(l)}, \mathbf{W}^{(l+1:L)}) \right| = \\ &= \left| \sum_{i_{L-1}=1}^{m_{L-1}} w_{i_{L-1}i_L}^{(L)} \left( \varphi(N_{i_{L-1}}^{(L-1)}(x; \mathbf{W}^{(1:L-1)})) - \right. \right. \\ &\quad \left. \left. - \varphi(N_{i_{L-1}}^{(L-1)}(x; \mathbf{W}^{(1:l-1)}, \hat{\mathbf{W}}^{(l)}, \mathbf{W}^{(l+1:L-1)})) \right) \right| \leq \\ &\leq C \sum_{i_{L-1}=1}^{m_{L-1}} \left| w_{i_{L-1}i_L}^{(L)} \right| \left| N_{i_{L-1}}^{(L-1)}(x; \mathbf{W}^{(1:L-1)}) - \right. \\ &\quad \left. - N_{i_{L-1}}^{(L-1)}(x; \mathbf{W}^{(1:l-1)}, \hat{\mathbf{W}}^{(l)}, \mathbf{W}^{(l+1:L-1)}) \right| \end{aligned} \quad (2.12)$$

Performing similar transformations as in (2.12) until layer  $l$ , we obtain:

$$\begin{aligned} \varepsilon(x) &\leq C^{L-l} \sum_{i_{L-1}=1}^{m_{L-1}} \left| w_{i_{L-1}i_L}^{(L)} \right| \sum_{i_{L-2}=1}^{m_{L-2}} \left| w_{i_{L-2}i_{L-1}}^{(L-1)} \right| \dots \\ &\quad \sum_{i_l=1}^{m_l} \left| w_{i_l i_{l+1}}^{(l)} \right| \left| \varphi(N_{i_l}^{(l)}(x; \mathbf{W}^{(1:l)})) - \varphi(N_{i_l}^{(l)}(x; \mathbf{W}^{(1:l-1)}, \hat{\mathbf{W}}^{(l)})) \right| \end{aligned} \quad (2.13)$$

Then, averaging (2.13) over all pruning samples from  $\mathbf{X}$  and using (2.7):

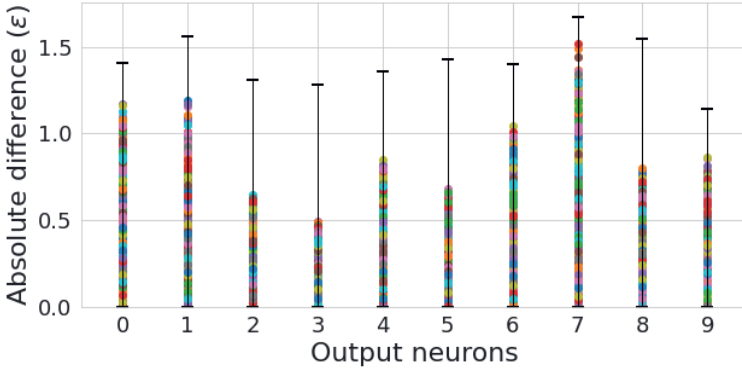


Figure 2.11: Absolute difference in the neurons' response at the output layer before and after pruning ResNet-20 ( $\alpha_{fc} = 0.95$ ). The bars represent the theoretical bounds, and the dots present empirical observations for 10000 images.

$$\begin{aligned} \bar{\varepsilon} &\leq C^{L-l} \sum_{i_{L-1}=1}^{m_{L-1}} \left| w_{i_{L-1}i_L}^{(L)} \right| \sum_{i_{L-2}=1}^{m_{L-2}} \left| w_{i_{L-2}i_{L-1}}^{(L-1)} \right| \cdots \sum_{i_l=1}^{m_l} \left| w_{i_l i_{l+1}}^{(l)} \right| \left| \bar{\Delta}_{p_{i_l}}^{(l)} \right| \leq \\ &(1-\alpha)C^{L-l} \sum_{i_{L-1}=1}^{m_{L-1}} \left| w_{i_{L-1}i_L}^{(L)} \right| \sum_{i_{L-2}=1}^{m_{L-2}} \left| w_{i_{L-2}i_{L-1}}^{(L-1)} \right| \cdots \sum_{i_l=1}^{m_{l+1}} \left| w_{i_l i_{l+1}}^{(l)} \right| S_{i_l}^{(l)}. \end{aligned} \quad (2.14)$$

Inequality (2.14) shows the bounds of the network error change after pruning on the pruning set  $X$ . Figure 2.11 shows the absolute changes in output responses for ResNet-20 after pruning on CIFAR-10 with  $\alpha_{fc} = 0.95$ . The error bars (in black) show the theoretical bounds of the difference estimated in (2.8) and (2.13), and the dots demonstrate the actual changes computed on the test data. From the figure, one can observe that the theoretical bounds capture the signal changes well in most cases, especially, for output neurons 0, 6 and 7.

## 2.6 CONCLUSION

In this chapter, we have aimed to create task-specific subnetwork within the main network for the given task and to answer Research Question 1 (RQ1). This has been achieved by proposing a novel pruning strategy and associated error estimation that applies to different neural network architectures. Our algorithm is based on eliminating layers' kernels or connections that do not contribute to the signal in the next layer. In this method, we estimate the average signal strength coming through each connection via proposed importance scores and try to keep the signal level in the neurons close to the original level. Thus, the number of removed parameters in a layer is selected adaptively, rather than being predetermined. As a result, the algorithm obtains a subnetwork capable of propagating most of the original signal while using fewer parameters than other reported

strategies. We explore the effect of Adam and SGD optimizers on our pruning strategy. The results show that for VGG architecture Adam optimizer gives much higher compression than SGD, however, for ResNet, we do not observe this behaviour. We observe that the obtained subnetwork tends to homogenize connection importance, hinting that every remaining connection is approaching similar importance for the dataset on average. The following step is to create multiple subnetworks, one for each task, in such a way that these subnetworks do not disturb each other.

## 3

## 3

## CONTINUAL LEARNING WITH SPECIALIZED SUBNETWORKS

*In the previous chapter, we described an algorithm to create a task-specific subnetwork for the given task via iterative pruning. This has been achieved by evaluating the average importance of every parameter while propagating the signal through the network. However, for multiple sequential tasks, we need multiple subnetworks created continually. Moreover, these subnetworks should not interfere with each other, causing forgetting.*

*In this chapter, we address the catastrophic forgetting challenge by considering a class-incremental learning scenario where a neural network sees test data without knowing the task from which this data originates. During training, the proposed solution finds a subnetwork within the DNN that is responsible for solving a given task with the algorithm proposed in Chapter 2. Then, in Section 3.3, we discuss how to sequentially create multiple subnetworks without conflicting with each other while still allowing knowledge transfer across them. This chapter provides the results on various image datasets (CIFAR-100, CUB-200-2011 and ImageNet-1000) for continual learning methods, significantly outperforming other state-of-the-art approaches.*

### 3.1 INTRODUCTION

Despite significant progress, deep learning methods tend to forget old tasks while learning new ones. This is known as *catastrophic forgetting* in neural networks [10, 11]. In the conventional setting, a machine learning model has access to the entire training data at any point in time. Instead, in continual learning or lifelong learning [70] data for a given task comes in sequentially at a specific learning moment, and then new data associated with another task comes in at a different moment. Continual learning aims at creating deep learning models that do not forget previously learned tasks while being able to learn new ones, i.e. addressing catastrophic forgetting. Often continual learning scenarios are divided into two categories [19]: task-incremental learning (task-IL) and class-incremental learning (class-IL). In the task-incremental case, the model knows which task is being solved at the testing stage, while in the class-incremental case, this is not known for the model. Therefore, the class-IL scenario is more challenging and more general than the task-IL one. In computer vision, there are works on continual learning for both scenarios in classification [5, 43, 71], while for object detection [72–74] and semantic segmentation problems [75–77] the class-IL case is more frequent. However, task-IL naturally arises in engineering applications, where task-ID is available at the inference stage.

There is evidence from neuroscience [21, 78, 79] that humans have special regions in the brain that are responsible for the recognition of specific patterns. Moreover, several studies show that the human brain encodes information in a sparse representation with an optimal fraction of active neurons of 1%-4% at the same time [20, 80]. Motivated by this observation, this thesis proposes a class-IL algorithm for image classification based on two steps: creating a subnetwork for a given task during training and selecting a previously obtained subnetwork during inference to make predictions. The first stage is achieved via iterative pruning that propagates input patterns through the network and eliminates the least useful connections. During inference, we first predict the current task when selecting the appropriate subnetwork from a small batch of test samples, and only then make a prediction with the selected subnetwork. We allow overlaps between subnetworks in order to induce knowledge transfer during training of new tasks. However, previously trained weights are not changed. Parameter update only occurs when training available neuron connections, which become part of a new subnetwork associated with the new task.

### 3.2 RELATED WORK

In this section, we provide a study on the current state of class-incremental continual learning methods, their categorization and the limitations of each category.

**Class-incremental learning** As previously mentioned, continual learning problems are usually classified according to whether or not the task-ID is available during inference. We focus on the class-IL scenario where the task ID is absent during inference since it is the most realistic and challenging scenario of continual learning. All class-IL methods are usually divided into three categories [16]: *regularization* [5, 81–85], *rehearsal* [86–89] and *architectural* [90–92]. The diagram with these types of methods and examples is shown in Figure 3.1.

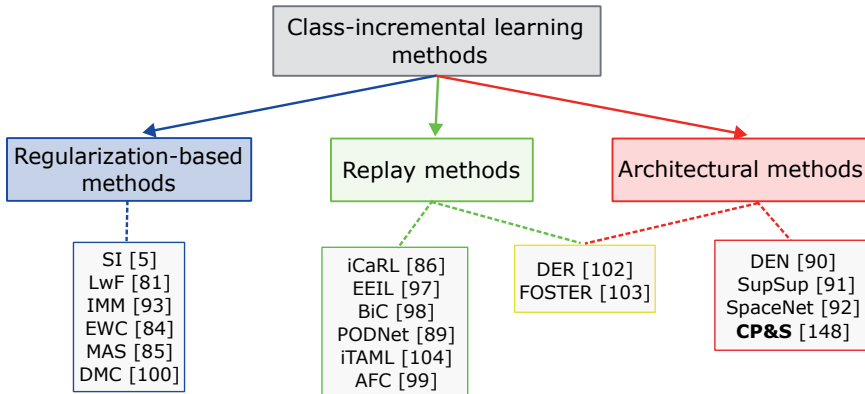


Figure 3.1: Categorization of continual learning approaches.

Purely *regularization-based* methods introduce an additional term in the loss function to prevent forgetting. Some approaches [5, 85, 93, 94] estimate the importance of connections for a given task and penalize the model for gradient updates during training for the next tasks. Learning without forgetting (LwF) [81] adds a term in the loss function that penalizes changes in old output heads for new data while training new output heads on new data. Regularization-based methods have the advantage of not storing past data in memory nor needing network expansion, but they perform worse compared to other approaches [19].

*Rehearsal (or replay)* methods replay small amounts of old classes [86] or generate synthetic examples [95] to be able to predict previously seen classes. iCaRL uses the nearest class mean [96] (NCM) classifier together with fixed memory of old data to mitigate forgetting. Bias-correction methods [87, 97, 98] aim to tackle the tendency of class-IL algorithms to be biased towards classes of the last tasks, which arises due to class imbalance at the latest stages [19]. PODNet [89] has multiple terms in the loss function using old data from the memory of a fixed size, penalizing signal deviations not only in the output layer but also in intermediate ones. AFC [99] uses knowledge distillation by estimating the importance of each feature map. The estimation is based on the increase of loss function from changing channels' parameters in the feature maps. The obvious limitation of rehearsal methods is the need to keep past data, which is often not desirable in practical applications due to privacy issues as mentioned by [100].

*Architectural* methods follow a different strategy where the network architecture is modified to avoid forgetting. For example, the Dynamically Expandable Network (DEN) [90] expands the network architecture in an online manner, increasing network capabilities, and introducing a regularization term to prevent forgetting. However, due to the expansion of the network, the final number of parameters is greater than for the original architecture, which increases the memory costs. Supermasks in Superposition (SupSup) [91] and SpaceNet [92] find subnetworks for every task. SpaceNet assigns parameters to one task only, without sharing knowledge between tasks which limits its allocation capabilities for long sequences of tasks. In addition, SpaceNet requires to pre-define the sparsity level for each task. SupSup uses a randomly weighted backbone [101] instead of pruning to obtain a task-related subnetwork. During inference, SupSup predicts the correct subnetwork for the



given test data, using all data points in the batch. In the provided experiments, the batch size is equal to 128 images which may not be applicable to real-life problems. DER [102] dynamically expands the feature extractor by introducing new channels and freezes old feature representation while learning a new task. Later, DER uses a small portion of old data and current data to finetune the network for all tasks. To stop the growth of the number of parameters, DER uses a pruning strategy, however, the final number of parameters is unpredictable. Similarly, FOSTER [103] introduces a new module with a feature map to learn new classes. However, it uses a knowledge distillation strategy inspired by gradient boosting instead of pruning to compress the model. As a result, the outcome of FOSTER is a single fixed-sized backbone network.

A Meta-Learning approach for class-IL is proposed by iTAML [104]. The algorithm for updating parameters for all old tasks also needs fixed-sized memory, but iTAML uses a momentum-based strategy for meta-updates to overcome catastrophic forgetting. At the test stage, iTAML starts by predicting the task associated with that sample using a given test batch, and then adapts its parameters to the predicted task using data from fixed memory. Finally, with the adapted model and predicted task-ID, iTAML makes a prediction. Overall, iTAML uses samples from previous tasks to prevent forgetting, and the batch of test data to predict task-ID, making it the most demanding algorithm out of consideration. Also, the model adaptation to the predicted task makes it computationally more expensive than other state-of-the-art methods.

**Iterative pruning for Continual learning** Typically, neural network pruning is used for model compression such that it reduces memory and computational costs. The pruning pipeline consists of three steps: network pretraining, deleting the least important connections or neurons based on some criterion, and network retraining. Iterative pruning is characterized by repeating the second and third steps several times. There are numerous approaches to pruning, namely magnitude pruning [29, 35, 49], data-driven pruning [22, 30, 50, 55] and sensitivity-based pruning [27, 28, 48, 105]. Iterative pruning has been recently applied in the context of task-IL but not in the more challenging class-IL scenario, where the task-ID is not known *a priori*. Unsurprisingly, pruning has been shown to lead to simplified neural networks with a small fraction of the original parameters. This can facilitate the accumulation of knowledge for new tasks, as demonstrated by task-IL methods based on iterative pruning, namely PackNet [43] that uses magnitude connections pruning [29], and CLNP [106] that uses data-driven neurons pruning [55]. Piggyback [44] learns the mask for every task, as well as CPG [107] which also expands a network in the ProgressiveNet manner [108]. The performance of these algorithms is strong for task-IL, but they have the significant limitation of requiring to know the task-ID.

We are interested in developing a class-IL method that contains the benefits of iterative pruning connections to obtain sparse network representations while being capable of selecting tasks without knowing the task-ID. To this effect, we developed a pruning strategy called NNrelief [22] that aims at leaving as many connections as possible available for future tasks, leading to sparser networks when compared to other pruning methods. The algorithm's idea is to propagate signal through the network, compute a metric called importance score for each connection which estimates its contribution to the signal of the following neuron, and then prune the least contributing connections incoming to the

Table 3.1: Assumptions used by different types of class-IL methods (“bs” means batch size).

Methods	Replay old data	test $bs > 1$	Adaptation
SI, MAS, LwF, LwM, SpaceNet	no	no	no
iCaRL, BiC, PODNet, DER, AFC, FOSTER	yes	no	no
iTAML	yes	yes	yes
<b>CP&amp;S (ours)</b>	no	yes	no

neuron.

**Task selection** Currently, there are few strategies for task selection in class-incremental learning. For example, iTAML [104] and SupSup [91] use similar ideas for task identification class-IL applied to image classification problems, and neither method uses pruning as a means to create space for new knowledge. The underlying assumption is that if a classifier network is well-trained, the highest output signal in the neuron of the output layer corresponds to the class belonging to the correct task. So, iTAML sums the largest output values of every task-related output in that layer over every test image in the batch, and then finds the layer with the highest total sum. SupSup relies on the entropy of the signal in each of the heads, in the hope that the model is confident in its prediction when it is in the correct head, meaning that the entropy of signal within the head should be smaller than in other heads. Note that both methods use batches of test samples to select the correct task: iTAML varies the batch size from 20 to 150 depending on the dataset, while SupSup uses 128 images in their experiments. A different strategy is pursued by Kim et al. [109], where an autoencoder is associated with a task during training. In the test stage, the reconstruction loss is computed for *every* autoencoder with the given test image, and the one with minimum reconstruction loss is chosen to make predictions. Subsequently, a classification model makes a prediction with the given predicted task-ID. It was shown that in the case of LwF [81] and LwM [82] this task-selection procedure improves classification accuracy. However, this task-selection approach requires training an autoencoder for every task which is impractical.

**Limitations of class-IL approaches** State-of-the-art class-IL methods have simplified training by replaying old data [86, 87, 89, 101], doing inference with a batch of images to determine the current task [91, 104] and by performing adaptation before inference [104]. Table 3.1 summarizes these assumptions for each method. In rehearsal methods, examples of previous classes are stored (with fixed or growing memory), which makes them inappropriate when images should not be kept for a long time. Similarly, the adaptation of a model for a given batch of test data before making a prediction (as in iTAML) is only possible when having examples in memory. Furthermore, the need for a significant number of images in a batch during inference arises from the difficulty of identifying the task-ID correctly with one image only. These can be strong model constraints when considering real-life applications.

### 3.3 PROPOSED METHOD

This section describes how to take advantage of neural network sparsification via pruning and use it to overcome forgetting old tasks while learning new ones. We discuss how to deal with the overlaps between subnetworks and how we activate proper subnetwork at the inference stage.

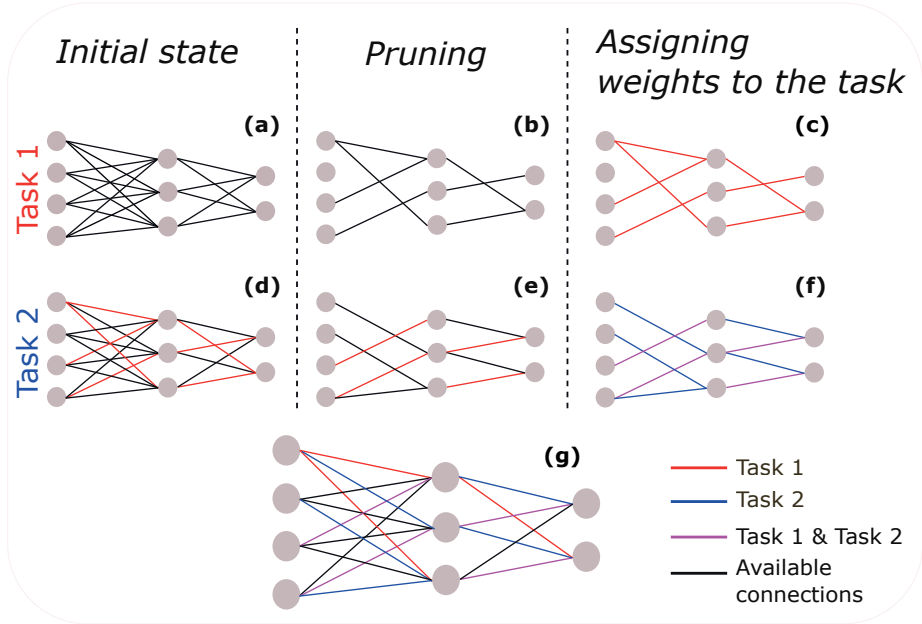


Figure 3.2: The overview of Continual Prune-and-Select (CP&S) training procedure. In stages (a)–(c), the first task is learned; in (d)–(f) the network learns task 2; the final outcome is in (g), where the network is trained for both tasks.

The proposed Continual Prune-and-Select (CP&S) method is based on training a subnetwork for each given task and then selecting the correct subnetwork when doing inference for new data with an unknown task-ID. We start by training a regular neural network for a specific task (Figure 3.2(a)), iteratively pruning it to find a subnetwork with good performance (Figure 3.2(b)). This creates a trained subnetwork capable of performing that particular task, leaving the remaining network free for future tasks (Figure 3.2(c)). Importantly, when a new task comes in (Figure 3.2(d)), the corresponding new subnetwork is found by iteratively pruning the entire original network – including all free neuronal connections *and* all existing subnetworks found for other tasks (Figure 3.2(e)). This is possible by freezing the parameters of previously found subnetworks (avoiding to forget past tasks associated with the corresponding subnetworks), updating all the remaining parameters of the network, and then pruning the entire network until the corresponding subnetwork is found. This way, the new subnetwork (Figure 3.2(f)) can contain connections from other subnetworks but it does not affect their performance on past tasks because it did not update the parameters of *shared connections* – it only updated the parameters

of *unshared connections*. This allows to have the transfer of knowledge from one task to another without forgetting (Figure 3.2(g)).

The new strategy proposed herein can be implemented with different pruning algorithms to create each subnetwork, and with different task selection algorithms to find the correct subnetwork for inference. As long as the pruning and selection strategies have reasonable performance, we expect this strategy to outperform previous continual learning methods in the class-IL scenario because (1) it avoids forgetting if the task is selected correctly (unlike iTAML); (2) it allows knowledge transfer among tasks (unlike SpaceNet). Also, CP&S (3) does not need to replay old data (unlike iCaRL, RPS-net, BiC, LUCIR, PODNet, AFC); (4) The backbone architecture is fixed and never changes during training (unlike DER), or requires additional temporary modules (unlike FOSTER).

Without loss of generality, we use NNrelief [22] pruning algorithm proposed in Section 2.3 because, as we can see from the results, it promotes sparser networks when compared to the state-of-the-art, and it creates a renormalization effect in the network that distributes the importance of neuronal connections. Concerning task selection (in our case subnetwork selection), we considered different strategies, including the one proposed in the literature that applies to our method (see iTAML [104] and SupSup [91]).

Formally, denoting our classification network as  $\mathcal{N}$  and considering  $T$  tasks, then:

$$\mathcal{N} = \cup_{t=1}^T \mathcal{N}^t, \quad (3.1)$$

where  $\mathcal{N}^t$  is the subnetwork for task  $t$ , with  $t = 1, 2, \dots, T$ . Each subnetwork  $\mathcal{N}^t$  is found with our NNrelief pruning algorithm that determines the most important parts of the main network for solving a given task  $t$ . This algorithm estimates each connection's contribution to the total signal of a receiving neuron when compared to the other connections that are incoming to that neuron. This contribution is computed by the *importance score* (IS) of every connection with Eq. 2.1 and Eq. 2.2 for the input signal  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  with  $N$  data points. Then NNrelief prunes the connections entering the neuron with the lowest contribution to the importance score whose sum is less than  $(1 - \alpha) \sum_{i=1}^m s_{ij}$ , where  $\alpha$  is the hyperparameter of the algorithm,  $0 \leq \alpha \leq 1$ . More details are given in our original article [22].

---

#### Algorithm 3 Pseudocode for CP&S training procedure

---

**Input:** network  $\mathcal{N}$ , datasets  $\{\mathbf{X}^t\}_{t=1}^T$ . Initialize learning parameters  $p$  (learning rate, weight decay, number of epochs, etc. ), pruning parameters (for NNrelief algorithm:  $\alpha$  and the number of pruning iterations  $k$ )

- 1: **for**  $t = 1, 2, \dots, T$  **do**
- 2:    $\mathcal{N}^t \leftarrow \text{Pruning}(\mathcal{N}, \mathbf{X}^t, \alpha, k)$
- 3:   freeze parameters  $w \in \mathcal{N}^t$  and never update them
- 4: **end for**

**Output:** network  $\mathcal{N}$  that learned tasks  $1, 2, \dots, T$ .

---

In the context of class-IL we receive datasets  $\mathbf{X}^1, \mathbf{X}^2, \dots, \mathbf{X}^T$  sequentially. The pruning algorithm then creates masks  $\mathbf{M}^1, \mathbf{M}^2, \dots, \mathbf{M}^T$  for every task  $t = 1, 2, \dots, T$ , where  $\mathbf{M}^t = (m_{ij}^t)_{i,j}$ ,

$$m_{ij}^t = \begin{cases} 1, & \text{if there is an active connection} \\ & \text{between neurons } i \text{ and } j, \\ 0, & \text{otherwise} \end{cases}$$

and corresponding importance scores  $S^1, S^2, \dots, S^T$ ,  $S^t = (s_{ij}^t)_{i,j}$ .

Once the subnetworks are created during training, selecting the correct subnetwork given a batch of test data becomes essential to do inference. In this article, we define a test batch of size  $s$  as  $\mathbf{X}^{test} = \{\mathbf{x}_1^{test}, \mathbf{x}_2^{test}, \dots, \mathbf{x}_s^{test}\}$ , and can simplify the notation to cases where the fully connected part of the network consists of one layer since we run all our experiments on ResNet architectures. However, there are no restrictions to apply this approach to any other type of architecture. We define the convolutional part for task  $t$  as  $\theta^t$ ,  $\theta^t : \mathbb{R}^{3 \times H \times W} \rightarrow \mathbb{R}^d$  ( $H, W$  are the height and width of an input image and  $d$  is the length of an output feature vector), and the fully connected layers as  $\varphi^t : \mathbb{R}^d \rightarrow \mathbb{R}^{num\_classes}$ .

Similarly to the selection of the pruning algorithm for subnetwork creation, we can also adopt different strategies to identify the correct subnetwork associated with a particular task. In order to establish a fair comparison with the literature, we focus on the *maximum output response* (maxoutput) strategy that is used by other methods (e.g. [91, 104]), but we also show that other task selection methods can lead to good results (see Appendix for a strategy based on Importance Scores).

The maxoutput strategy for task prediction is simply formulated as:

$$t^* = \arg \max_{t=1,2,\dots,T} \sum_{i=1}^s \max \varphi^t(\theta^t(\mathbf{x}_i^{test})). \quad (3.2)$$

This does not require data storage. There are no memory costs associated with this prediction.

---

**Algorithm 4** Pseudocode for CP&S inference procedure

---

**Input:** network  $\mathcal{N}$ , test batch  $\mathbf{X}^{test}$ .

1: predict task  $t^*$  for the test batch  $\mathbf{X}^{test}$  using Eq. (3.2)

2: make a prediction  $\hat{\mathbf{y}} = \mathcal{N}^{t^*}(\mathbf{X}^{test})$

**Output:** predicted classes  $\hat{\mathbf{y}}$  for test data  $\mathbf{X}^{test}$ .

---

**Complexity analysis** We estimate this separately for the training and inference stages. In the training stage for one task, we perform initial training and then the prune-retrain steps over  $k$  iterations. For retraining, we use a smaller number of epochs. Therefore, if we initially train the network with  $N$  epochs, then we define the number of retraining epochs as  $N_1$  with  $N_1 < N$ . Overall, we have  $N + kN_1 < (1+k)N$  epochs. In practice, we use  $k \leq 3$ , so the total number of training epochs for one task can be estimated as  $\mathcal{O}(N)$ . During inference, the maxoutput approach requires propagating input signal through every created subnetwork. That means if  $T$  tasks are learned, we need  $T$  inference operations.

### 3.4 NUMERICAL EXPERIMENTS

We compare CP&S with different methods available in the literature. Aiming to establish a fair comparison, we use different measurements of accuracy during the learning process,

namely *average multi-class accuracy* (ACC), *backward transfer metric* (BWT) [110] and *average incremental accuracy* (AIA) [86]. These metrics can be written assuming that a model learned  $T$  tasks and denoting  $R_{t_2, t_1}$  as the accuracy for task  $t_1$  after learning up to task  $t_2$  (inclusive, i.e.  $t_2 \geq t_1$ ):

$$\text{ACC}(T) = \frac{1}{T} \sum_{t=1}^T R_{T,t} \quad (3.3)$$

$$\text{BWT}(T) = \frac{1}{T-1} \sum_{t=1}^{T-1} R_{t,t} - R_{T,t} \quad (3.4)$$

$$\text{AIA}(T) = \frac{1}{T} \sum_{t=1}^T \text{ACC}(t) \quad (3.5)$$

The idea of the BWT is to measure the forgetting of the incremental-learning models, evaluating how much information about previous tasks is lost after learning a new one. We evaluate all methods using several class orderings to obtain robust results, as recommended in [19].

**Datasets** We evaluate CP&S on three datasets: ImageNet-1000, including its subset ImageNet-100 [111]; CUB-200-2011 [112]; and CIFAR-100 [63]. We also consider different task construction scenarios. For completeness, the datasets are briefly described as follows:

- ImageNet-1000 consists of 1,281,167 with  $224 \times 224$  pixels RGB images for training and 50,000 images for validation of 1000 classes. We split both ImageNet-100 and ImageNet-1000 in 10 incremental steps of equal size, similarly to the literature;
- CUB-200-2011 consists of 11,788  $224 \times 224$  pixels RGB images of 200 classes with 5,994 training and 5,794 for testing images;
- CIFAR-100 consists of 60,000 with  $32 \times 32$  pixels RGB images of 100 classes with 6k images per class. There are 50,000 training samples and 10,000 test samples;

We start our experiments with ImageNet-100 (the first 100 classes of the ImageNet-1000 dataset) and with CIFAR-100 before considering more challenging datasets such as ImageNet-1000 and CUB-200-2011. For all datasets, we use the ResNet-18 architecture, as considered by previous methods. For ImageNet-100/1000 datasets, we split them into 10 tasks of the same size (each task having 10 classes). We compare CP&S with other state-of-the-art models, namely iCaRL [86], EEIL [97], BiC [87], RPS-net [101], iTAML [104], DER [102] and FOSTER [103]. In addition, we provide a comparison with the case of Finetuning, when no anti-forgetting actions are performed, and a network sequentially learns new tasks one by one. For comparison with other works, we either reproduce the results from the official GitHub repository using the hyperparameters mentioned in the original articles or report the results from the original works when available. See the details in Appendix B.1.

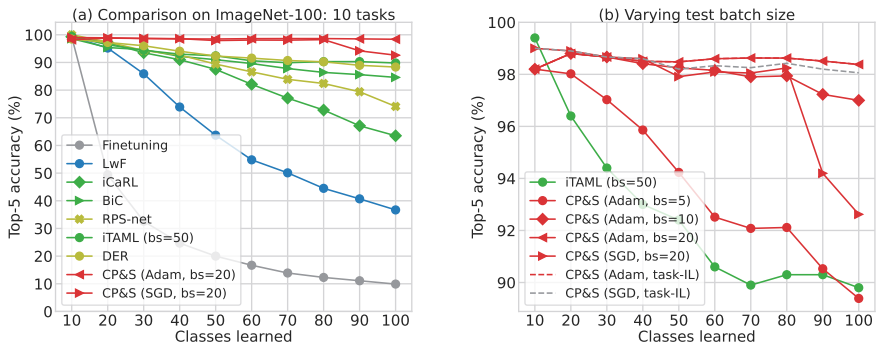


Figure 3.3: Results on ImageNet-100 and comparison with other approaches. Notation: "bs" refers to the test batch size; "task-IL" refers to the task-IL scenario where the task-ID is known, providing an upper bound to the results. The pruning parameter of CP&S is  $\alpha_{conv} = 0.9$  for both optimizers, SGD and Adam. The class ordering is generated by seed 1993 (iCaRL seed).

**ImageNet-100** Figure 3.3 shows that CP&S outperforms state-of-the-art methods for this dataset, even when considering two different optimizers – Stochastic Gradient Descent (SGD) and Adam [66]. We use a learning rate of 0.1 for SGD and 0.01 for Adam, dividing it by 10 on epochs 30 and 60, and we consider weight decay of  $10^{-4}$  for both optimizers. Figure 3.3(a) shows our predictions using a smaller batch size than iTAML – 20 samples instead of 50 – and compares them with other methods. Figure 3.3(b) clarifies the influence of considering different test batch sizes in CP&S method, where it is demonstrated that even when using 5 or 10 samples per batch we still perform better. The same figure also shows that when using Adam we identify the correct subnetwork in 100% of the cases because we reach the upper bound provided in the task-IL scenario, i.e. where the task-ID is known and subnetwork selection is not necessary. For SGD, we observe a slight drop in accuracy after task 8 compared to the task-IL scenario, although it still outperforms iTAML even though the latter uses 50 images for task identification and requires keeping images in memory. Overall, CP&S reaches 98.38% accuracy with Adam and 92.62% with SGD, translating into improvements for this dataset beyond 8% and 2.5% when compared to next best method, and even larger when compared to other methods after all classes are learned.

We note that using Adam [66] is advantageous for CP&S due to the higher level of sparsity that is produced after pruning with NNrelief when compared with other optimizers [22]. Note that pruning makes neuron connections available for creating new subnetworks associated with future tasks. If the number of available connections is small, then new subnetworks may not be sufficiently expressive to reach high accuracy for a given task. A similar effect is expected if the number of tasks is large, as shown in the next experiments for CIFAR-100.

**CIFAR-100** Before considering more challenging datasets such as ImageNet-1000 and CUB-200-2011, we focus on CIFAR-100 where we split its 100 classes by a different number of tasks: 5, 10 and 20 tasks composed of 20, 10 and 5 classes, respectively. For iTAML, we followed the original implementation with hyperparameters described in the paper

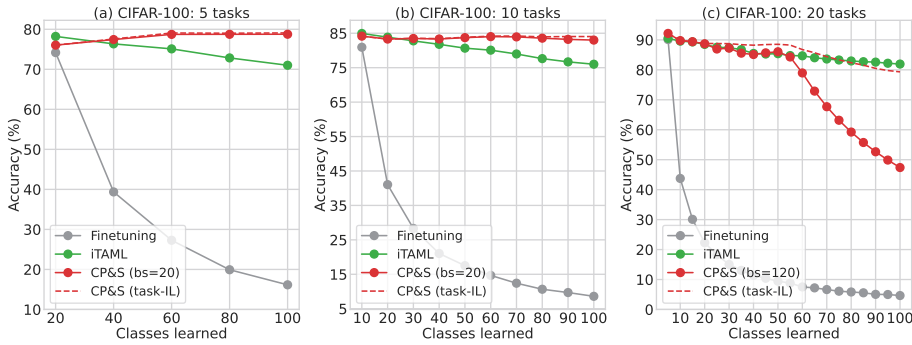


Figure 3.4: Comparison with iTAML on CIFAR-100 split in 5, 10 and 20 tasks. Notation: “bs” refers to the batch size during inference, “task-IL” refers to the task-IL scenario where the task ID is known, providing an upper bound to the results. Five different class orderings are used.

including the test batch size equal to 20, and using *ResNet-18(1/3)* [110] which is a modified version of the standard ResNet-18 architecture where the number of filters is divided by three. For CP&S, we use Adam for training using 70 epochs and starting with the learning rate 0.01 which is then divided by 5 every 20 epochs. We use  $\alpha_{conv} = 0.9$  and 3 pruning iterations as the pruning parameters of NNrelief.

Figure 3.4 shows that when considering 5 or 10 tasks CP&S significantly outperforms iTAML with the same batch size. However, for twenty tasks our performance drops sharply after the eleventh task, even in the ideal case where the task-ID is given (task-IL scenario). Despite being able to alleviate this drop by considering a larger test batch size, or by considering a different strategy for task (subnetwork) selection based on Importance Scores (see Appendix B.1), we observe this drop occurs approximately at the same number of tasks, independently of the class ordering used. Figure 3.5 (left) explains what occurs for the 20 tasks case by showing a heatmap with the task-selection accuracy by row for every task after a new task is learned. We also evaluate the prediction accuracy for each task when the task-ID is known (task-IL scenario) so that we can isolate the effect of not being able to appropriately select the subnetwork of interest for a given task and the effect of achieving low accuracy for a specific task. We observe that even in this case, the accuracy for each new task after the eleventh also drops (see Figure 3.5 (right)). Therefore, our method performs well until we reach a saturation point when there are not enough neuron connections available to create a sufficiently large subnetwork to achieve high prediction accuracy for a new task. This is a logical conclusion, as one can only learn new tasks while sufficient neuronal connections remain available for training. We call this issue *network saturation*. Notably, if there are enough available connections, this type of method reduces capacity saturation [113] by isolating task-specific parameters, unlike to regularization-based and replay methods [114]. However, the model cannot learn new concepts at network saturation point. Increasing the size of the original architecture eliminates this issue, but then we can unpredictably increase the memory costs, therefore, we work with a fixed architecture.

In addition, note that we do not keep data in memory (no replay), nor do we need to use adaptation to estimate the task before making a final prediction, unlike the methods



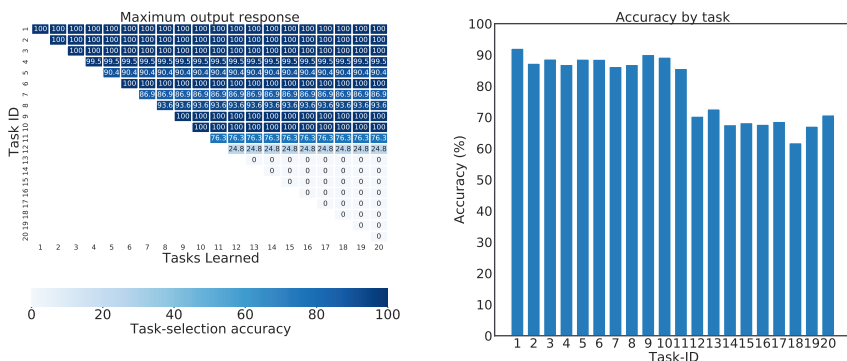


Figure 3.5: CIFAR-100 divided into 20 tasks of 5 classes each. Task-selection accuracy with maxoutput (**left**) and accuracy by task (**right**).

reviewed above. We also use smaller test batch sizes than iTAML, despite using the same task selection strategy. The following experiments show that this conclusion holds for more challenging datasets.

**ImageNet-1000** Focusing now on a more challenging dataset, we split ImageNet-1000 into 10 tasks of 100 classes. To evaluate CP&S, we train ResNet-18 with 90 epochs and SGD with a learning rate equal to 0.1 dividing it by 10 every 30 epochs. Figure 3.6(a) shows that CP&S performs better than the state-of-the-art, exhibiting more than 10% higher Top-5 accuracy than the next best method, which is DER [102] and more than 20% improvement over the second best BiC [87]. We found this result to be particularly striking, since the prediction accuracy remains around 94% with virtually no forgetting for the first time in the literature, to the best of our knowledge. Figure 3.6(b) also shows results for different test batch sizes for determining the task-ID and corresponding subnetwork. Once again, a batch size of 20 provides a good trade-off between accuracy and sample size. Interestingly, prediction accuracy is better for CP&S method than others even when using only 5 test samples in the batch. With 20 images in the test batch, we can almost reach the upper bound of the task-IL scenario, completely reaching it when using 50 images (i.e. identifying the task-ID correctly in 100% of the cases).

In addition, we provide a comparison on the ImageNet-1000 dataset calculating Top-1 accuracy. In Table 3.2, we observe that CP&S outperforms the two most recent state-of-the-art methods, DER and FOSTER, by more than 10%.

**CUB-200-2011** We split CUB-200-2011 dataset into four tasks with 50 classes in each of the tasks. For testing, we take standard ResNet-18 pretrained on ImageNet-1000 [111] and fine-tuned with SGD. For iTAML, we also use pretrained weights and use the same hyperparameters for fine-tuning that are used in the original paper for other large-scale datasets. The pruning parameter for CP&S is  $\alpha_{conv} = 0.95$  and only one pruning iteration is used.

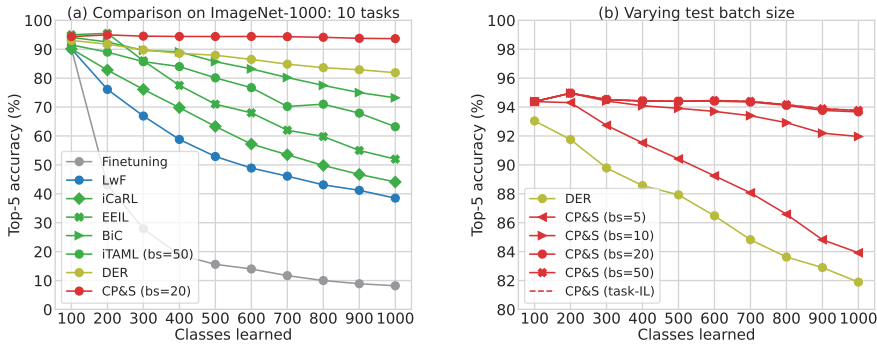


Figure 3.6: Results obtained for ImageNet-1000 dataset and comparison with other approaches (a) and different batch sizes (b). Notation: “bs” refers to the test batch size; “task-IL” refers to the task-IL scenario (upper bound obtained when task ID is known). The pruning parameter is  $\alpha_{conv} = 0.9$  for CP&S. Class ordering is generated by seed 1993 (referring to iCaRL’s seed).

Table 3.2: Average incremental accuracy on ImageNet-1000

Method	Top-1 AIA
iCaRL [86]	38.4
DER [102]	66.73
FOSTER [103]	68.3
<b>CP&amp;S (ours)</b>	<b>79.08</b>

Figure 3.7 presents the accuracy and BWT history with 20 test images per batch, once again using maxoutput as the task-selection strategy. For 20 test images, it can be observed that CP&S once again exhibits almost no forgetting of information about previous tasks while learning new ones. However, iTAML even though it keeps 2000 images in memory, continuously forgets previous tasks. In addition, note that 2000 images represent 1/3 of the CUB-200-2011 dataset, and that we see a dramatic loss of performance for iTAML when using 1000 images (which is still 1/6 of all the images). In the case of 5 images per test batch, we obtain similar forgetting as iTAML with 2000 images in memory but still a better forgetting metric than iTAML with 1000 images in memory. A more detailed comparison can be found in Appendix B.3.

In summary, CP&S outperformed the state-of-the-art for all datasets considered with the exception of CIFAR-100 when considering a large number of tasks. We demonstrate that we can perform better for small-scale and large-scale datasets (ImageNet-1000) where the second best methods are different. We considered scenarios where each task has a small or a large number of classes, including cases where there is a small number of training examples (CUB-200-2011) without keeping them in memory.

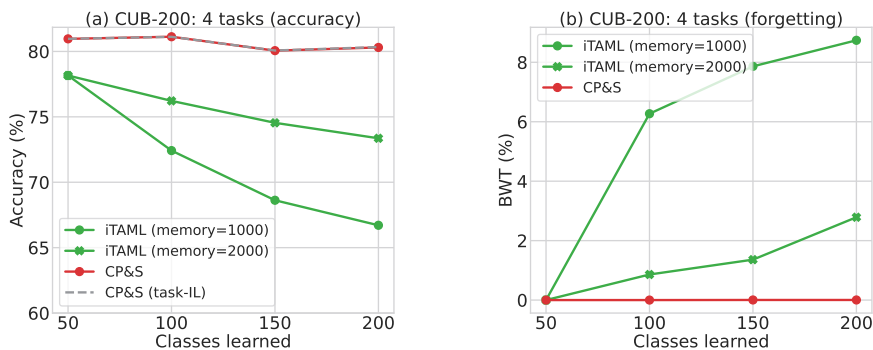


Figure 3.7: Comparison with iTAML on four tasks constructed from CUB-200-2011. Notation: “memory” is the number of images from previous tasks; “task-IL” refers to task-IL scenario as an upper bound for our approach.

### 3.5 FURTHER ANALYSIS

CP&S method’s performance degrades if there are too many tasks because the number of available neuron connections is not enough to create an expressive subnetwork and to select the correct task. This was shown for CIFAR-100 when considering 20 tasks (see Figure 3.4). In addition, there are scenarios where task selection during inference should be performed by a different strategy instead of maxoutput. For example, when there is an imbalanced number of classes within the tasks we note that using the modification of Importance Scores (IS) to select tasks is advantageous. Focusing on fully connected layers, for the given dataset  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_s\}$  we can compute:

$$\hat{s}_{ij}^t = \overline{w_{ij} \cdot m_{ij}^t \cdot \theta_i^t(\mathbf{X})} = \begin{cases} \overline{w_{ij} \cdot \theta_i^t(\mathbf{X})}, & \text{if there is an active connection} \\ & \text{between neurons } i \text{ and } j \\ 0, & \text{otherwise,} \end{cases} \quad (3.6)$$

where  $\overline{w_{ij} \cdot \theta_i^t(\mathbf{X})} = \frac{1}{s} \sum_{k=1}^s w_{ij} \cdot \theta_i^t(\mathbf{x}_k)$  and  $\theta_i^t$  are the feature extractor layers of task  $t$ .

Suppose we have importance scores  $S^1, S^2, \dots, S^T$  obtained from the training set, we can estimate the importance scores of these connections based on  $\mathbf{X}^{test}$  for every subnetwork  $t = 1, 2, \dots, T$ , and denote these estimations as  $\hat{S}^1, \hat{S}^2, \dots, \hat{S}^T$ .

Assuming that importance scores should be similar for train and test data for the true task-ID, we can formulate the decisive rule as:

$$t^* = \arg \min_{t=1,2,\dots,T} \sqrt{\sum_{i,j} (s_{ij}^t - \hat{s}_{ij}^t)^2}, \quad (3.7)$$

where  $s_{ij}^t$  and  $\hat{s}_{ij}^t$  are the elements of matrices  $S^t$  and  $\hat{S}^t$  respectively,  $t = 1, 2, \dots, T$ .

We consider the case where the first task consists of 50 classes, and 10 classes are in each of the following tasks, providing the comparison with iCaRL [86], LUCIR [87], PODNet [89] and AFC [99] (see Table 3.1 to recall different assumptions for each method). However, we also show that when using IS for task selection we require a larger batch size to improve task identification (60 test samples).

The maxoutput strategy does not work well in this case because most of the parameters are assigned to the first task (with 50 classes). As a result, this strategy predicts the first task when considering the last tasks for almost every batch, as shown in Appendix B.1.

As a final comment, we also investigated an alternative solution when we have a first task that is significantly larger than the following ones. This can be solved by pretraining convolutional weights with the first task and training only the task-specific parts in the network. We denoted this last strategy as “CP&S-frozen” since we pretrain *all* convolutional parameters with the first 50 classes, and, for the next tasks we train task-related batch normalization parameters and the fully connected part that is task-specific by construction. So, in this last strategy, each subnetwork consists of a common convolutional part (pretrained on the first task), batch normalization layers and an output classification head. We present an additional task-selection accuracy comparison between IS and maxoutput in Appendix B.1. We again observe poor performance for maximum output response strategy. The final results can be seen in Figure 3.8 and Table 3.3.

Table 3.3: Comparison between algorithms by average incremental accuracy and backward transfer metric at the end of all tasks. Mean values and standard deviation are computed using three different orderings.

Method	AIA (%)	BWT (%)
iCaRL [86]	61.63 $\pm$ 0.25	12.77 $\pm$ 0.30
LUCIR [87]	63.29 $\pm$ 0.36	10.09 $\pm$ 0.12
PODNet-CNN [89]	64.56 $\pm$ 0.28	11.90 $\pm$ 0.04
PODNet-NME [89]	65.07 $\pm$ 0.44	<b>1.18</b> $\pm$ 0.16
AFC [99]	65.73 $\pm$ 0.09	7.46 $\pm$ 0.38
CP&S (bs=60, IS)	64.97 $\pm$ 4.23	11.68 $\pm$ 0.20
CP&S-frozen (bs=60, IS)	<b>70.55</b> $\pm$ 5.05	4.90 $\pm$ 2.35

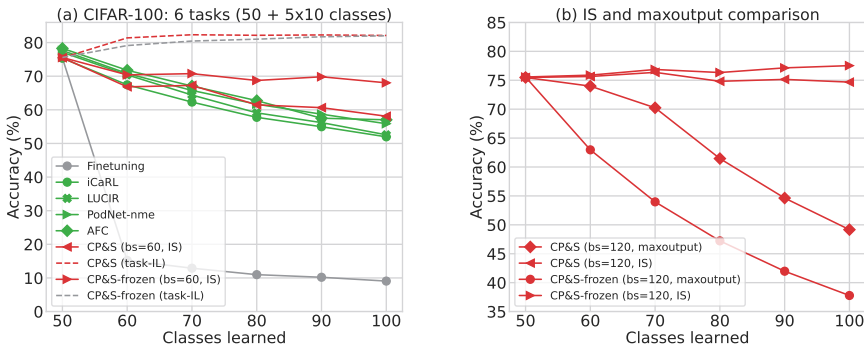


Figure 3.8: Accuracy history for ResNet-32 trained with CP&S and state-of-the-art. The pruning parameter is  $\alpha_{conv} = 0.9$  for CP&S strategy, and “task-IL” refers to the same upper bound mentioned in the previous figures. Three different class orderings are used.

We believe this knowledge transfer strategy might be interesting to explore in the future, where the first heads of the network specialize in selecting tasks and the deeper layers specialize in class prediction for each task.

**Knowledge transfer** Let us explore how many parameters are used by every task in the case of ResNet-18 on ImageNet-1000. We consider the union and the intersection of all masks as sets. In Figure 3.9(a) we show how the union and intersection are distributed across parameters after the last task is learned in the case of ResNet-18 on ImageNet-1000. From the union, we observe that the last layers are almost fully occupied in contrast to the first layers. From the intersection, it can be seen that a significant fraction of parameters is shared between each of the tasks across all layers. At the same time, about 85% are assigned to two and more tasks (Figure 3.9(b)). Notably, 35% of parameters are shared across all ten tasks and about 50% of parameters are used for nine or more tasks. From these figures, we can conclude that almost all parameters are occupied at the end, having significant overlaps between subnetworks. However, looking at Figure 3.6, we see that performance remains stable, without drops. This allows us to conclude that subnetworks share knowledge between tasks, which helps to assimilate new patterns.

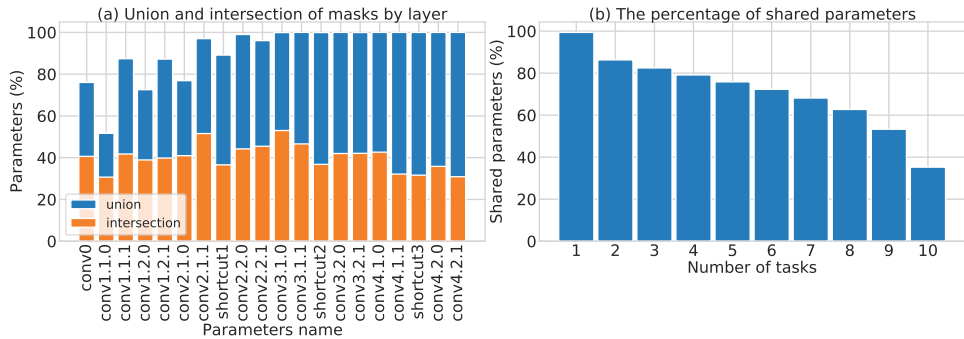


Figure 3.9: Visualization of employed masks and shared parameters for ResNet-18 on ImageNet-1000.

### 3.6 CONCLUSION

In this chapter, we have proposed a continual learning algorithm (CP&S) to overcome catastrophic forgetting issue. This algorithm creates a task-specific subnetwork for every task via an iterative pruning approach described in Chapter 2, which is developed to provide sparser network representation compared to other pruning approaches. During training of a task, weights are pruned and then fixed, such that future tasks cannot destroy the weights in this subnetwork, while still being able to use them for other subnetworks. During the evaluation of new data, the correct task-ID and associated subnetwork have to be inferred from a small batch of samples.

Notwithstanding, the Chapter demonstrates that combining subnetwork creation and subnetwork selection methods into one paradigm provides a general approach to solving class-IL problems. We believe the proposed strategy can be further improved by developing better task-prediction strategies that do not need a batch of test data. CP&S outperforms all state-of-the-art methods on a variety of datasets. For ImageNet-1000, we show an improvement of more than 10% accuracy when compared to previous algorithms. Even though we apply CP&S to image classification tasks, no additional limitations are foreseen when applying it to other machine learning problems. In Chapter 4, we consider a special

case of semantic segmentation — surface defect segmentation problem — where we have improved the subnetwork selection strategy by training a separate model for this purpose.



# 4

## SUBNETWORK SELECTION IN SURFACE DEFECT SEGMENTATION PROBLEM

4

Chapter 3 has described a general algorithm for class-incremental learning classification problem. The proposed strategy includes the subnetwork creation stage for every task during training and the subnetwork selection stage during inference to predict the correct subnetwork. As we have discussed in the previous chapter, the considered subnetwork selection approaches have a significant limitation of requiring several test images to predict the subnetwork properly.

In this Chapter, we improve the subnetwork prediction stage for the case when every task consists of input data of the same class. Therefore, we consider a continual surface defect segmentation problem where the model receives a new type of defect at each incremental step. We train the linear discriminant analysis (LDA) classifier in a continual learning manner to infer the defect-related subnetwork for the prediction. As in the previous chapter, we demonstrate that the proposed paradigm outperforms other learning strategies by a margin and is even comparable with the scenario when all data is available at the same time.



## 4.1 INTRODUCTION

The problem that we consider in this Chapter belongs to a general class of automatic defects inspection tasks. In contrast to classification, here we aim to predict the location of the defect region pixel-wise. In the following sections, we describe the problem in general and explain the difference from the classification case

Automatic defects inspection plays an important role in product quality evaluation [115]. In the beginning of the field, the creation of meaningful features to find defective regions was done manually [116–119]. Although classical machine learning methods have been proposed to identify images with defective surfaces [120–122], recent advances in deep learning research have led to an increase in performance [115]. Typically, there are three types of tasks for defect inspection with neural networks – classification, detection [123] and segmentation [124]. In the case of defect classification, transfer learning helps to increase the network’s ability to detect defective surfaces [125, 126]. For segmentation, most methods are based on the U-Net architecture [127] taking advantage of convolutional layers that automatically extract features from the images of the surfaces [128–131]. Attention mechanisms [132] employed in the model’s architecture can lead to even more accurate predictions [133, 134].

The advent of deep learning models came with more data for training and comparing these models in different real-life scenarios. For instance, after [118] proposed their NEU-DET dataset with Hot Rolled Steel Strip Surface defects, containing six types of defects, other groups collected datasets with either different defect categories or a more significant number of defects, e.g., GC10-DET [135] and X-SDD [136]. In segmentation literature, we can also find examples of different categorizations of surface defects, e.g., the Magnetic tile dataset [131] contains images of five types of defects together with defect-free cases. As a final example, the dataset collected by [137] also contains a large number of images but only has three types of defects.

Notwithstanding the increase in availability of datasets, there are many instances where there are few types of defects in each dataset. This is a natural occurrence in Engineering practice because many processes are not amenable to high-throughput. Simultaneously, if new defects occur or if another defect identification task with similar characteristics is encountered, using the original dataset and neural network model while considering new types of defects in similar (or even different) materials can be invaluable. However, training the same neural network model on a new dataset currently requires retraining it on all the data, even if the model was already capable of detecting some types of defects. This happens because deep learning models suffer from catastrophic forgetting [10, 11, 138]. In conventional training, neural networks cannot learn new tasks without forgetting old ones if the tasks are learned incrementally. Instead, the continual learning field [114] aims to solve this type of problem where the model receives data in batches (tasks) but aims to learn information mitigating the forgetting issues.

We illustrate the impact of catastrophic forgetting on segmentation tasks in Figure 4.1 by considering the defect segmentation dataset SD-saliency-900 [139]. This dataset consists of images with three types of defects: scratches, patches and an inclusion. We illustrate this phenomenon by focusing on three typical learning scenarios: 1) *single-task* training where each defect is learned with a single network, meaning there are three networks in total; 2) *joint* training where the model has access to the entire dataset at once; 3) *finetuning*, in

which the network learns to segment sequentially, adapting the parameters for the new task, having them pretrained on previous ones.

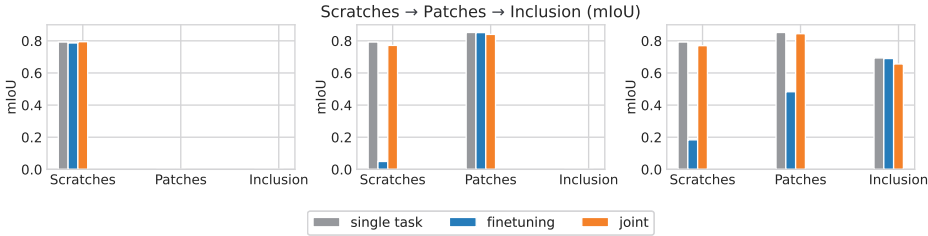


Figure 4.1: Example of forgetting in the case of incremental learning of three types of defects.

For all three learning scenarios, we quantify the segmentation performance via the mean Intersection over Union (mIoU) score for every task after each incremental step (Figure 4.1). We observe that finetuning on a new task leads to a significant drop in performance for the previous task(s), as indicated by the blue bars – a clear illustration that learning a sequence of tasks with a single network leads to forgetting the previous tasks in the sequence (catastrophic forgetting). However, forgetting does not occur in the case of single- and joint-task training because the network is capable of learning each of the defects separately without any pretraining, while also being capable of learning all of them together. We also note that both single- and joint-task training have comparable performance, despite a small decrease in the latter case<sup>1</sup>.

Therefore, we see that the ability to predict defects of previous types is lost when training for a new type of defect, i.e. that is out-of-distribution. The main objective of our work is to propose a continual learning algorithm suitable for the surface segmentation problem. To the best of our knowledge, this is the first work that develops a continual learning approach for surface defect segmentation.

## 4.2 RELATED WORK

In particular, we focus in this Section on regularization-based methods that penalize parameters obtained on incremental step  $t - 1$  from drastic changes while learning the task on incremental step  $t$ . For example, SI [5], EWC [84] and MAS [85] employ total loss  $\mathcal{L}^{(t)}(x; \theta^{(t)})$  on incremental step  $t$  that consists of the loss computed for the current data, and a penalty term to prevent forgetting:

$$\mathcal{L}^{(t)}(x; \theta^{(t)}) = \mathcal{L}_{curr}(x; \theta^{(t)}) + \frac{\lambda}{2} \sum_{i=1}^{\#params} \Omega_i (\theta_i^{(t)} - \theta_i^{(t-1)})^2, \quad (4.1)$$

where  $\mathcal{L}_{curr}(x; \theta^{(t)})$  is a loss on the current data,  $\sum_{i=1}^{\#params} \Omega_i (\theta_i^{(t)} - \theta_i^{(t-1)})^2$  is the penalty term,  $\Omega_i$  is the cumulative importance for parameter  $i$ , and  $\theta^{(t-1)}, \theta^{(t)}$  are network parameters

<sup>1</sup>As a short note, marginal improvements in performance sometimes occur when changing the task order (investigated at the end of the article). For example, the mIoU performance for the Scratches task improved by 0.13 points after learning the Inclusion task, but the improvement is small compared to how much it degrades after learning the Patches task.

at incremental steps  $t - 1$  and  $t$ , respectively. Learning without forgetting (LwF) [81] aims to mitigate forgetting by minimizing the cross-entropy between output probabilities before and after the model is trained on a new task.

Literature on continual learning for semantic segmentation is scarcer than for the classification problem. We can find examples that adapt classification continual learning algorithms to segmentation [9, 140], or some new approaches designed specifically for segmentation [75, 76, 141]. Similar to the classification case, better results are achieved by the methods that use a fixed-size memory buffer with samples from old tasks to overcome forgetting [142, 143]. However, even though these methods use old data (facilitating training), they still show significant forgetting of the first tasks while performing well only on the last ones.

Continual learning also finds its application in industrial and manufacturing cases. For example, MAS [85] was applied for product quality evaluation [144]. The approach clones the output head for previous tasks with the lowest loss on the current data and uses this copy as initialization for a new task. The weight transfer for the output layer, and MAS algorithm that penalizes parameters from previous layers, show good performance for the considered regression problem. Regularization-based methods have been examined for anomaly detection in manufacturing process [145] and fault prediction in lithium-ion batteries [146]. [147] developed an adaptive classification framework based on continual learning to identify new unlabeled samples. The proposed approach uses Mahalanobis distance and is employed to decide whether a new batch of data belongs to the already seen defect type, or forms a new one.

4

### 4.3 PROPOSED METHOD

We propose to take advantage of architectural methods that create task-specific subnetworks for each task, eliminating the subnetwork selection issue. As a base method, we consider Continual Prune-and-Select (CP&S) [148] proposed in Chapter 3 where we improve the subnetwork selection process by training a model for this purpose, instead of having simple metric-based decision rules. In general, the task-prediction problem is quite challenging in continual learning [109] and can be seen as an out-of-distribution (OOD) detection problem [149]. The difficulty arises from the presence of arbitrary classes in each task, leading to cases where classes within each task may not be similar, while classes from different tasks may have important similarities. This poses a challenge to identify the task-ID and corresponding subnetwork, affecting the performance of the continual learning model when the wrong subnetwork is selected. Conversely, these methods have the advantage that when the correct subnetwork is identified then there is no forgetting, which explains their state-of-the-art performance in different image-classification datasets [148].

However, in contrast to image classification, every task in defect segmentation problems consists of defects of only one type. This represents an opportunity for architectural continual learning methods because we can train a model that learns the distribution of each defect separately. To do so, we use linear discriminant analysis (LDA) model trained on features extracted from a pretrained convolutional neural network [150, 151]. The choice of LDA is explained by its efficiency and simplicity relative to deep learning-based methods (e.g., [152]), where for each class it is necessary to train a separate model with

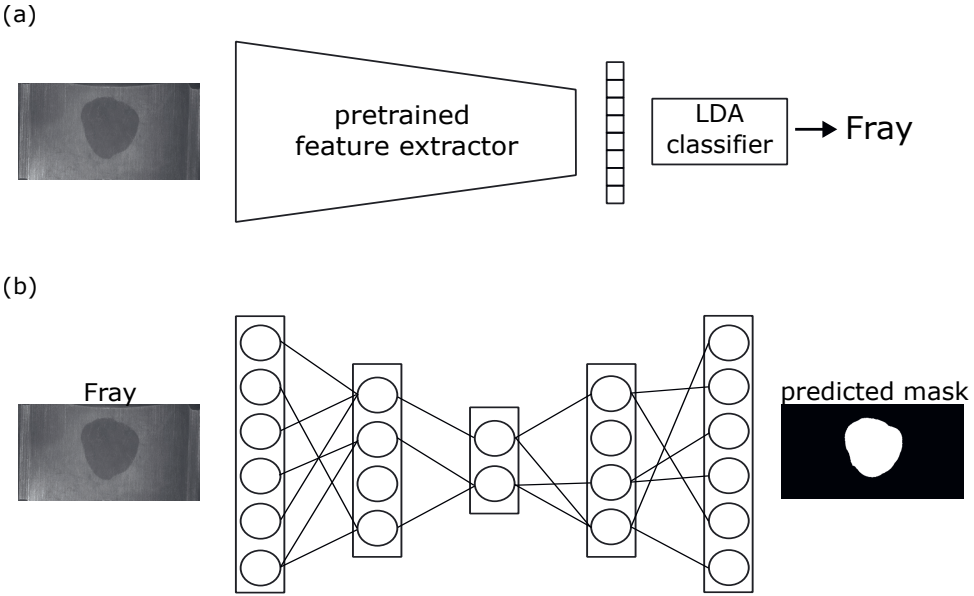


Figure 4.2: An overview of the proposed method: (a) task ID (i.e., defect type/subnetwork) prediction; (b) defects segmentation.

several hundred thousand parameters. For the segmentation model, we use the U-Net architecture [127], in which we create task-specific subnetworks via iterative pruning. As a pretrained feature extractor, we use the EfficientNet-B5 architecture [153] pretrained on ImageNet-1000 [111].

In Figure 4.2, we illustrate the inference stage of our approach, which consists of two steps: (1) predicting the defect type (task ID) with LDA; and (2) using a subnetwork that corresponds to the predicted defect to predict the segmentation mask. Note that at the inference stage, defect type prediction and defect mask prediction need to be done sequentially. Training for these steps can be done in parallel and independently from each other. We call our proposed approach LDA-CP&S since it uses the CP&S paradigm of creating subnetworks during training, and it employs LDA for the subnetwork selection.

Referring again to Figure 4.1, we recall that the three separate models (single-task grey bars) are capable of learning the defects slightly better than joint training with all the tasks together (orange bars). This hints that having task-specific parameters associated with only one task can even help the learning process. At the same time, the shared parameters provide a transfer learning effect between a new subnetwork and all the ones created before. Both advantages can be exploited by LDA-CP&S.

Notwithstanding, our method could suffer a performance drop from two possible sources: the pruning stage, and the LDA classification stage. The performance reduction due to pruning may occur because some important parameters could be deleted when creating additional space (free connections) for future tasks. In addition, misclassification by LDA could result in signal routing through the wrong subnetwork and consequently poor segmentation performance. In Section 4.4, we show that these two sources of error

are negligible compared to the benefits of our approach. In the following subsections, we describe the processes for subnetwork creation and LDA training.

### 4.3.1 SUBNETWORK CREATION

To create a subnetwork for the given task, we use NNrelief pruning algorithm [22] (Chapter 2). The approach evaluates the strength of the signal that propagates through every connection/kernel. This pruning technique shows better sparsity results than other connection/kernel-based pruning techniques [29, 49, 54].

For the set of  $m_{l-1}$ -channelled input samples  $\mathbf{X}^{l-1} = \{\mathbf{x}_1^{l-1}, \dots, \mathbf{x}_N^{l-1}\}$ , where  $\mathbf{x}_k^{(l-1)} = (x_{k1}^{l-1}, \dots, x_{km_{l-1}}^{l-1}) \in \mathbb{R}^{m_{l-1} \times h_{l-1}^1 \times h_{l-1}^2}$  with  $h_{l-1}^1$  and  $h_{l-1}^2$  being the height and width of feature maps for convolutional layer  $l$ . For every kernel  $\mathbf{K}_{1j}^l, \mathbf{K}_{2j}^l, \dots, \mathbf{K}_{m_{lj}}^l$ ,  $\mathbf{K}_{ij}^l = (k_{ijqt}^l) \in \mathbb{R}^{\eta \times \eta}$ ,  $q \geq 1$ ,  $r_l \geq t$ , where  $r_l$  is a kernel size, and for every bias  $b_j^{(l)}$  in filter  $\mathbf{F}_j^l$ , we define  $\hat{\mathbf{K}}_{ij}^l = \left( |k_{ijqt}^l| \right)$  as a matrix consisting of the absolute values of the matrix  $\mathbf{K}_{ij}^l$ . Then we compute importance scores  $s_{ij}^l, i \in \{1, 2, \dots, m_{lj}\}$  of kernels  $\mathbf{K}_{ij}^l$  as follows:

$$s_{ij}^l = \frac{\frac{1}{N} \sum_{n=1}^N \left\| \hat{\mathbf{K}}_{ij}^l * \left| x_{ni}^{l-1} \right| \right\|_F}{S_j^l},$$

where  $S_j^l = \sum_{i=1}^{m_{lj}} \left( \frac{1}{N} \sum_{n=1}^N \left\| \hat{\mathbf{K}}_{ij}^l * \left| x_{ni}^{l-1} \right| \right\|_F \right)$  is the total importance score in filter  $\mathbf{F}_j^l$  of layer  $l$ , with  $*$  indicating a convolution operation, and where  $\|\cdot\|_F$  is the Frobenius norm.

The sketch of the algorithm for pruning filter  $\mathbf{F}_j^l$  in a convolutional layer  $l$  can be described as follows (see Chapter 2, Section 2.3 for more details):

1. Choose  $\alpha \in (0, 1)$  – the amount of kernels' importance that we want to keep relative to the total importance of the kernels in the filter  $\mathbf{F}_j^l$ .
2. Compute importance scores  $s_{ij}^l$  for all kernels in the filter  $\mathbf{F}_j^l$ ,  $i = 1, \dots, m_{lj}$ , using Eq. 4.2.
3. Sort importance scores  $s_{ij}^l$  for the filter  $\mathbf{F}_j^l$ .
4. For the sorted importance scores  $s_{ij}^l$  find minimal  $p \leq m_{lj}$  such that  $\sum_{i=1}^p \hat{s}_{ij}^l \geq \alpha$ .
5. Prune kernels with the importance score  $s_{ij}^l < \hat{s}_{ij}^l$  for all  $i \leq m_{lj}$  and fixed  $j$ .

Overall, NNrelief finds kernels that propagate on average the lowest signal according to the Frobenius norm and prune these kernels. As the outcome of the procedure, we obtain a subnetwork (sub-U-Net) that predicts the defect for only one type of defects. Then we fix all parameters that are assigned to this subnetwork and do not update them anymore. When the network receives a new task with a new type of defect, CP&S finds a subnetwork for this task within the main U-Net, using the parameters assigned to the previous tasks, but without updating them. Algorithm 3 illustrates the pseudocode for CP&S and Figure 4.3 illustrates the method.

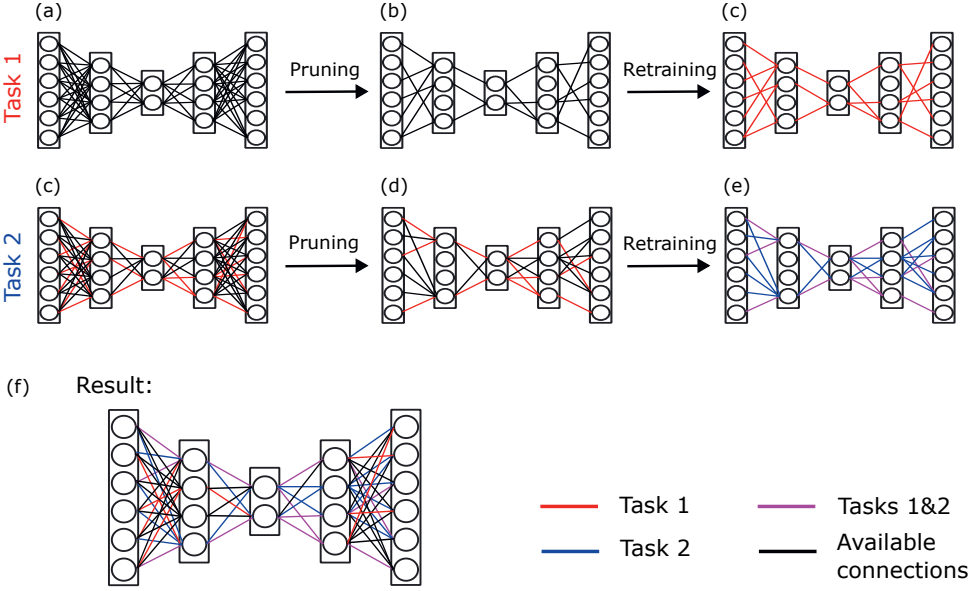


Figure 4.3: An overview of Continual Prune-and-Select (CP&S): an example with two tasks on U-Net based architecture.

### 4.3.2 SUBNETWORK SELECTION

To predict the task ID (type of defect) at the inference stage, we propose to use linear discriminant analysis (LDA). In this subsection, we describe the training procedure for LDA. In LDA, it is assumed that all classes have class means  $\mu^{(1)}, \mu^{(2)}, \dots, \mu^{(T)}$  and share the same covariance matrix  $\Sigma$ . However, in continual learning, we do not have access to all tasks at the same time, but only task  $t$ . Therefore, the covariance matrix needs to be updated online with respect to the new data batch.

Let us denote a new given task as  $\mathbf{X}^{(t)} = \{x_1^{(t)}, x_2^{(t)}, \dots, x_{N_t}^{(t)}\}$ . Following streaming LDA (SLDA) strategy [151], we use a feature extractor  $\mathcal{F}$  pretrained on ImageNet-1000 to obtain low-dimensional data representation  $\mathbf{Z}^{(t)} := \{z_1^{(t)}, z_2^{(t)}, \dots, z_{N_t}^{(t)}\}$ ,  $z_i^{(t)} = \mathcal{F}(x_i^{(t)}) \in \mathbb{R}^d$ . Then we can compute the class mean  $\mu^{(t)} \in \mathbb{R}^d$  and update the shared covariance matrix  $\Sigma^{(1:t)} \in \mathbb{R}^{d \times d}$  after incremental step  $t$  as follows [154]:

$$\mu^{(t)} = \frac{1}{N_t} \sum_{i=1}^{N_t} z_i^{(t)} \quad (4.2)$$

$$\Sigma^{(1:t)} = \frac{(t-1)\Sigma^{(1:t-1)} + \Delta^{(t)}}{t}, \quad (4.3)$$

where  $\Delta^{(t)} = \frac{(t-1)(Z^{(t)} - \mu^{(t)})(Z^{(t)} - \mu^{(t)})^\top}{t}$  and  $(Z^{(t)} - \mu^{(t)}) := (z_1^{(t)} - \mu^{(t)}, z_2^{(t)} - \mu^{(t)}, \dots, z_{N_t}^{(t)} - \mu^{(t)}) \in \mathbb{R}^{d \times N_t}$ . In SLDA, the regularized version of LDA is implemented by applying shrinkage regularization to covariance matrix:  $\Lambda^{(1:t)} = [(1-\varepsilon)\Sigma^{(1:t)} + \varepsilon\mathbf{I}]^{-1}$ , where  $\mathbf{I}$  is an identity matrix of the corresponding dimension.

At the inference stage, after learning all class means  $\mu^{(t)}$ ,  $t = 1, 2, \dots, T$ , and shared covariance matrix  $\Sigma^{(1:T)}$  (and  $\Lambda^{(1:t)}$  as a result), we can make a prediction for a new test sample  $x$  as follows:

$$c = \operatorname{argmax}_{i=1,2,\dots,T} (\mathbf{W}\mathcal{F}(x) + \mathbf{b})_i, \quad (4.4)$$

where  $\mathbf{W} = \mathbf{M}^{(1:T)}\Lambda^{(1:T)}$ , rows of  $\mathbf{M}^{(1:T)}$  are mean vectors  $\mu^{(t)}$  ( $t = 1, 2, \dots, T$ ), and  $\mathbf{b}_i = -\frac{1}{2}\mu^{(i)}\Lambda^{(1:T)}\mu^{(i)}$ .

Unlike previous task prediction strategies [91, 104], with LDA we can predict the task ID with a single test sample, rather than with a batch of samples, representing an important advantage. This is possible because each task consists of defects of the same type and can be described well by a normal distribution with class means  $\mu^{(t)}$  and common covariance matrix  $\Sigma^{(1:T)}$ .

4

## 4.4 NUMERICAL EXPERIMENTS

We evaluate our LDA-CP&S approach on the SD-saliency-900 [139] and Magnetic tile defects [131] datasets, comparing with the following scenarios:

- joint training: the model has access to all data at each incremental step. This case is an upper bound for rehearsal-based methods.
- finetuning: the model is trained at each incremental step  $t$  without preventing forgetting, i.e., we finetune the model to a new task  $t$  that is pretrained on previous tasks  $1, 2, \dots, t-1$ , inevitably causing forgetting of previous tasks because the network parameters (weights and biases) are updated for task  $t$ .
- regularization-based continual learning methods: LwF, MAS that penalize important parameters from changing (see Section 4.2, Eq. 4.1), in an attempt to alleviate forgetting.

We do not consider rehearsal-based approaches that replay a small portion of data from previous tasks while learning a new one because our premise is that old data is not available and should not be used. Furthermore, our comparative investigation of the proposed LDA-CP&S method with others includes the joint training strategy, which is an upper bound for rehearsal-based methods, where all data is available at each incremental step. Therefore, if we show that LDA-CP&S performs similarly to joint training, there is no need to consider rehearsal-based continual learning methods.

As performance metrics, we follow other segmentation works and use the mean Intersection over Union score. For ground truth  $Y$  and prediction  $\hat{Y}$ , IoU score is computed as follows:

$$\text{IoU}(\hat{Y}, Y) = \frac{|Y \cap \hat{Y}|}{|Y \cup \hat{Y}|}. \quad (4.5)$$

To train the model, we use IoU loss which leads to better performance in our experiments than other losses, e.g., Tversky loss [155] and Focal loss [156]. However, it is worth noting

that the difference in IoU scores between models trained with different loss functions is not significant. The IoU loss is computed as follows:

$$\text{IoULoss}(\hat{P}, Y) = 1 - \frac{\sum_{i=1}^H \sum_{j=1}^W \hat{P}_{ij} \cdot Y_{ij} + \varepsilon}{\sum_{i=1}^H \sum_{j=1}^W \hat{P}_{ij} + Y_{ij} - \hat{P}_{ij} \cdot Y_{ij} + \varepsilon}, \quad (4.6)$$

where  $\hat{P}_{ij} \in [0, 1]$  are the output probabilities, and  $H$  and  $W$  are the height and width of the output image,  $\varepsilon$  is a smoothing parameter.

#### 4.4.1 SD-SALIENCY-900 DATASET

In the case of the SD-saliency-900 dataset, we consider a smaller version of U-Net with 16, 32, 64 and 128 in the encoder block and 256 channels in the bottleneck because it consists of only three types of defects – Scratches, Patches and Inclusion – with 300 images per defect. The original size of the images is  $200 \times 200$  but we resize the images to  $224 \times 224$  to make them acceptable for U-Net. We train the segmentation model for 70 epochs with 8 images in a batch, using Adam [66] optimizer and learning rate 0.001. During the pruning stage, we use  $\alpha = 0.9$  and 3 pruning iterations. More details about the influence of hyperparameters on the results are shown in Section 4.4.3. As it is common in continual learning literature [19], we consider different task orderings in our experiments. We can construct six task orderings for the current dataset (e.g., Patches  $\rightarrow$  Scratches  $\rightarrow$  Inclusion).

Table 4.1: Classification accuracy (%) for SD-saliency-900 dataset. The numbers are averaged over all six orderings.

	Scratches	Patches	Inclusion	Average
accuracy (%)	98.33	100	100	99.44

First, we have to make sure that LDA can accurately predict the defect type in an incremental manner. Table 4.1 illustrates the classifier’s accuracy for each defect averaged over all six orders. We can observe the high performance of LDA, misclassifying only a few images from the Scratches dataset. Since 60 images were selected to test each defect type, the prediction error presented corresponds to only 1 misclassified image.

In Figure 4.4, we show mIoU score after every incremental step for every task order. Regularization-based methods only slightly outperform finetuning strategy, while our LDA-CP&S shows comparable results to joint training. Poor performance of the regularization methods can be explained by the lack of a task-specific output layer, which is present in classification network architectures as a classification head. Therefore MAS and LwF update all parameters but change them slightly less than finetuning. On the contrary, LDA-CP&S creates fixed task-specific subnetworks that can overlap and transfer knowledge between each other. Since LDA predicts the defect type (i.e., subnetwork) well at the inference stage, we almost do not have any losses in segmentation performance. We also do not observe network saturation, i.e., the situation when the model does not have enough free space to learn a new task, even though we use a smaller version of U-Net.



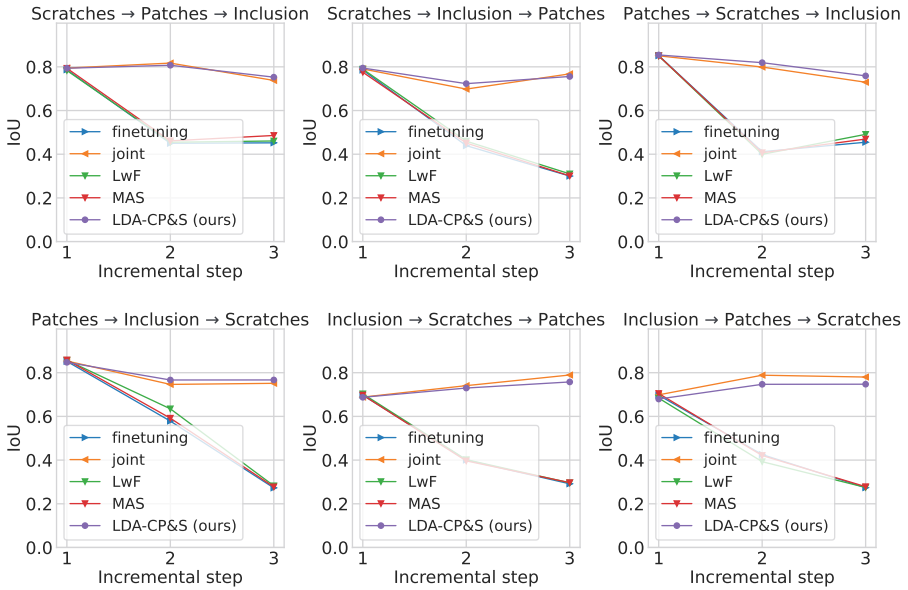


Figure 4.4: IoU score after every incremental step for SD-saliency-900 dataset. The results are presented for all six possible defect orderings.

#### 4.4.2 MAGNETIC TILE DEFECTS DATASET

Magnetic tile defects dataset [131] contains five types of defects, namely Blowhole, Break, Crack, Fray and Uneven, and images that are free from defects (Free). In this work, we consider only images with defects, i.e., five classes. Since the number of defects is higher in this case, we use a U-Net of the original size with 64, 128, 256, and 512 in the encoder block and 1024 channels in the bottleneck. All images in the dataset have different image sizes and, therefore, we resize them to  $224 \times 224$ . For every defect, we randomly select 80% images for training and the rest for testing. U-Net is trained for 150 epochs with 8 images in a batch, using Adam optimizer and learning rate 0.0001. Since the total number of possible task orderings is quite large ( $5! = 120$ ), we consider only five of them at random and we do not have reason to believe that the final performance would be very different when choosing other orderings:

- Blowhole → Break → Crack → Fray → Uneven;
- Break → Uneven → Fray → Crack → Blowhole;
- Crack → Blowhole → Break → Uneven → Fray;
- Fray → Crack → Uneven → Blowhole → Break;
- Uneven → Fray → Blowhole → Break → Crack,

where each defect type appears exactly once for each ordering.

One of the main difficulties with this dataset is class imbalance. Table 4.2 presents LDA accuracy with the same feature extractor considered in the previous example: the pretrained EfficientNet-B5 architecture. Overall, our classification model is able to identify correctly four out of five types of defects, having some difficulties with the Fray sub-dataset that contains the smallest number of images. The only mistake was done in the Fray sub-dataset where we have only 7 test images, meaning that only one image is classified wrongly.

Table 4.2: Classification accuracy (%) for Magnetic tile dataset and the total size of the dataset. The numbers for accuracy are averaged over all five orderings.

	Blowhole	Break	Crack	Fray	Uneven	Average
# train (test) images	92 (23)	92 (22)	68 (17)	25 (7)	72 (21)	N/A
test accuracy (%)	100	100	100	85.71	100	98.75

We would like to highlight the necessity of pretraining the network that extracts features for LDA. The pretrained EfficientNet-B5 produces lower dimensional embeddings that can be used for training and classification with an accuracy of 98.75%, misclassifying only one test image. Meanwhile, if we were to consider a feature extractor with random parameters it would compress the input images in such a way that the LDA classifier would only achieve 16.24% of accuracy.

In Figure 4.5, we present the mIoU score after every incremental step, comparing our LDA-CP&S with other continual learning methods. As we saw in the previous example, regularization-based methods do not handle this type of segmentation problem well. The tasks that we constructed from the Magnetic tile dataset can be quite dissimilar having significant differences in defects areas. Therefore, by updating all the parameters without having task-specific ones, regularization-based approaches are only slightly better than simple finetuning where no anti-forgetting measures are considered. In contrast, our LDA-CP&S creates task-specific parameters for each defect, fixing the values of the parameters once they are assigned to a subnetwork (i.e., defect type or task ID). This allows LDA-CP&S to deal with sequences of tasks as well as joint training, which is very encouraging because joint training is a performance upper bound since all the data is available at each incremental step.

We also investigated how the mIoU score changes for every task after each incremental step. In Figure 4.6, we consider one of the task orderings: Fray  $\rightarrow$  Crack  $\rightarrow$  Uneven  $\rightarrow$  Blowhole  $\rightarrow$  Break. The figure clearly shows the advantage of our algorithm over regularization-based ones because they are heavily dependent on the similarity of the tasks in the order. For example, learning the Break sub-dataset (the last incremental step) improves performance on Fray and Crack sub-datasets compared to the previous incremental step for MAS, LwF and finetuning strategies. However, Uneven is totally forgotten after the network is trained on the Blowhole sub-dataset.

On the contrary, LDA-CP&S does not forget previous tasks and is still able to learn new ones even having fewer free parameters. It has comparable performance with a single-task scenario, where a separate U-Net is trained for every task. Also, we observe that task-wise

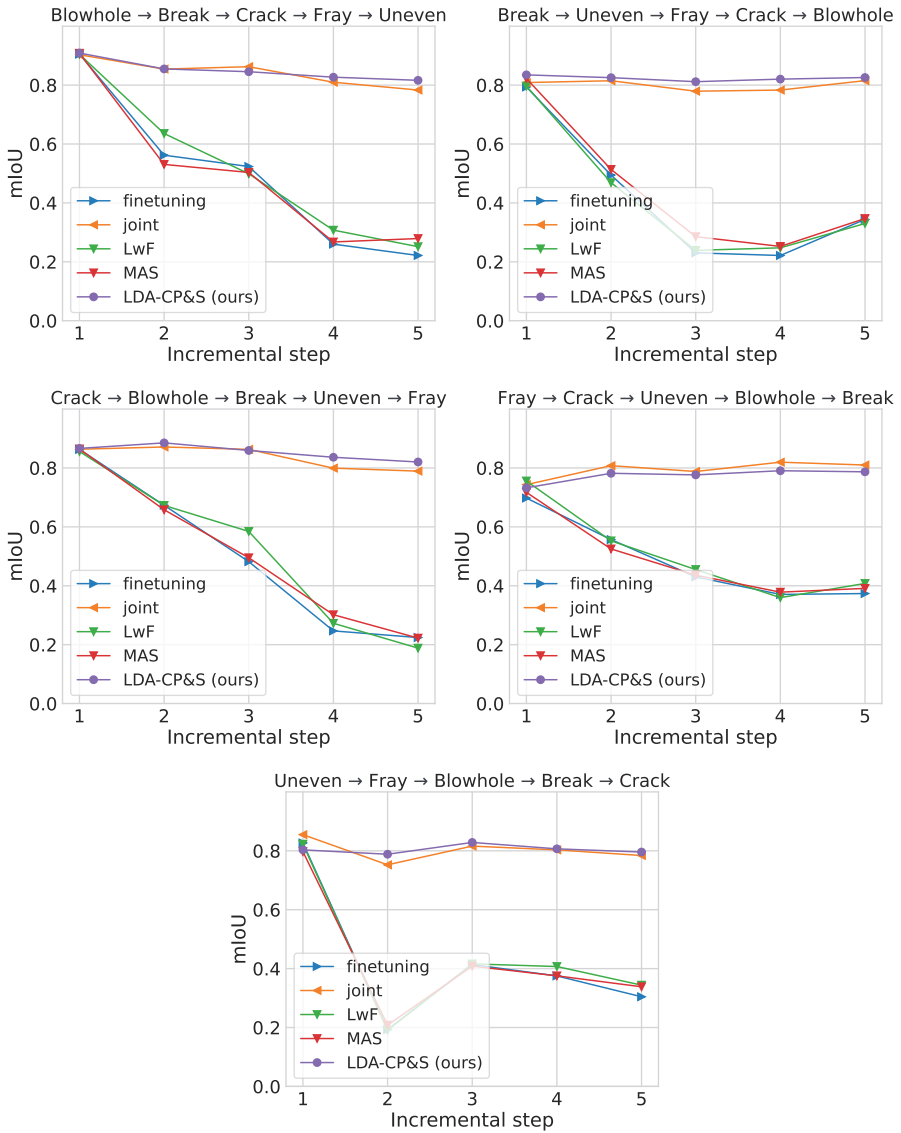


Figure 4.5: IoU score after every incremental step for Magnetic tile datasets. The results are shown for all five selected defect orderings.

performance is almost the same as for joint training, meaning the subnetwork overlaps provide enough knowledge transfer to learn a new task.

Figure 4.7 illustrates the model output in every learning scenario. We observe that regularization-based methods and finetuning cannot capture the defects of the first tasks

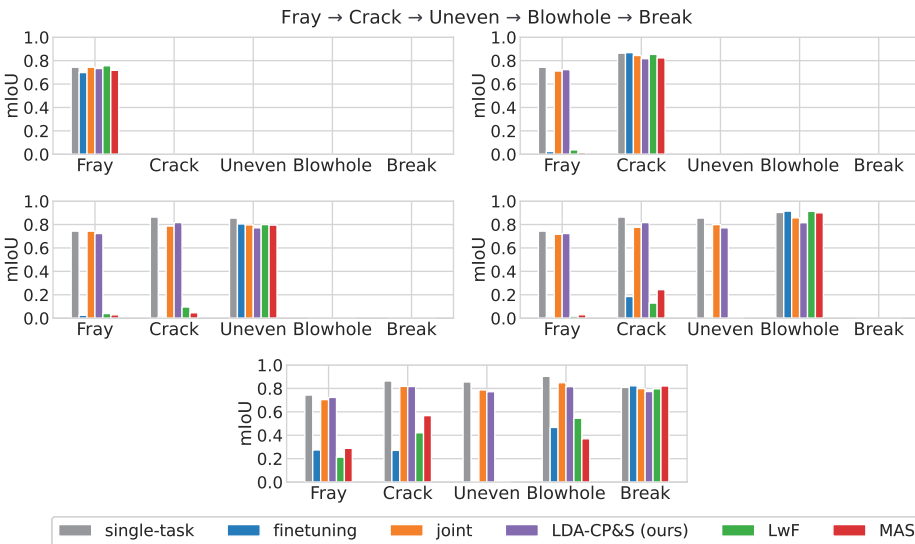


Figure 4.6: IoU score after every incremental step for one of the defects orderings from the Magnetic tile dataset.

in the sequence, while our LDA-CP&S finds defects' segments close to the joint training.

### 4.4.3 HYPERPARAMETERS CHOICE

The choice of hyperparameters for pruning has a significant impact on subnetwork sparsity and, as a result, performance. In this subsection, we compare different options for the pruning hyperparameter  $\alpha$  and the number of pruning iterations. A lower number of  $\alpha$  and a higher number of pruning iterations lead to higher sparsity (more free connections to learn future tasks) but may cause lower segmentation performance. Also, the values for hyperparameters depend on the length of task sequences. In our work, we pre-define these hyperparameters at the beginning and do not change them during the training process.

Figure 4.8 illustrates how different pairs of hyperparameters affect the training process for our approach. For both datasets, we clearly see that the network starts to saturate if pruning is not aggressive enough (e.g.,  $\alpha = 0.95$  where most of the signal is conserved) because the network does not have enough free parameters for new tasks. In the case of the SD-saliency-900 dataset, we can also observe the trade-off between sparsity and mIoU score: with  $\alpha = 0.9$  it is clear that pruning the network twice leads to better performance than doing it three times, as the subnetwork that results is less expressive (has fewer parameters). The results on the Magnetic tile dataset show the trade-off between learning the first tasks and the last ones: if we prune the network twice,  $\alpha = 0.9$  leads to better performance if there are no more than three tasks, while  $\alpha = 0.85$  is better suitable for longer task sequences.

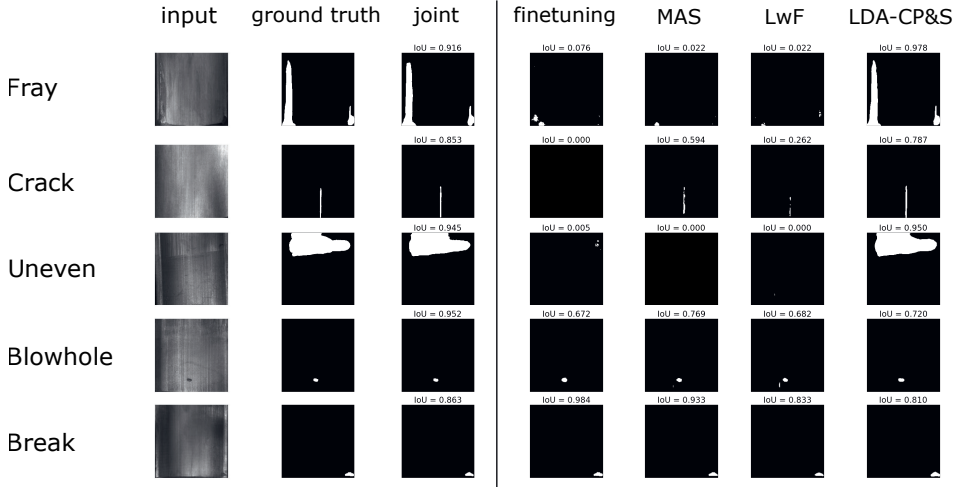


Figure 4.7: Visualization of defects prediction for the considered scenarios on Fray → Crack → Uneven → Blowhole → Break task ordering.

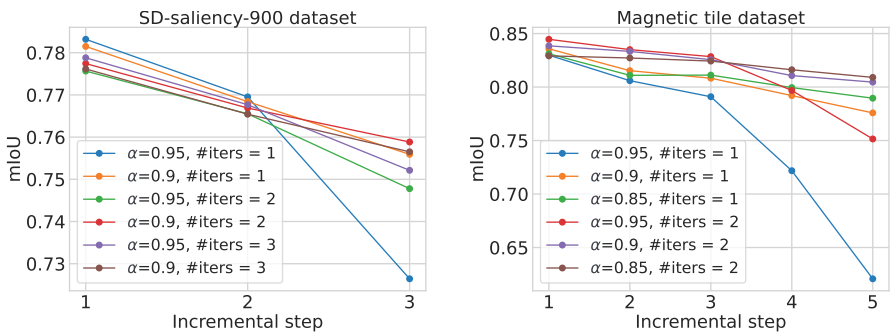


Figure 4.8: Comparison between different pairs of hyperparameters for the pruning step in our LDA-CP&S method on SD-saliency-900 dataset (left) and Magnetic tile dataset (right). The results after each incremental step averaged over the number of considered task orderings.

## 4.5 CONCLUSION

The LDA-CP&S method that we have proposed successfully learns to segment the defects incrementally, without any forgetting, using only the data that is given at the current time step. Meanwhile, other methods that do not use data from previous tasks fail to remember all tasks, exhibiting considerable forgetting in segmenting previously seen defects. Overall, the performance of LDA-CP&S is more than two times higher in terms of mean Intersection over Union score for the two datasets considered herein when compared to other continual learning methods.

This Chapter aims to illustrate the importance of a precise subnetwork selection strategy. By training the LDA classifier and eliminating the issue of requiring multiple images for subnetwork prediction, we have obtained a powerful continual learning model that shows comparable performance as the scenario where all data is available at every incremental step. Thus we want to point out the necessity of developing other task prediction approaches for a more general class of problems.



## **PART II: APPLICATION**





## 5

# INCREMENTAL LEARNING FOR PHYSICS-INFORMED NEURAL NETWORKS

5

*In previous chapters, we have described a general approach to tackling catastrophic forgetting issues in deep neural networks. The proposed solution is based on constructing task-related subnetworks during training and subnetwork prediction, if necessary, to make a prediction. In some cases, the information about the task being solved is provided, which means that subnetwork selection is not required. In this chapter, we consider one of these examples.*

*This Chapter introduces an incremental learning procedure for physics-informed neural networks (PINNs), which have recently become a powerful tool for solving partial differential equations (PDEs). Our goal is to show that by learning multiple PDEs incrementally with the proposed in the first part paradigm, the algorithm can improve the generalization capability of PINNs compared to conventional training. In addition, the algorithm does not exhibit significant forgetting of old PDEs.*

## 5.1 INTRODUCTION

Deep neural networks (DNNs) play a central role in scientific machine learning (SciML). Recent advances in neural networks find applications in real-life problems in physics [157–160], medicine [161–163], finance [164–167], and engineering [168–171]. In particular, they are also applied to solve Ordinary Differential Equations and Partial Differential Equations (ODEs/PDEs) [23, 172–174]. Consider the following PDE,

$$\mathcal{F}[u(\mathbf{x}, t)] = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, t \in (t_0, T], \quad (5.1)$$

$$\mathcal{B}[u(\mathbf{x}, t)] = b(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \quad (5.2)$$

$$u(\mathbf{x}, t_0) = h(\mathbf{x}), \quad \mathbf{x} \in \Omega, \quad (5.3)$$

where  $\mathcal{F}$  is a differential operator,  $\mathcal{B}$  is a boundary condition operator,  $h(\mathbf{x})$  is an initial condition, and  $\Omega$  is a bounded domain.

The first neural network-based approaches incorporated a form of the equation into the loss function with initial and boundary conditions included as hard constraints [175, 176]. However, these works used relatively small neural networks with one or two hidden layers. On the contrary, PINNs [23] encode initial and boundary conditions as soft constraints into the loss function of a DNN. Subsequently, PINNs and their extensions found applications in fluid mechanics [177–179], inverse problems [180–182] and finance [23, 183]. Later, the generalized version of PINNs, called XPINNs [184], was proposed by decomposing the domain into multiple subdomains. However, this method uses as many networks as the number of subdomains, increasing the algorithm’s complexity. Almost simultaneously with our work, Multi-head PINNs (MH-PINNs) [185] have been proposed as a multi-task and meta-learning approach for PINNs that is employed to learn stochastic processes, synergistic learning of PDEs and uncertainty quantification. MH-PINNs have a shared part of the network and task-specific output heads for prediction. Therefore, it uses additional parameters for every head, increasing the model’s size with respect to the number of tasks, without sharing knowledge between them. In addition, the parameters in MH-PINN are shared between all tasks in the non-output layer, which is a limitation if the tasks are very different as the authors noted [185]. Other meta-learning approaches were also employed in the context of PINNs [186, 187]. However, meta-learning literature focuses on obtaining good initialization for a new task given some tasks for pretraining. Unfortunately, once the network is adapted to a new task, it loses the ability to solve the previous ones, i.e. it undergoes *catastrophic forgetting* of other tasks. We take a different route inspired by the incremental learning and continual (or lifelong) learning literature, as discussed below.

Despite the popularity of DNNs, and PINNs in particular, there are few *incremental* learning algorithms available in SciML literature. Yet, incremental learning and continual learning algorithms [97, 99, 188] are capable of handling tasks sequentially, instead of altogether as in multi-task learning and other strategies. Moreover, they are still capable of not forgetting how to solve all of the previously learned tasks. If tasks have some similarities with each other, new tasks have the potential of being learned better (i.e., faster or with lower testing error) with the help of previously learned ones. The goal of this work is to propose an incremental learning algorithm for PINNs such that similar symbiotic effects can be obtained.

**Background and main challenges** PINNs formulate the PDE solution problem by including initial and boundary conditions into the loss function of a neural network as soft constraints. Let us denote the output of the network  $\mathcal{N}$  with learnable parameters  $\theta$  as  $\hat{u}(\theta, \mathbf{x}, t) = \mathcal{N}(\theta; \mathbf{x}, t)$ . Then sampling the set of collocation points, i.e. a set of points in the domain,  $\mathcal{CP} = \{(x^i, t^i) : x^i \in \text{int } \Omega, t^i \in (t_0, T], i = 1, 2, \dots, N_{\mathcal{F}}\}$ , the set of initial points  $\mathcal{IP} = \{(x^j, t_0) : x^j \in \partial\Omega, j = 1, 2, \dots, N_{u_0}\}$  and the set of boundary points  $\mathcal{BP} = \{(x^k, t^k) : x^k \in \partial\Omega, t^k \in (t_0, T], k = 1, 2, \dots, N_b\}$  one can write the optimization problem and loss function arising from PINNs as follows:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{F}}(\theta) + \mathcal{L}_{u_0}(\theta) + \mathcal{L}_b(\theta) \rightarrow \min_{\theta}, \quad (5.4)$$

$$\mathcal{L}_{\mathcal{F}}(\theta) = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathcal{F}}} \|\mathcal{F}[\hat{u}(\theta, x^i, t^i)] - f(x^i)\|^2, \quad (x^i, t^i) \in \mathcal{CP}, \quad (5.5)$$

$$\mathcal{L}_{u_0}(\theta) = \frac{1}{N_{u_0}} \sum_{j=1}^{N_{u_0}} \|\hat{u}(\theta, x^j, t_0) - h(x^j)\|^2, \quad (x^j, t_0) \in \mathcal{IP}, \quad (5.6)$$

$$\mathcal{L}_b(\theta) = \frac{1}{N_b} \sum_{k=1}^{N_b} \|\mathcal{B}[\hat{u}(\theta, x^k, t^k)] - b(x^k)\|^2, \quad (x^k, t^k) \in \mathcal{BP}. \quad (5.7)$$

However, sometimes PINNs struggle to learn the ODE/PDE dynamics [189–192] (see Figure 5.1). Wight & Zhao [193] proposed several techniques to improve the optimization process compared to the original formulation: mini-batch optimization and adaptive sampling of collocation points. Adaptive sampling in time, splits the time interval  $[t_0, T] = \cup_{k=0}^K [t_{k-1}, t_k]$ ,  $t_K = T$ , and solves an equation on the first interval  $[t_0, t_1]$ , then on  $[t_0, t_2]$ , and so on up to  $[t_0, T]$ . Thus, if a solution can be found on a domain  $\Omega \times [t_0, t_{k-1}]$ , then the network is pretrained well for the extended domain  $\Omega \times [t_0, t_k]$ . Krishnapriyan et al. [190] proposed the *seq2seq* approach that splits the domain into smaller subdomains in time and learns the solution on each of the subdomains with a separate network. Thus, both adaptive sampling in time and *seq2seq* are based on the idea of splitting the domain into multiple subdomains, on which solutions can be learned more easily.

As explained in [191], improving PINN's solutions by considering small subdomains is possible because the loss residuals ( $\mathcal{L}_{\mathcal{F}}$  term) can be trivially minimized in the vicinity of fixed points, despite corresponding to nonphysical system dynamics that do not satisfy the initial conditions. Therefore, the reduction of the domain improves the convergence of the optimization problem (5.4) and helps to escape nonphysical solutions.

We propose *incremental PINNs* (iPINNs) and implement this strategy by creating one subnetwork per task such that a complete neural network can learn multiple tasks. Each subnetwork  $\mathcal{N}_i$  has its own set of parameters  $\theta_i \subset \theta$ , and the model is trained sequentially on different tasks. A subnetwork for a new task can overlap with all previous subnetworks, which helps to assimilate the new task. As a result, the network consists of overlapping subnetworks, while the free parameters can be used for future tasks. To illustrate the benefits of the algorithm we consider two problem formulations (Section 5.3). Firstly, we learn a family of equations (e.g., convection) starting from a simple one and incrementally learning new equations from that family. Secondly, we learn a dynamical system that

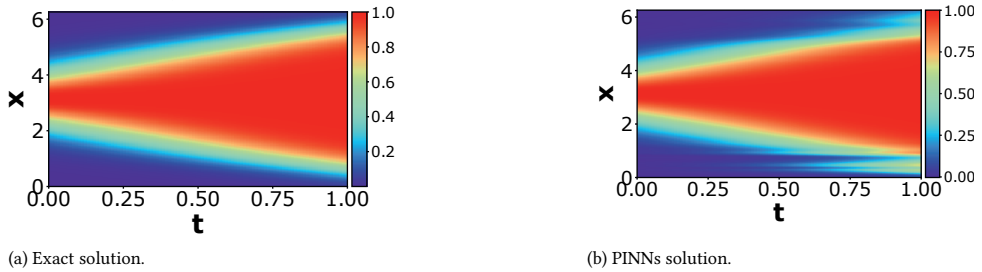


Figure 5.1: 1-D reaction equation with parameter  $\rho = 5$  (see P1.2).

consists of two processes (e.g., reaction-diffusion) by first learning the individual components of the process. Both scenarios demonstrate that the incremental approach enables an iPINN network to learn for cases where regular PINNs fail. To the best of our knowledge, this is the first example where one network can sequentially learn multiple equations without extending its architecture, with the added benefit that performance is significantly improved.

5

## 5.2 RELATED WORK

Our methodology is based on creating task-specific subnetworks that transfer knowledge between each other. Also, similarly to other PINN research, the algorithm is sensitive to the choice of activation functions. We briefly highlight key related work on these topics herein.

**Transfer learning** Transfer learning is commonly used in computer vision and natural language processing [194–197]. It tries to improve the optimization process by starting with better weight initialization. In PINNs, transfer learning is also successfully used to accelerate the loss convergence [198–201]. For instance, Chen et al. [202] apply transfer learning to learn faster different PDEs creating tasks by changing coefficients or source terms in equations. Analogously, curriculum regularization (similar to curriculum learning [203]) is proposed in [190] to find good initial weights.

**Choice of the activation function** There are several studies that investigate how different activation functions affect the performance of neural networks in classification and regression tasks [204, 205]. It was shown that ReLU [206] activation function which can be powerful in classification tasks, in the case of physics-informed machine learning (PIML) regression, may not be the optimal choice. Meanwhile, hyperbolic tangent ( $\tanh$ ) or sine ( $\sin$ ) perform well for PIML. Sinusoidal representation networks (SIRENs) [207] tackle the problem of modeling the signal with fine details. Special weights initialization scheme combined with  $\sin$  activation function allows SIREN to learn complex natural signals. Hence, we use  $\sin$  activation function in our experiments. In Section 4.4.3, we provide the comparison in results between the discussed activation functions.

### 5.3 PROBLEM FORMULATION

We focus on two scenarios: (1) incremental PINNs learning, where the network sequentially learns several equations from the same family; and (2) learning a combination of multiple equations that create another physical process. To illustrate these cases, we consider one-dimensional convection, reaction and reaction-diffusion problems with periodic boundary conditions.

#### SCENARIO 1: EQUATION INCREMENTAL LEARNING

We consider the problem of learning the sequence of equations that belong to one family:

$$\mathcal{F}_k[u(x, t)] = 0, \quad x \in \Omega, t \in [t_0, T], k = 1, 2, \dots, \quad (\text{P1})$$

where  $\mathcal{F}_k, k = 1, 2, \dots$  are differential operators from the same family of equations.

##### 1-D convection equation

$$\begin{aligned} \frac{\partial u}{\partial t} + \beta_k \frac{\partial u}{\partial x} &= 0, & (\text{P1.1}) \\ u(x, 0) &= h_1(x), \\ u(0, t) &= u(2\pi, t), \end{aligned}$$

##### 1-D reaction equation

$$\begin{aligned} \frac{\partial u}{\partial t} - \rho_k u(1 - u) &= 0, & (\text{P1.2}) \\ u(x, 0) &= h_2(x), \\ u(0, t) &= u(2\pi, t), \end{aligned}$$

where  $t \in (0, 1], x \in [0, 2\pi], \beta_k \in \mathcal{B} \subset \mathbb{R}$ . where  $t \in (0, 1], x \in [0, 2\pi], \rho_k \in \mathcal{R} \subset \mathbb{R}$ .

In this case, every task  $k$  is associated with  $\mathcal{D}_k = \{(x, t, k) : x \in [0, 2\pi], t \in [t_0, T], k \in \mathbb{N}\}$ .

Following [190], we take  $h_1(x) = \sin x$  and  $h_2(x) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}}$ .

#### SCENARIO 2: COMBINATION OF MULTIPLE EQUATIONS

We also consider the case when a dynamic process consists of multiple components. Let us consider the reaction-diffusion equation:

$$\begin{aligned} \frac{\partial u}{\partial t} - v \frac{\partial^2 u}{\partial x^2} - \rho u(1 - u) &= 0, & (\text{P2}) \\ u(x, 0) &= h_2(x), \\ u(0, t) &= u(2\pi, t), \end{aligned}$$

where  $t \in [0, 1], x \in [0, 2\pi], v, \rho > 0$ . This process consists of two parts: reaction term ( $v = 0$ ):  $-\rho u(1 - u)$  and diffusion term ( $\rho = 0$ ):  $-v \frac{\partial^2 u}{\partial x^2}$ . Therefore, we construct one task as the reaction, another one as the diffusion, and the final one as the reaction-diffusion. We can change the order of the reaction tasks and diffusion tasks to show the robustness of incremental learning. The reaction-diffusion task should be the last one since our goal is first to learn the components of the system and only then the full system.

Considering these two problems, we want to show that better generalization can be achieved by pretraining the network with simpler related problems rather than by dividing the domain into smaller subdomains. In the following section, we show how one network can incrementally learn different equations without catastrophic forgetting.

## 5.4 PROPOSED METHOD

The proposed method needs to be applicable to both types of problems P1 and P2. However, these problems cannot be solved by one network with the same output head for every different task, since  $\mathcal{F}_i[u(x,t)] \neq \mathcal{F}_j[u(x,t)]$  for  $i \neq j$  and  $x \in \Omega$ ,  $t \in [t_0, T]$ . Instead, the incremental learning algorithm we propose (iPINNs) focuses on learning task-specific subnetworks  $\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_k, \dots$  for each task  $k$ .

We start by creating the above-mentioned subnetworks using an iterative pruning algorithm that we have developed in Chapter 2. Other pruning strategies could be considered, without loss of generality – see Remark 5.1.

**Remark 5.1** *In principle, any connections pruning algorithm or any other approach that is able to find and train sparse network representations is suitable for the iPINNs strategy we propose herein. However, most pruning algorithms aim at reducing memory requirements or reducing inference time, instead of aiming at subnetwork creation with the smallest number of neuron connections. NNrelief was developed with this in mind, so it creates sparser subnetworks for a given performance level when compared to state-of-the-art methods, as we showed in Chapter 2 for multiple datasets. This makes it particularly interesting for iPINNs, as the subnetworks we generate are smaller and leave additional free connections for subsequent incremental training.*

5

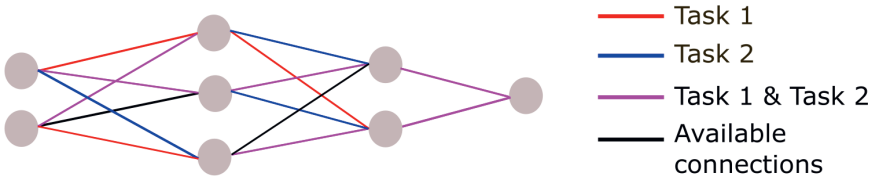
An important concept in the proposed iPINN strategy is that the pruning method is used to train task-specific subnetworks, but allowing the subnetworks to *naturally* overlap on some connections (see Figure 5.2). This way the method provides knowledge sharing between the subnetworks. These overlaps are updated with respect to all tasks that are assigned to a particular connection. Let us denote the loss of each task  $\mathcal{D}_j$  as  $\mathcal{L}_j = \mathcal{L}(\theta_j; \mathcal{D}_j)$ , where  $\theta_j$  is the parameter vector for task  $\mathcal{D}_j$ ,  $1 \leq j \leq k$ . Then the total loss and its gradient with respect to a parameter  $w$  can be written as:

$$\mathcal{L} = \sum_{j=1}^k \mathcal{L}_j, \quad (5.8)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{j=1}^k \frac{\partial \mathcal{L}_j}{\partial w} = \sum_{j: w \in \mathcal{N}_j} \frac{\partial \mathcal{L}_j}{\partial w}, \quad (5.9)$$

because if  $w \notin \mathcal{N}_j$ , then  $\frac{\partial \mathcal{L}_j}{\partial w} = 0$ .

Algorithm 5 includes the pseudocode for iPINNs. For every new task  $k$  that enters the network, we first find a corresponding subnetwork  $\mathcal{N}_k$  with NNrelief (line 4 of Algorithm 5), then adapt the overlaps between previous subnetworks  $\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_{k-1}$  and a new one  $\mathcal{N}_k$  (line 5 of the Algorithm 5). We can prune a (sub)network multiple times (hyperparameter `num_iters`) to achieve a lower sparsity level, however, this is computationally expensive. Therefore we prune every network only once and control the sparsity level with parameter  $\alpha$ .



$$\mathcal{L}(\theta; \mathcal{D}_1 \cup \mathcal{D}_2) = \mathcal{L}_1(\theta_1; \mathcal{D}_1) + \mathcal{L}_2(\theta_2; \mathcal{D}_2)$$

Figure 5.2: An example of iPINNs with two PDEs: every subnetwork corresponds to only one task (PDE).

---

**Algorithm 5** PINN incremental learning: adding new task  $k$

---

**Input:** neural network  $\mathcal{N}$ , training datasets  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_{k-1}$  and  $\mathcal{D}_k$ , training hyperparameters, pruning hyperparameters ( $num\_iters, \alpha$ ).

- 1:  $\mathcal{N}_k \leftarrow \mathcal{N}$  ▷ set full network as a subnetwork
  - 2: Train  $\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_k$  on tasks  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k$  using Eq. 5.9. ▷ training step
  - 3: **for**  $it = 1, 2, \dots, num\_iters$  **do** ▷ repeat pruning
  - 4:      $\mathcal{N}_k \leftarrow \text{NNrelief}(\mathcal{N}_k, \mathcal{D}_k, \alpha)$  ▷ pruning step: Algorithm 1
  - 5:     Retrain subnetworks  $\mathcal{N}_1, \mathcal{N}_2, \dots, \mathcal{N}_k$  on tasks  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k$  using Eq. 5.9. ▷ retraining step
  - 6: **end for**
- 

**Remark 5.2** *The pruning strategy allows us to have more flexible variation in parameter sharing because we can keep task-specific parameters within a subnetwork that are not shared with other subnetworks, but we can also keep parameters that are shared among different subnetworks. Task-specific parameters are shown by the red and blue connections in Figure 5.2, and they result from pruning the entire network and training free connections (in black) that are unused. The magenta connections in Figure 5.2 highlight cases where their parameters are being shared across different tasks, and they result from the pruning algorithm not removing those connections when training for a new task.*

The main advantage of the proposed approach is that a neural network learns *all* tasks (equations) that were given during training and not only the last one. This is achieved by constantly replaying old data. Data for previous tasks is easily available by sampling collocation points, which eliminates all issues of data replaying for continual learning problems in computer vision and natural language processing tasks and makes the algorithm well-suited in the context of PINNs. We want to emphasize that iPINN does not need to know how many tasks will be handled overall, and it accesses only those that were considered up to task  $k$  inclusive, which distinguishes it from multi-task learning. In the next section, we experimentally show that pretrained parts of the network help to improve the convergence process.



## 5.5 NUMERICAL EXPERIMENTS

Our findings illustrate the advantage of Algorithm 5 over regular PINNs [23]. The Algorithm allows the network to learn multiple equations (P1) from the same family. Furthermore, by starting with simpler tasks, the network can subsequently learn more complex ones that cannot be learned in isolation.

**Experiments setup** Let us start by examining the proposed algorithms on the convection and reaction equations with periodic boundary conditions (P1). Following the setup in [190], we use a four-layer neural network with 50 neurons per layer. We use 1000 randomly selected collocation points on every time interval between 0 and 1 for  $\mathcal{L}_{\mathcal{P}}$ . Adam is used as the optimizer [66] with a learning rate of 0.01 and 20000 epochs to train the model. We divide the learning rate by 3 every 500 epochs in which the loss does not decrease. We repeat our experiments multiple times with different random initializations of the network parameters and show the average values of error.

To evaluate the performance of the algorithms we compare the final error after the last task. In addition, following continual learning literature [110], we compare backward and forward transfer metrics. Let us denote the test set as  $\mathcal{D}^{test} = \{(x^i, t^i, l) : x^i \in [0, 2\pi], t^i \in [0, 1], l \text{ is the task-ID}\}$  and  $N = \#\mathcal{D}^{test}$ , the solution of the equation at the point  $(x^i, t^i, l)$  as  $\mathbf{u}_{l,k}^i = u_{l,k}^i(x^i, t^i)$ , and  $\hat{\mathbf{u}}_{l,k}^i$  is a prediction of the model at point  $(x^i, t^i, l)$  after task  $\mathcal{D}_k$  is learned. Relative and absolute errors are denoted as  $r_{l,k}$  and  $\varepsilon_{l,k}$ , respectively, as they are calculated for task  $l$  after task  $k$  is learned ( $l \leq k$ ).

$$\text{Relative error: } r_{l,k} = \frac{\|\mathbf{u}_l - \hat{\mathbf{u}}_{l,k}\|_2}{\|\mathbf{u}_l\|_2} \times 100\%, \quad (5.10)$$

$$\text{Absolute error: } \varepsilon_{l,k} = \frac{1}{N} \sum_{i=1}^N |\mathbf{u}_l^i - \hat{\mathbf{u}}_{l,k}^i|, \quad (5.11)$$

$$\text{Backward Transfer: } \text{BWT} = \frac{1}{k-1} \sum_{l=1}^{k-1} \varepsilon_{l,k} - \varepsilon_{l,l} \quad \text{or} \quad (5.12)$$

$$\text{BWT} = \frac{1}{k-1} \sum_{l=1}^{k-1} r_{l,k} - r_{l,l} \quad (5.13)$$

### 5.5.1 RESULTS

Table 5.1 presents the results after all reaction equations are learned varying  $\rho$  from 1 to 5. Figure 5.3(a) shows the error history for every equation after incremental steps. The Table summarizes the performance improvement of iPINNs compared to regular PINNs, exhibiting negligible error for all values of  $\rho$ , which is especially relevant for cases when  $\rho$  is larger. Moreover, iPINNs provide negative BWT which means that previous subnetworks help to learn the following ones.

Similarly, we observe for the convection equation the same learning behaviour. By learning incrementally the sequence of convection equations, we achieve much lower absolute and relative errors for the equations that are more difficult to learn ( $\beta = 30, 40$ ). In Table 5.2 we show final errors at the end of the training, and Figure 5.3(b) shows the

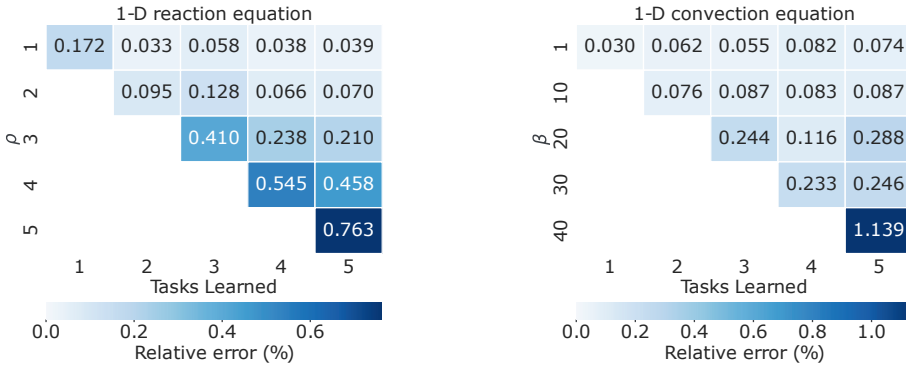


Figure 5.3: Relative error history for reaction equations (a) and convection equations (b). Every row shows the error after a new task is learned.

Table 5.1: Final error and forgetting after all reaction equations are learned.

		regular PINN	iPINN
$\rho = 1$	abs. err	$1.09 \times 10^{-3}$	$1.5 \times 10^{-4}$
	rel. err	0.263%	<b>0.039%</b>
$\rho = 2$	abs. err	$1.97 \times 10^{-3}$	$2.5 \times 10^{-4}$
	rel. err	0.479%	<b>0.070%</b>
$\rho = 3$	abs. err	$6.72 \times 10^{-3}$	$6.1 \times 10^{-4}$
	rel. err	2.05%	<b>0.210%</b>
$\rho = 4$	abs. err	$1.13 \times 10^{-2}$	$1.18 \times 10^{-3}$
	rel. err	3.68%	<b>0.458%</b>
$\rho = 5$	abs. err	$5.04 \times 10^{-2}$	$1.91 \times 10^{-3}$
	rel. err	12.19%	<b>0.763%</b>
BWT	abs. err	N/A	$-3.8 \times 10^{-4}$
	rel. err	N/A	-0.112%

absolute error history for each equation. In this case, we observe some level of forgetting, however, it is insignificant compared to the error values.

In Figures C.1 and C.2, we illustrate the error of iPINNs on convection and reaction equations and the exact solutions for every value of parameter  $\beta$  or  $\rho$  that were considered. Overall, we see that the neural network learns more complicated tasks more accurately if parts of the network are pretrained with easier tasks. At the same time, iPINNs replay the training data for previous PDEs during training for the new one. There are no additional costs to store or generate input points  $(x, t)$  for previous tasks since they can be easily

Table 5.2: Final error and forgetting after all convection equations are learned.

		regular PINN	iPINN
$\beta = 1$	abs. err	$2.3 \times 10^{-4}$	$4.2 \times 10^{-4}$
	rel. err	0.042%	0.074%
$\beta = 10$	abs. err	$1.3 \times 10^{-3}$	$5.0 \times 10^{-4}$
	rel. err	0.222%	0.087%
$\beta = 20$	abs. err	$1.9 \times 10^{-3}$	$1.66 \times 10^{-3}$
	rel. err	0.339%	0.288%
$\beta = 30$	abs. err	$2.2 \times 10^{-1}$	$1.44 \times 10^{-3}$
	rel. err	3.957%	0.246%
$\beta = 40$	abs. err	$2.3 \times 10^{-1}$	$6.02 \times 10^{-3}$
	rel. err	37.4%	1.139%
BWT	abs. err	N/A	$1.8 \times 10^{-4}$
	rel. err	N/A	0.0280%

5

sampled when necessary.

**Remark 5.3** *The proposed iPINNs may require more training epochs than standard PINNs for one PDE because of the subnetwork creation strategy. However, the algorithm pursues a different goal: provide the ability to learn the solutions sequentially sharing previously learned knowledge.*

We also illustrate the effectiveness of the iPINN method by addressing problem P2. We consider the values of  $\rho$  and  $\nu$  for which a PINN does not have difficulties when learning each component of the reaction-diffusion separately. Results obtained when first learning the reaction part (or vice-versa, the diffusion part) are shown in Table 5.3 (Table 5.4). The main finding is that the network can learn every equation at least as well as when it is learned independently. In fact, for the reaction equation, the neural network improves significantly the prediction error. Another interesting observation is that the model learns the reaction-diffusion equation with almost the same error, regardless of the order of the tasks. This gives us a hint about the robustness of the algorithm to different task orders in terms of prediction error. In Section 5.6.2, we analyze the percentages of parameters assigned to every subnetwork to illustrate the same conclusion in terms of the number of allocated parameters.

Table 5.3: Final error and forgetting for reaction  $\rightarrow$  diffusion  $\rightarrow$  reaction-diffusion.

parameters	equation		regular PINN	iPINN
$\rho = 3, \nu = 5$	reaction	abs. err	$6.72 \times 10^{-3}$	$9.41 \times 10^{-4}$
		rel. err	2.05%	0.31%
	diffusion	abs. err	$1.38 \times 10^{-4}$	$1.85 \times 10^{-4}$
		rel. err	0.05%	0.06%
	reaction-diffusion	abs. err	$4.89 \times 10^{-3}$	$4.10 \times 10^{-3}$
		rel. err	0.80%	0.68%
$\rho = 4, \nu = 4$	reaction	abs. err	$1.13 \times 10^{-2}$	$7.88 \times 10^{-3}$
		rel. err	3.68%	2.99%
	diffusion	abs. err	$4.35 \times 10^{-4}$	$5.84 \times 10^{-4}$
		rel. err	0.16%	0.19%
	reaction-diffusion	abs. err	$4.58 \times 10^{-3}$	$4.42 \times 10^{-3}$
		rel. err	0.70%	0.67%
$\rho = 4, \nu = 5$	reaction	abs. err	$5.04 \times 10^{-2}$	$4.20 \times 10^{-3}$
		rel. err	12.19%	1.71%
	diffusion	abs. err	$5.18 \times 10^{-4}$	$2.30 \times 10^{-4}$
		rel. err	0.18%	0.08%
	reaction-diffusion	abs. err	$4.61 \times 10^{-3}$	$4.58 \times 10^{-3}$
		rel. err	0.69%	0.68%

Table 5.4: Final error and forgetting for diffusion  $\rightarrow$  reaction  $\rightarrow$  reaction-diffusion.

parameters	equation		regular PINN	iPINN
$\rho = 3, \nu = 5$	diffusion	abs. err	$1.38 \times 10^{-4}$	$8.64 \times 10^{-4}$
		rel. err	0.05%	0.28%
	reaction	abs. err	$6.72 \times 10^{-3}$	$2.11 \times 10^{-3}$
		rel. err	2.05%	0.68%
	reaction-diffusion	abs. err	$4.89 \times 10^{-3}$	$4.07 \times 10^{-3}$
		rel. err	0.80%	0.67%
$\rho = 4, \nu = 4$	diffusion	abs. err	$4.35 \times 10^{-4}$	$3.45 \times 10^{-4}$
		rel. err	0.16%	0.12%
	reaction	abs. err	$1.13 \times 10^{-2}$	$4.91 \times 10^{-3}$
		rel. err	3.68%	1.97%
	reaction-diffusion	abs. err	$4.58 \times 10^{-3}$	$4.42 \times 10^{-3}$
		rel. err	0.70%	0.67%
$\rho = 4, \nu = 5$	diffusion	abs. err	$5.18 \times 10^{-4}$	$1.05 \times 10^{-3}$
		rel. err	0.18%	0.33%
	reaction	abs. err	$5.04 \times 10^{-2}$	$9.15 \times 10^{-3}$
		rel. err	12.19%	3.39%
	reaction-diffusion	abs. err	$4.61 \times 10^{-3}$	$4.30 \times 10^{-3}$
		rel. err	0.69%	0.65%

## 5.6 ADDITIONAL STUDY

In this section, we provide additional information about the learning procedure of iPINNs. We highlight some important training details such as the presence of regularization and the choice of activation functions. Also, we explore the subnetworks that our approach produces showing the proportion of parameters allocated to each task.

### 5.6.1 SENSITIVITY TO HYPERPARAMETERS

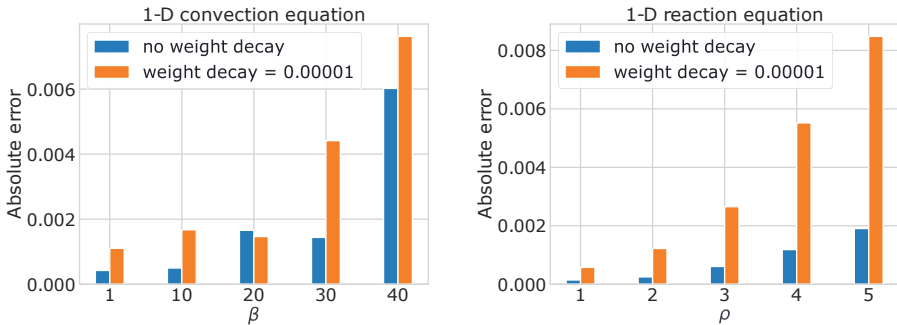


Figure 5.4: Influence of weight decay on the results for reaction (left) and convection (right) equations after all tasks are learned.

Here we illustrate the influence of different training hyperparameters on the performance of iPINNs. First, we compare the results with and without regularization parameter (weight decay). In Figure 5.4, it can be observed that the presence of weight decay worsens the prediction error. However, looking at the result it is clear that iPINNs still work if weight decay is present. We can explain the lack of need for weight decay with the fact that many parameters are assigned to multiple tasks and cannot overfit to a particular one. Each subnetwork is also less parameterized than the original network and therefore does not tend to overfit. Thus, weight decay is not necessary and its presence only worsens the result due to the complication of the optimization procedure.

Furthermore, we compare the performance when using `sin` and `tanh` activation functions for two task orderings in Figure 5.5. We observe that `sin` works significantly better in both cases. Also, we test `ReLU` activation but it exhibits poor performance in both PDE orderings, as expected. If the reaction is learned first, the absolute errors are 0.4959, 0.2369 and 0.1493. If we start with the diffusion equation and then learn reaction and reaction-diffusion PDEs, the errors are 0.2399, 0.2977 and 0.3003.

In addition, we present how different values of pruning parameter  $\alpha$  affect the results. The higher the value of  $\alpha$  is, the less the network is pruned. Therefore, if  $\alpha = 0.95$  the task-specific subnetworks are sparser than with  $\alpha = 0.99$  but less sparse if  $\alpha = 0.9$ . In Figure 5.6, we observe that for the reaction equation, we can prune less and achieve better performance which can be explained by the fact that PDEs in the reaction family are quite similar. Therefore, we can allow the network to have more overlaps to share knowledge between subnetworks. For the case of learning within the same family of convection PDEs, the value of  $\alpha = 0.95$  was revealed to be a better option for constructing a sufficiently

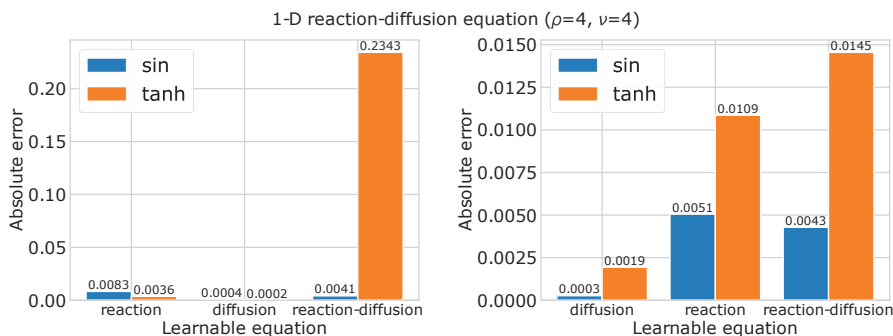


Figure 5.5: Influence of activation function on the results when the reaction learned first (left) and diffusion learned first (right).

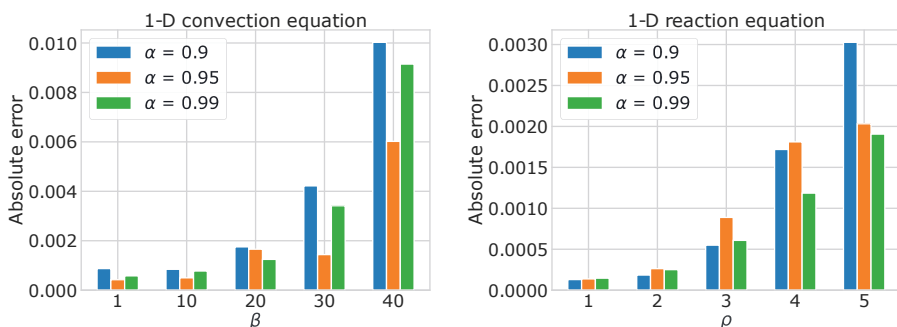


Figure 5.6: iPINNs with different values of pruning parameter  $\alpha$ .

expressive task-specific subnetwork and frees space for future tasks. Notwithstanding, the performance is good with any reasonable choice of pruning parameter.

### 5.6.2 SUBNETWORKS ANALYSIS

In Figure 5.7, we present the portions of the subnetworks that are occupied by each task. We will illustrate this by considering both orders – when the model learns the reaction equation first (Figure 5.7a), and when diffusion comes first (Figure 5.7b). These results are averaged over 3 different runs for each of the orderings. It is noteworthy that the percentage of parameters occupied by all tasks is very similar for both orderings (31.8% and 31.5% respectively of all network parameters). On the other hand, the percentages of used parameters for both cases are 79.5% and 79.3%. This means that the total number of trained parameters for the two incremental procedures is the same for both cases, which shows the robustness of the method. Moreover, the network has about 20% of free connections to learn new tasks.

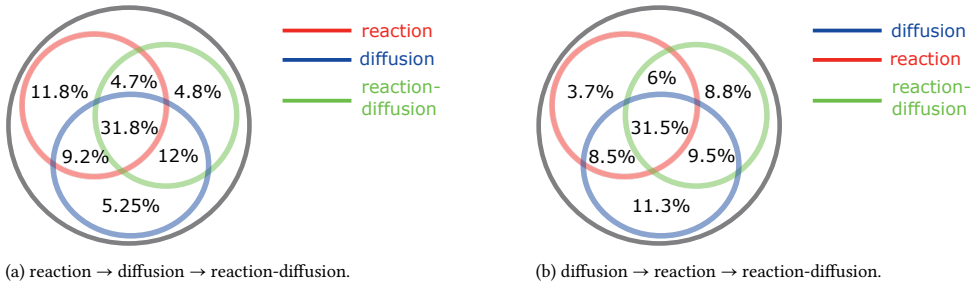


Figure 5.7: Percentage of parameters used for every equation with  $\rho = 4$ ,  $\nu = 4$ .

## 5.7 CONCLUSION

This chapter proposes an incremental learning approach for physics-informed neural networks where every task is presented as a new PDE. The algorithm is based on task-related subnetworks for every task obtained by iterative pruning. The parameters in these subnetworks are updated with respect to the assigned task. To illustrate our idea, we consider two cases when incremental learning applies to a sequence of PDEs. In the first case, we consider the family of convection/reaction PDEs, learning them sequentially. In the second example, we consider the reaction-diffusion equation and learn firstly the components of the process, namely reaction and diffusion, and only then the reaction-diffusion equation.

The purpose of this chapter is twofold. First, we show the possibility of incremental learning for PINNs without significantly forgetting previous tasks. From our numerical experiments, the proposed algorithm can learn all the given tasks, which is not possible with standard PINNs. Second, we also illustrate that future tasks are learned better because they can share connections trained from previous tasks, leading to significantly better performance than if these tasks were learned independently. We demonstrate that this stems from the transfer of knowledge occurring between subnetworks that are associated with each task. Interestingly, the model's performance on previous tasks is also improved by learning the following tasks. In essence, iPINNs demonstrate symbiotic training effects between past and future tasks by learning them with a single network composed of dedicated subnetworks for each task that share relevant neuronal connections.

## 6


# COOPERTATIVE MODELING VIA CONTINUAL LEARNING OF DIFFERENT MATERIAL BEHAVIOR

*The previous chapter illustrated the benefits of the parameters sharing between subnetworks. We have shown better error generalization in the case of physic-informed neural networks by creating PDE-related subnetworks.*

*In this chapter, we present another example where training a model sequentially with task-specific subnetworks may facilitate the learning process. We show that the chosen continual learning strategy can sequentially learn several constitutive laws without forgetting them, using less data to achieve the same error as standard (non-cooperative) training of one law per model. This research direction is also motivated by FAIR research principles to make the field more transparent and open.*

6

---

This chapter is based on  Aleksandr Dekhovich, O. Taylan Turan, Jiaxiang Yi, and Miguel A. Bessa. Cooperative data-driven modeling. *Computer Methods in Applied Mechanics and Engineering*, 417:116432, 2023. <https://doi.org/10.1016/j.cma.2023.116432>, and it has been partly presented at *Workshop on Machine Learning for Materials, ICLR 2023*.

Code repository: <https://github.com/bessagroup/CDDM>.



## 6.1 INTRODUCTION

Machine learning permeated almost every scientific discipline [208–210], and Solid Mechanics is no exception [211–213]. With all their merits and flaws [214], these algorithms provide a means to understand large datasets, finding patterns and modeling behavior where analytical solutions are challenging to obtain or not accurate enough. This work introduces the concept of cooperative data-driven modeling by highlighting the importance of continual or lifelong learning and exemplifying it in Solid Mechanics. Without loss of generality, the examples provided in this article pertain to using neural networks to create constitutive models from synthetic data [211], but the proposed strategy is based on a general method introduced by the authors in the Computer Science community [148], so it is applicable to many other fields that can also benefit from cooperative data-driven modeling.

For readers unfamiliar with the field of using machine learning to learn constitutive models of materials, we provide a short review of the topic. Using neural networks to describe constitutive material behavior was first proposed decades ago by Ghaboussi et al. [215] using simple experimental data. However, advances in numerical modeling and the ability to create large synthetic datasets have led to a new era of data-driven modeling initiated in [211] that is based on fast analysis of representative volume elements of materials. Since then, there has been rapid progress in the field, first by considering similar architectures [216–219], then by considering deep learning strategies to characterize more complex behavior [220–225], and recently including physical constraints [226, 227]. In particular, the first work to propose the use of recurrent neural networks for plasticity modeling [220] showed that these architectures could learn the path- and time-dependency behavior of materials. Soon after, several research groups proposed new neural network architectures and solved increasingly complex plasticity problems [223, 228–230]. A similar trend is ongoing in other fields within and outside Mechanics [231–234].

Simultaneously, the scientific community is experiencing strong incentives to adhere to open science, with vehement support from funding agencies throughout the World to share data and models according to FAIR principles (Findable, Accessible, Interoperable and Reusable) [235–237]. There is also a clear need for end users to reuse these models and data. Nevertheless, there is a serious issue that obstructs the synergistic use of machine learning models by the community. Artificial neural networks, unlike biological neural networks, suffer from catastrophic forgetting [6, 10, 11]. Human beings when learning a new task, e.g. playing tennis, do not forget how to perform past tasks, e.g. swimming. Unfortunately, artificial neural networks fail at this because they are based on updating their parameters (weights and biases) for the task and data being considered, but this changes the previous configuration obtained for a past task (that led to different values of weights and biases). This catastrophic forgetting has important implications in practice, as illustrated by the following scenario.

Imagine that Team A of scientists collects computational or experimental data about the constitutive behavior of Material A, and then trains an artificial neural network to predict the behavior of that material. In the end, Team A publishes the artificial neural network model and corresponding data according to FAIR principles. Later, if Team B aims to create a model that predicts the behavior of Material B then it faces two options: 1) collect data and train a model from scratch for this new material; or 2) use the model developed by

Team A in an attempt to get a better model for Material B and use less data during the training process. If the material behavior for B has some commonality with the one for A, there is an advantage in leveraging the work from Team A. However, the state of the art in the literature is to use transfer learning or meta-learning methods [195, 238–240] to adapt Model A and retrain it for Team B’s scenario [241, 242]. Unfortunately, in this case, the new model obtained by Team B is no longer valid for Team A’s scenario. Although Team B may create a model that is valid for its purposes, this would not be a truly cooperative effort with Team A because a general model valid for both scenarios would not be obtained. This represents a significant challenge to cooperative data-driven modeling because it discourages different groups from working towards a common model, ultimately leading to many independent models. Note that the problem gets worse as more tasks accumulate (more materials and more teams).

We believe that continual learning addresses this limitation and will unlock a new era of cooperative data-driven modeling traversing all fields of application. The CP&S method proposed in Chapter 3 is applied to various standard computer science datasets to demonstrate the best performance to date in the challenging class-incremental learning scenario when compared with state-of-the-art methods. Further developments are needed, but this algorithm represents an essential step towards democratizing cooperative data-driven modeling. Here, the continual learning method is applied to a new architecture suitable for plasticity modeling, demonstrating its benefit and motivating future research in this nascent field.

## 6.2 RELATED WORK

Here the focus is to be the first to apply continual learning strategies to mechanics problems, in particular for plasticity modeling. These cases are expected to involve known task-IDs, as illustrated by the previously invoked example of Team A and B that were aiming to model the behavior of two different materials because each team already knows *a priori* the material of interest to them (known task-ID). Therefore, this article only needs to consider the simpler formulation of our CP&S method presented in Chapter 3.

Unlike the standard computer vision problems for which CP&S was originally implemented and compared [148], here we apply CP&S to model irreversible material behavior. Without loss of generality, we focus on plasticity simulations but other history- and time-dependent phenomena such as damage or visco-elasticity could be considered. In fact, CP&S can be applied to other classification or regression tasks (they do not need to be tasks in Mechanics). Given the time and/or history-dependency of these problems, considering neural network architectures with recurrent units facilitates the learning process.

Recurrent neural networks (RNNs) [243, 244] typically deal with sequential data, e.g. in natural language processing problems [245] or voice recognition [246]. However, they suffer from vanishing and exploding gradients issues [247, 248]. Further improvements of RNNs, such as the Long Short-Term Memory (LSTM) network [249] and Gated Recurrent Unit (GRU) [250] solved this problem, enabling their application in different contexts. In particular, LSTMs and GRUs have been shown to learn history-dependent phenomena in mechanics [220, 251]. In this work, we use GRUs as described in our past work [220] due to their simplicity and effectiveness when compared to LSTM, although other neural network architectures with recurrent units could be considered. Computations that occur in the

GRU can be described as follows:

$$z_t = \text{sigmoid}(W_z x_t + U_z h_{t-1} + b_z), \quad (6.1)$$

$$r_t = \text{sigmoid}(W_r x_t + U_r h_{t-1} + b_r), \quad (6.2)$$

$$\hat{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \quad (6.3)$$

$$h_t = z_t \odot \hat{h}_t + (1 - z_t) \odot h_{t-1}, \quad (6.4)$$

where  $h_0 = 0$ ,  $x_t$  is an input vector,  $t = 0, 1, \dots, T$ . Matrices  $W_z, W_r, W_h, U_z, U_r, U_h$  are learnable weights and  $b_z, b_r, b_h$  are learnable biases. In this work, the PyTorch [252] implementation of GRUs is used to conduct the experiments consisting of simplified representative volume elements of materials with different microstructures, as described in Section 6.4.

### 6.3 PROPOSED METHOD

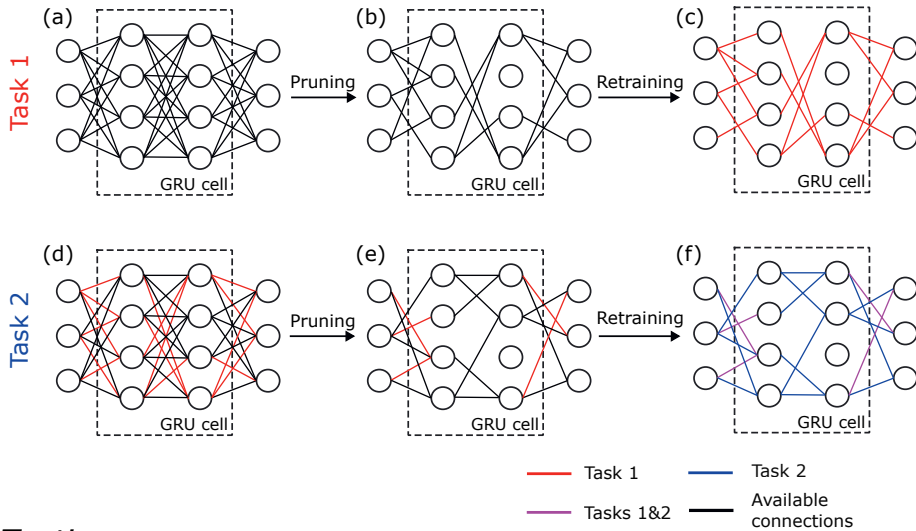
Independently of the architecture chosen for the neural network, we advocate here that architectural continual learning algorithms such as CP&S [148] (see Chapter 3) enable Cooperative Data-Driven Modeling (CDDM). The method creates different subnetworks within an artificial neural network that are associated to particular tasks without forgetting past tasks. For example, in our context each subnetwork is associated to a specific material model for a class of materials with a given microstructure, constituent properties and external conditions (see e.g. [211]). Architectural methods, however, are limited by the availability of free neural connections that can be trained for a new task. A particular advantage of CP&S is that it is based on the NNrelief (Chapter 2) iterative pruning strategy [22] which was developed aiming to create sparser subnetworks (using fewer connections) than other pruning methods such as magnitude pruning [29] or neural pruning [30]. Additionally, although CP&S was implemented in the original article for convolutional and fully connected networks, it can easily be adapted to GRU networks as shown in this work.

CP&S is based on a set of simple steps to create a group of overlapping subnetworks, each of them learning a particular task without disrupting the knowledge accumulated by the other subnetworks. Importantly, the subnetworks can (and usually do) share knowledge among them by sharing connections that are useful to each other. This mechanism allows to learn different tasks, and transfer knowledge between them but avoids forgetting, unlike transfer learning methods.

Overall, cooperative data-driven modeling via the proposed CP&S method is described as follows:

- Training:
  1. At the beginning of the learning process, initialize the entire neural network with random weights (see Figure 6.1a).
  2. Set the hyperparameters (architecture, optimizer, learning rate, weight decay, pruning parameters, etc.).
  3. For a task  $T_i$  (e.g. learning a new material), create a subnetwork  $\mathcal{N}_i$  that is associated to that task:

## Training



## Testing

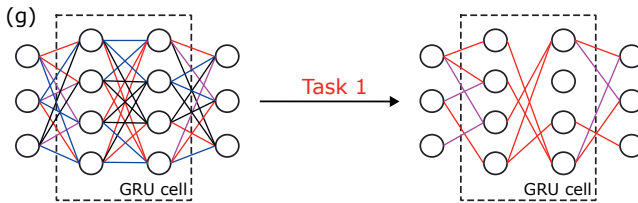


Figure 6.1: Overview of the proposed CDDM approach.

(a) Use the NNrelief [22] algorithm to prune connections from the entire neural network (see Figure 6.1b, details in Chapter 2).

- **Remark 6.1** Every connection can be pruned, whether they are already part of another subnetwork or not.
- **Remark 6.2** Pruning is controlled by hyperparameter  $\alpha \in (0, 1)$  that represents the amount of information that is maintained coming out of the neurons after removing the connections. So,  $\alpha = 0.95$  indicates that 95% of the signal coming out of the neurons is kept on average for the training set of task  $T_i$  after removing the connections. The amount of information is measured according to a metric called importance score – see the original article [22] or Chapter 2 for details.
- **Remark 6.3** Allowing to prune any connection of the entire network (whether belonging to a previous subnetwork or not) is crucial because it provides a mechanism to keep connections that are useful for performing the current task but remove the ones that are not. If some connections are kept from other subnetworks, then there is potential for knowledge transfer.

(b) Among the remaining connections, i.e. the ones that are not pruned, retrain the “free” connections but do not update the ones that are part of other subnetworks (see Figure 6.1c).

– **Remark 6.4** *By refraining from updating connections that are part of other subnetworks, the CP&S algorithm avoids forgetting past tasks because the other subnetworks are not affected by the current training of this subnetwork. Instead, only connections that have not been used previously by any subnetwork are updated so that new knowledge can be accumulated, as needed to solve a new task.*

(c) Assess performance of this subnetwork ( $\mathcal{N}_i$ ) on the dataset for this task ( $T_i$ ). If error is not acceptable, go to 3a and iterate again until a maximum number of iterations  $n$  is met. Else save the connections that form this new subnetwork into mask  $\mathbf{M}_i$ . The weights belonging to this subnetwork will no longer be updated when training another subnetwork later.

4. Consider the next task  $T_{i+1}$  by finding a new subnetwork ( $\mathcal{N}_{i+1}$ ) as described in step 3 (see Figure 6.1d-f)).

• Inference (testing):

1. For the given test point  $x$  from task  $T_i$ , select subnetwork  $\mathcal{N}_i$  (see Figure 6.1g).

– **Remark 6.5** *In this chapter, task selection from the data is not necessary, because practitioners already know what is the material model that they want to consider. Otherwise, see the original article for the task incremental learning scenario [148].*

2. Make a prediction  $y = \mathcal{N}_i(x)$ .

We highlight from the above description that the performance of CP&S is controlled only by two hyperparameters: the number of pruning iterations  $n$  and the pruning parameter  $\alpha \in (0, 1)$ . A low value of  $\alpha$  will cause more connections to be pruned (information compression) but also lower expressivity (worst performance of the subnetwork after pruning and retraining for  $n$  iterations). The subnetwork is represented by a binary mask  $\mathbf{M}$ , where every active connection is represented by 1 and where every inactive connection is represented by 0. The weights and biases that are first assigned to a particular task are not updated for subsequent tasks – ensuring that the original subnetwork for which these connections were trained remains unchanged by the training process of a new subnetwork. This strategy was proven effective in the context of computer vision, and it should remain valid for computational mechanics applications.

An important disadvantage of CP&S is that fixing the parameters associated with a trained subnetwork can quickly exhaust the number of “free” connections available to be trained in subsequent tasks. In other words, the artificial neural network can be “saturated” after several tasks have been learned. This is reported in Chapter 3 and the original investigation [148]. However, we note that CP&S does not have limitations on the type of neural network architecture to be considered. Therefore, the choice of architecture depends only on the solvable problem. In this paper, we consider the case when all tasks have the same input and output dimensions. This condition can be relaxed if one uses

separate task-specific input and output heads. In this case, shared parameters are the ones that are in the intermediate layers.

## 6.4 FIRST CASE STUDY: LEARNING PLASTICITY LAWS FOR DIFFERENT MICROSTRUCTURES

The first and simpler case study considers non-periodic material domains subjected to uniform displacements at the boundary. The goal is to create constitutive models from these material domains similarly to the original publication on this subject [211], but including history-dependent behavior [220]. We consider four different materials with different porous microstructures but the same matrix phase – a simple von Mises plasticity model. This canonical example can be extended to more complex cases, including realistic microstructures, as shown in case study 2.

We present two case studies to demonstrate the usefulness of cooperative data-driven modeling (CDDM). CDDM aims to create a model that is capable of performing multiple tasks but that also uses past knowledge from different tasks to decrease the number of training points required to learn a new task. Therefore, CDDM should achieve better performance than conventional training (without cooperation) for the same number of training points. In addition, the fact that CDDM uses the same neural network should make it more efficient in terms of the number of new parameters needed to learn a new task.

### 6.4.1 PROBLEM DESCRIPTION

We consider an elastoplastic von Mises material with hardening to create a path-dependent problem. After applying the boundary conditions, the average Cauchy strain ( $\bar{\epsilon}_{1:t}$ ) and Cauchy stress ( $\bar{\sigma}_{1:t}$ ) measures are obtained for the deformation path. Then the learning problem for a single task can be defined as,

$$\bar{\sigma}_{1:t} = f(\bar{\epsilon}_{1:t}), \quad (6.5)$$

where a machine learning model is utilized to find the relationship  $f : \bar{\epsilon}_{1:t} \mapsto \bar{\sigma}_{1:t}$ , which is a supervised regression problem with the history of the average strain components as an input and the average stress components as an output. Since we consider the 2D problems, every pair of input-output datapoint is  $(\bar{\epsilon}_{1:t}, \bar{\sigma}_{1:t})$ , where  $\bar{\epsilon}_{1:t} = (\epsilon_{11}, \epsilon_{12}, \epsilon_{22})$ ,  $\bar{\sigma}_{1:t} = (\sigma_{11}, \sigma_{12}, \sigma_{22})$  and all  $\epsilon_{ij}, \sigma_{ij} \in \mathbb{R}^t$ .

The tasks originating from different domains can be seen in Figure 6.2. Each domain represents a square plate with the different number of holes. More details about the data simulation process can be found in Appendix D.1. We reinforce that Case Study 1 considers simpler finite element analysis because we want to facilitate the dissemination of the methodology and use fully open-source software – facilitating replication of the example and easy adaptation to other scenarios.

The four material domains are subjected to the same 1000 paths of deformation. These 1000 paths are obtained from 100 end displacement values of the top boundary of the domain that are sampled from a Gaussian Process posterior that is conditioned on 20 displacement values sampled from a uniform distribution for each path. Then the average stress obtained for each path is calculated via the finite element method – see for example Figure 6.3. The

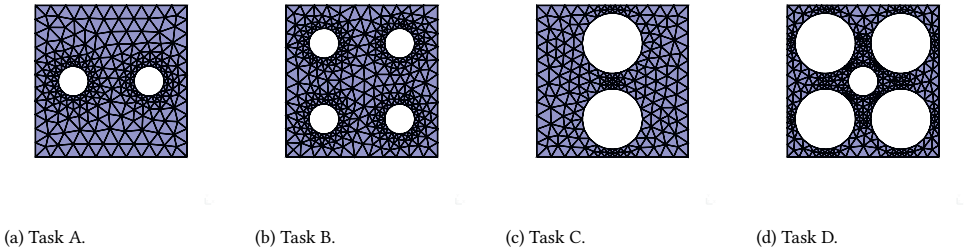


Figure 6.2: Different domains introduced as different tasks.

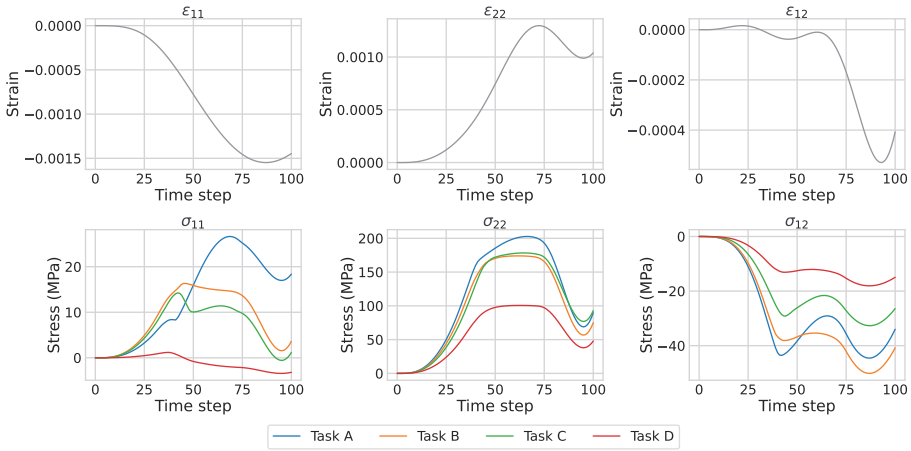


Figure 6.3: An example of strain and stress paths.

dataset is made available, and the simulations used to obtain it can be replicated via the source code. We highlight that different strategies for sampling the paths of plasticity material laws have been proposed [253] and that this choice can affect the number of paths needed to train the neural network up to the desired accuracy. Nevertheless, each presented task was subjected to the same deformation paths to calculate the domain-specific average stress and strain values. The data is generated using FEniCS [254].

In order to illustrate the difference in stresses obtained for the four tasks, Figure 6.4 shows the error according to Eq. 6.6 and the mean-squared-error (MSE) between stresses for the training data for all tasks. We scale all the data since the model receives it in a scaled format. Here we use standard scaling where we remove the mean from every input feature and divide it by the standard deviation of the training set. The figure clarifies that Tasks A, B and C are more similar with each other than Task D.

To evaluate the performance of CDDM we measure the error  $E_i$  on every test path as follows:

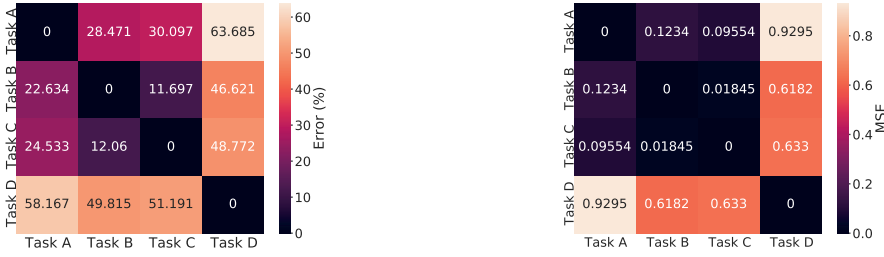


Figure 6.4: The difference between the stress train data for considered tasks, calculated using the error measurement (left) and the MSE measurement (right).

Table 6.1: Training hyperparameters.

training epochs	learning rate	weight decay	pruning iterations	pruning parameter $\alpha$	retraining epochs
1000	0.01	$10^{-6}$	1	0.95	200

$$E_i = \frac{1}{3} \left( \frac{\|\sigma_{11}^i - \hat{\sigma}_{11}^i\|_2}{\|\sigma_{11}^i\|_2} + \frac{\|\sigma_{22}^i - \hat{\sigma}_{22}^i\|_2}{\|\sigma_{22}^i\|_2} + \frac{\|\sigma_{12}^i - \hat{\sigma}_{12}^i\|_2}{\|\sigma_{12}^i\|_2} \right) \cdot 100\%, \quad i = 1, 2, \dots, N, \quad (6.6)$$

where  $N$  is the number of testing points,  $\|\cdot\|_2$  is the L2 norm,  $\hat{\sigma}_{11}^i, \hat{\sigma}_{22}^i, \hat{\sigma}_{12}^i \in \mathbb{R}^t$  are predicted stress components, and  $\sigma_{11}^i, \sigma_{22}^i, \sigma_{12}^i \in \mathbb{R}^t$  are the test ones.

Then, we compute the average over all  $N$  test points to compute the final test error:

$$Err = \frac{1}{N} \sum_{i=1}^N E_i. \quad (6.7)$$

The hyperparameters that we use to train a neural network are shown in Table 6.1. We use Adam [66] optimizer to train the model with the mean-squared-error (MSE) loss function. To prevent overfitting, we add weight decay regularization [255, 256] to the networks' parameters.

As it is common in continual learning literature [257], we test the approach with different task orderings. Overall, we consider four orderings for the case of four tasks in a sequence:

- ordering 1: Task A  $\rightarrow$  Task B  $\rightarrow$  Task C  $\rightarrow$  Task D;
- ordering 2: Task B  $\rightarrow$  Task D  $\rightarrow$  Task A  $\rightarrow$  Task C;
- ordering 3: Task C  $\rightarrow$  Task A  $\rightarrow$  Task D  $\rightarrow$  Task B;
- ordering 4: Task D  $\rightarrow$  Task C  $\rightarrow$  Task B  $\rightarrow$  Task A.



## 6.4.2 RESULTS

Firstly, we train the GRU with 2 cells and a hidden size of 128 in the sequence of four tasks. We train the first task with 800 training paths, and for each of the following tasks, we consider the cases of 800, 400, 200 and 100 training paths. We compare these results with the conventional case (non-cooperative) where every new task is trained with the same GRU but independently of the other tasks. Our main goal is to show the benefits of cooperative learning compared to non-cooperative under a low-data regime.

Figure 6.5 shows this comparison, where the blue bars refer to the cooperative model (CDDM) and the orange ones to the conventional case (standard or non-cooperative training). The test error is computed using Eq. 6.6. It is clear that CDDM significantly outperforms standard training when we decrease the number of training points. This effect is consistent across all four orders, independently of which task is considered to be the first. The main advantage of CDDM is that the pretrained parameters have an accumulative effect on future tasks. This multi-transfer effect has more significance under the low-data regime (e.g., 100 training paths). Also, the set of parameters depends on the order in which the tasks are learned. In Table 6.2, we present the average error for every task when considering a different number of training points (number of paths); note that this error is the average over the four task orderings. We can clearly see that CDDM performs better than standard training for tasks 2 – 4 on average.

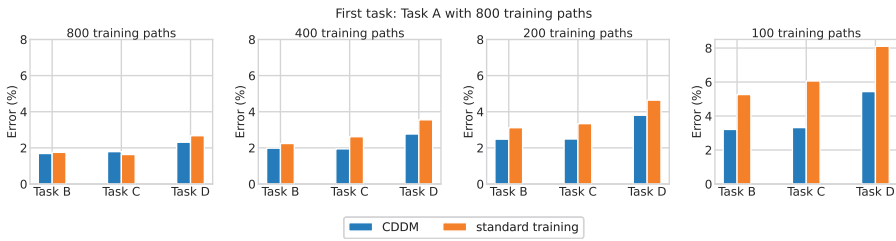
Table 6.2: Test error (%) averaged over four considered task orders.

	task 1	tasks 2–4			
	800 paths	800 paths	400 paths	200 paths	100 paths
standard training	<b>2.02</b>	2.02	2.82	3.75	6.43
CDDM	2.14	<b>1.92</b>	<b>2.18</b>	<b>2.84</b>	<b>3.97</b>

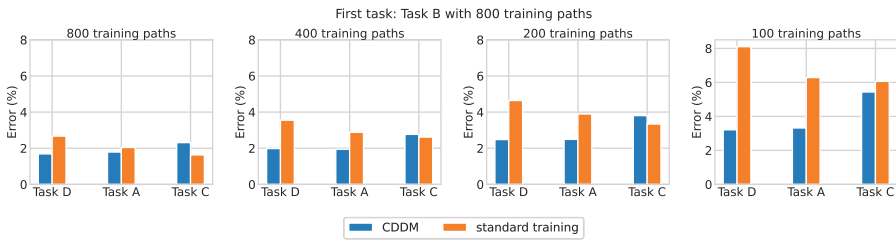
Moreover, we learn all four tasks with one network, while four separate networks are necessary for the conventional case. Therefore, using fewer parameters and achieving better performance. As clarified next, this is explained by the knowledge transfer that happens between subnetworks. Also, it should be noted that the network still has free space to learn future tasks, although saturation would occur soon if more tasks were considered because the neural network is small.

In Figure 6.6, we show the prediction of CDDM with 200 training paths for tasks 2-4 respectively; the first task is learned with 800 paths. We compare CDDM with standard (non-cooperative) training with one network per task. As the figure illustrates, CDDM learns the data better than conventional training and requires just one network instead of four. Hence, Figure 6.6 justifies our hypothesis of using continual learning as a possible solution to tackle the data scarcity problem in history-dependent constitutive law modeling. From this figure, it is clear that the GRU is able to learn material behavior by having 200 training paths for tasks 2-4.

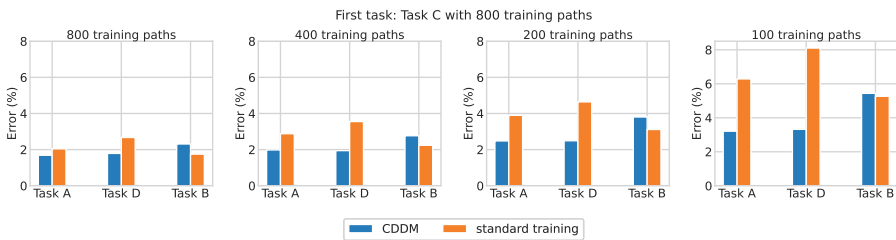
Overall, we observe that the continual learning strategy allows us not only to learn four different geometries with one network but also improves test error under the limited data regime. In addition, we can see that even if we have enough data, continual learning



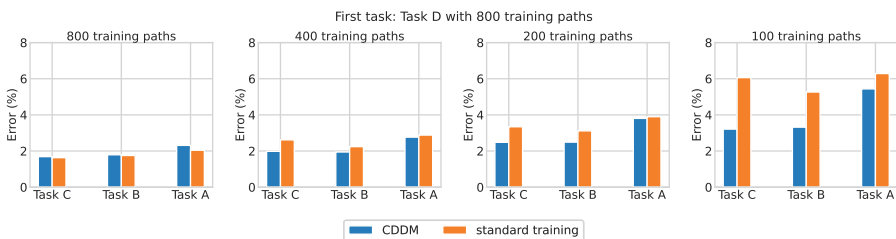
(a) Ordering 1.



(b) Ordering 2.



(c) Ordering 3.



(d) Ordering 4.

Figure 6.5: First case study: CDDM results on orderings 1-4.

does not worsen the results significantly compared to standard training (no more than 0.5% difference).

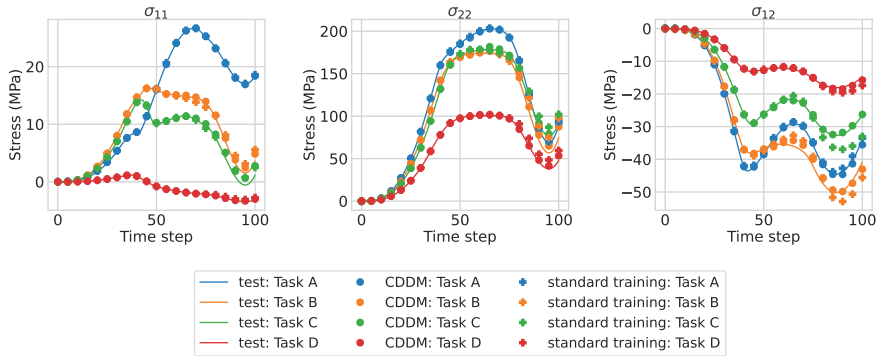


Figure 6.6: First case study: 800 training paths for task 1 and 200 training paths for tasks 2-4.

## 6.5 SECOND CASE STUDY: RVEs WITH PERIODIC BOUNDARY CONDITIONS

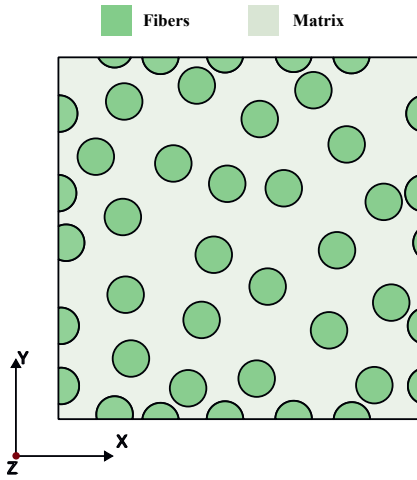
In this second case study, we apply the CDDM strategy to more realistic representative volume elements (RVEs) subjected to periodic boundary conditions, as thoroughly discussed in a past work [211]. The data is generated with a commercial finite element solver, but both the code to generate the data as well as the dataset itself are made available with RVE simulator<sup>1</sup> [258]. The RVE average stress–strain response is influenced by the microstructure, properties of each material phase, and loading conditions (average strain path that is converted into a periodic displacement at the boundary). Four different RVEs are considered in order to create four tasks. Each task pertains to learning the homogenized plasticity constitutive behavior (average stress–strain response) of a corresponding RVE. These RVEs were created such that they share some similarities and significant differences by considering different microstructures and material properties while applying the same average deformation paths to generate the dataset.

### 6.5.1 PROBLEM DESCRIPTION

Figure 6.7 illustrates the type of two-dimensional composite material used to create the 4 RVEs (tasks). The two-phase RVEs are defined by 4 geometric descriptors: (1) RVE size, (2) fiber<sup>2</sup> volume fraction ( $v_f$ ), (3) fiber mean radius ( $r$ ), and (4) fiber radius standard deviation ( $r_{\text{std}}$ ). The last two descriptors are used to create circular fibers whose radius is drawn from a Gaussian distribution with the corresponding mean and standard deviation. The descriptors of the material properties are simply the elastic properties of the fibers and matrix (Young’s modulus ( $E$ ) and Poisson’s ratio ( $\nu$ )), and the plasticity properties of the matrix (isotropic hardening law that depends on the yield stress that completely defines the von Mises yield surface). These descriptors are all defined on the right part of Figure 6.7, where the red font indicates the descriptors that are changed among the 4 RVEs and where the ones in black font indicate parameters that are fixed in this investigation, without loss

<sup>1</sup>The code for the RVE simulator is available at <https://github.com/bessagroup/rvesimulator>.

<sup>2</sup>We consider plane strain conditions, so we tend to view the reinforcement phase as fibers instead of particles.



### Microstructure descriptors

- RVE length:  $L$
- Volume fraction:  $v_f$
- Fiber mean radius:  $r$
- Fiber radius std:  $r_{std}$

### Material properties descriptors

*Matrix plasticity constitutive law*

- Young's modulus:  $E_{matrix}$
- Poisson's ratio:  $\nu_{matrix}$
- Hardening law:  $\sigma_y = f(\sigma_{y_0}, \bar{\epsilon})$

*Fiber elasticity constitutive law:*

- Young's modulus:  $E_{fiber}$
- Poisson's ratio:  $\nu_{fiber}$

Figure 6.7: Illustration of commonalities and discrepancies on setting up tasks

of generality. The 4 RVEs (i.e. Tasks) are labeled A, B, C and D and the corresponding values for the descriptors are included in Table 6.3. For clarity, the 3 microstructures defining the RVEs (note that two RVEs have the same microstructure) are shown in Figure 6.8.

Table 6.3: Parameters configuration of different tasks (Units:SI(mm))

Task	Microstructure parameters			Hardening law	$E_{fiber}$	Fixed parameters			
	$v_f$	$r$	$r_{std}$			size	$E_{matrix}$	$\nu_{matrix}$	$\nu_{fiber}$
A	0.45	0.01	0.003	$\sigma_y = 0.5 + 0.5\bar{\epsilon}$	10	0.048	100	0.30	0.19
B	0.30	0.003	0.0	$\sigma_y = 0.5 + 0.5(\bar{\epsilon})^{0.4}$	1				
C	0.15	0.0015	0.0003	$\sigma_y = 0.5(1 + \bar{\epsilon})^{\frac{1}{0.4}}$	1000				
D	0.30	0.003	0.0	$\sigma_y = 3.0 + 0.5(\bar{\epsilon})^{0.4}$	1				

As in Case Study 1, the target is to learn the average Cauchy stress ( $\bar{\sigma}_{1:t}$ ) which is dependent on the applied average strain path ( $\bar{\epsilon}_{1:t}$ ). Therefore, the learning problem can be defined as:  $\bar{\sigma}_{1:t} = f(\bar{\epsilon}_{1:t})$ , where  $t$  is the pseudo-time step (load step) in the simulation defined to be 100. Meanwhile, 1000 different strain paths are generated according to a simple interpolation method. Specifically, for each average strain component ( $\bar{\epsilon}_{11}$ ,  $\bar{\epsilon}_{22}$ , and  $\bar{\epsilon}_{12}$ ) of a path, 8 equally spaced points are sampled within the strain path, and the quadratic interpolation method is adopted to generate the full strain path. Then, the strain path is converted into a boundary value problem of the RVE and the finite element prediction is conducted with the commercial software ABAQUS [259] to simulate the corresponding average stress for different tasks. In Figure 6.9, we present the difference between data computed with error metric (Eq. 6.6) and the mean squared error (MSE).

## 6.5.2 RESULTS

First, we train GRU on tasks A, B and C with the same hyperparameters as in Section 6.4. We consider three different tasks orders:

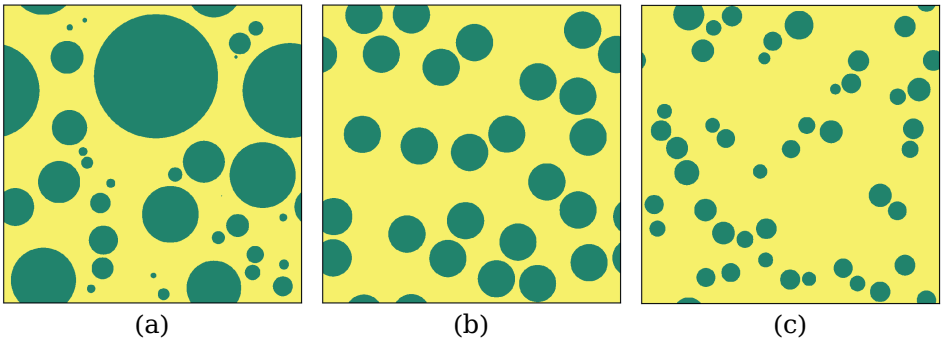


Figure 6.8: Schematics of different microstructure configurations: (a)  $v_f = 0.45$ ,  $r = 0.01$ ,  $r_{std} = 0.003$ ; (2)  $v_f = 0.30$ ,  $r = 0.003$ ,  $r_{std} = 0.0$ ; (c)  $v_f = 0.15$ ,  $r = 0.0015$ ,  $r_{std} = 0.0003$ .

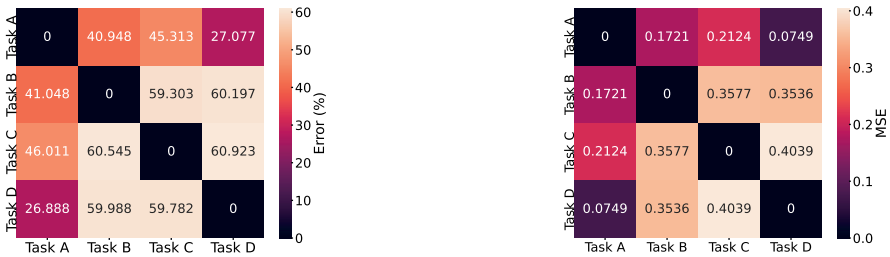


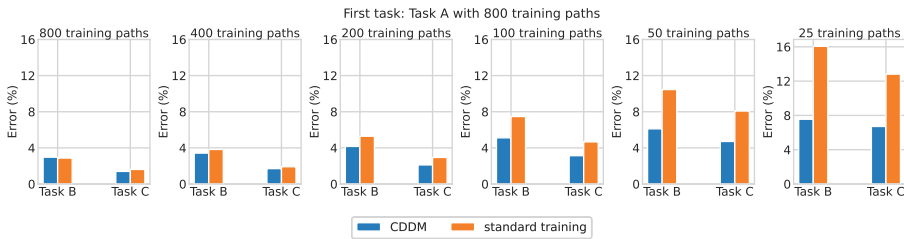
Figure 6.9: The difference between the stress train data for considered RVE tasks, calculated using the error measurement (left) and the MSE measurement (right)

- ordering 1: Task A  $\rightarrow$  Task B  $\rightarrow$  Task C;
- ordering 2: Task C  $\rightarrow$  Task A  $\rightarrow$  Task B;
- ordering 3: Task B  $\rightarrow$  Task C  $\rightarrow$  Task A.

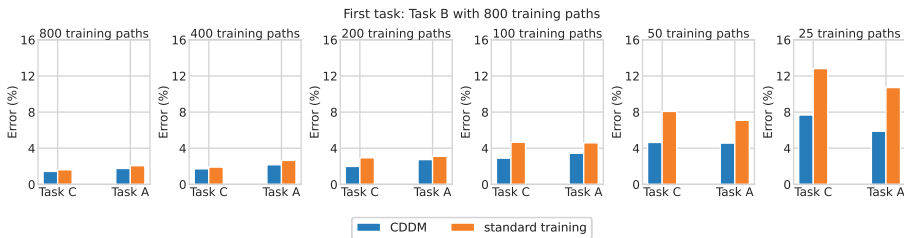
In Figure 6.10, we show the results for these three task orderings. It is clear that with the decrease in the number of training paths, CDDM starts to outperform conventional training.

In Figure 6.11, we present the prediction of one stress path with CDDM and standard training. The first task (task B) is trained with 800 training paths, while the next two tasks (tasks C and A) are trained using 200 training paths (**left**) and 25 training paths (**right**). If tasks C and A are learned with 200 paths, both CDDM and standard training predict stress well, however, CDDM does this with a single network. We observe that if GRU learns tasks in the cooperative approach, the prediction is more accurate than with conventional training if 25 training paths are given for the second and third tasks.

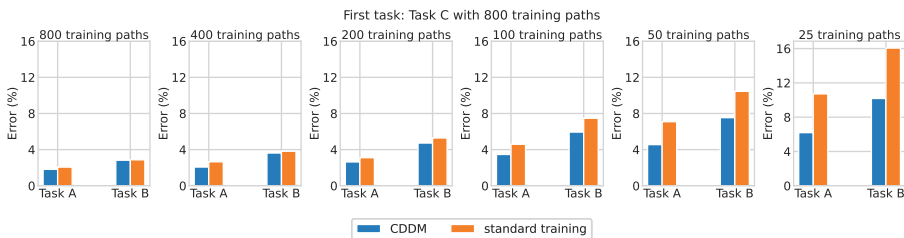
However, in an attempt to explore and report on the limitations of the presented method, we also investigated what occurs when we add task D (see Figure 6.12) where an RVE has much larger yield stress (see Table 6.3, i.e. where the yield stress of the matrix becomes



(a) Ordering 1.



(b) Ordering 2.



(c) Ordering 3.

Figure 6.10: Second case study: CDDM results on orderings 1-3.

3.0 MPa instead of 0.5 MPa as in the other tasks. In this case, the plastic response of the RVE is delayed, and we noticed that when Task D is learned first then there would be no advantage in learning cooperatively – see Figure 6.12b. However, if the ordering is different, as shown in Figure 6.12a, then the proposed cooperative model is still better. We think it is important to be clear that there might be situations in which the ordering of tasks actually leads to difficulties in learning cooperatively.

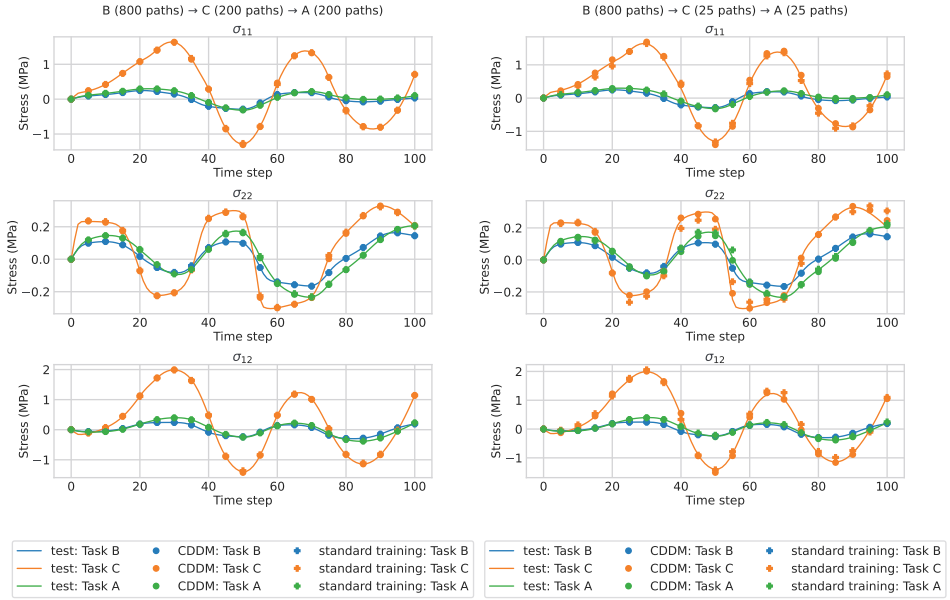
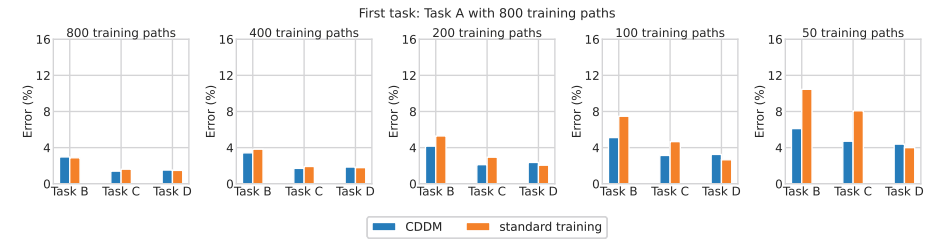
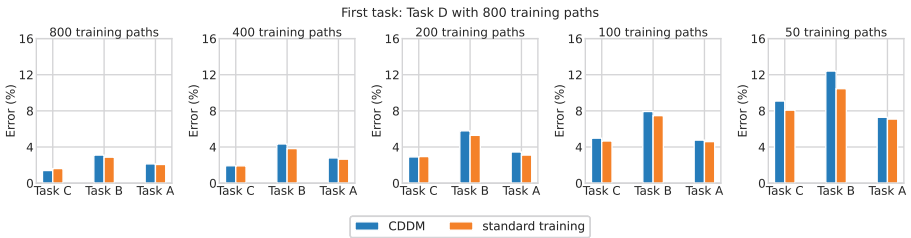


Figure 6.11: Second case study: Comparison of the CDDM and standard training predictions with different numbers of training paths.

6



(a) Task A  $\rightarrow$  Task B  $\rightarrow$  Task C  $\rightarrow$  Task D.



(b) Task D  $\rightarrow$  Task C  $\rightarrow$  Task B  $\rightarrow$  Task A.

Figure 6.12: Second case study: CDDM results on the sequences of four tasks.

## 6.6 DISCUSSION

We want to highlight that all the above-mentioned results for both case studies are robust to the hyperparameter choice. For simplicity of presenting the previous results, they refer to a particular architecture configuration. However, in this section we elaborate on the robustness of the CDDM approach presented herein by considering different GRU architectures, varying the number of units and hidden state size. In addition, we elaborate on the knowledge transfer effect and analyze the portion of parameters shared between tasks and the ones dedicated only to one task.

### 6.6.1 ARCHITECTURES COMPARISON

In this section, we explore how CDDM depends on different GRU architecture hyperparameters such as the number of units or hidden states size. To begin, we consider the first learning case study (Section 6.4) with the corresponding four tasks: the first task is trained with 800 training paths and for the second task we vary the number of training points from 800 to 50. For GRU, we change the number of units from 1 to 3 and consider the hidden state size equal to 64, 128, and 256. Corresponding numbers of learnable parameters are shown in Table 6.4. In Figure 6.13, we compare these three network configurations. Overall, we observe similar performance for all of these architectures with insignificant differences. From the figure, we observe that all architectures give us similar results, therefore CDDM is not limited to some special network configuration.

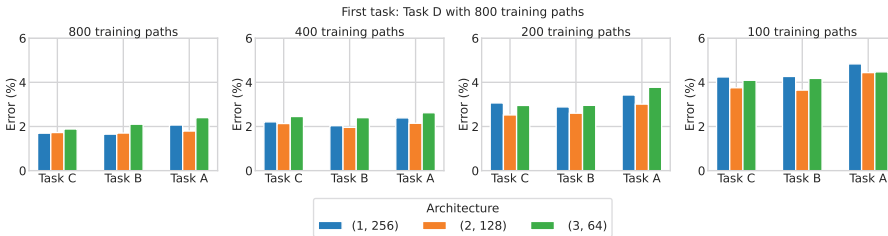


Figure 6.13: First case study: architectures comparison on ordering 4.

Table 6.4: Architectures comparison.

Architecture	(1, 256)	(2, 128)	(3, 64)
The number of parameters	201K	151K	63K

We also want to note that smaller networks (fewer parameters) do not predict the material behavior better when considering the conventional training scenario (non-cooperative). To illustrate this, consider the GRU with 1 cell and the hidden state size of 64 which results in 13K parameters. We train this network on the second case study data and consider average error on tasks A – D using 800, 400, 200, 100, 50 and 25 training paths. The test errors for these cases are 2.09%, 2.74%, 3.59%, 5.2%, 7.73%, 12.9%. On the other hand, we consider the model with 2 cells and the hidden size of 128 (151K parameters, see Table 4),



for which the test errors are 2.01%, 2.54%, 3.35%, 4.84%, 7.40%, 11.34%. As can be seen, the larger model has lower test error.

## 6.6.2 KNOWLEDGE TRANSFER

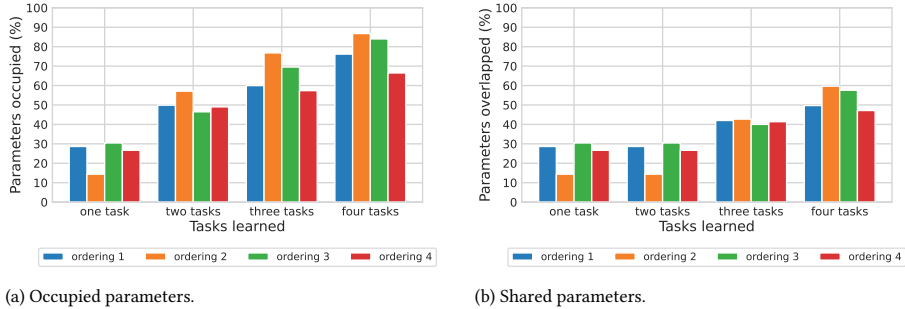


Figure 6.14: First case study: subnetworks analysis: occupied and shared parameters.

A crucial characteristic necessary to establishing the CDDM paradigm is the robustness to different task orderings, both in terms of prediction error and the number of parameters used for every task. Focusing on the first case study, we show the relation between subnetworks in the case where the first task is trained with 800 paths and all the other tasks with 200 training paths. In Figure 6.14a, we show which percentage of the total number of parameters is occupied after a new task is learned. We compare these percentages across four orderings, and in general, we observe the consistency in the number of used parameters with insignificant differences. This means that the ordering of tasks has a negligible effect on how many parameters are occupied in the end.

At the same time, in Figure 6.14b, we demonstrate the percentage of shared connections while the model learns a new task. So, for instance, we observe in ordering 3 that up to 60% of parameters are assigned to more than one subnetwork when all the tasks are learned. Overall, at least 45% of the parameters are shared between multiple tasks without a negative impact on model performance. From the figure, it is clear that the changes in the numbers of shared parameters are consistent across all orderings, illustrating robustness to different task sequences. Nevertheless, after experimenting with many tasks and considering two different case studies, we were able to find one task ordering for Case Study 2 where the cooperative data-driven modeling process was not beneficial when compared to learning all tasks separately (recall Figure 6.12b).

## 6.7 CONCLUSION

This chapter introduces the concept of continual learning in the notion of cooperative data-driven modeling. We focus on solid mechanics applications by considering two case studies involving history-dependent plasticity problems. To the best of our knowledge, this is the first example of the application of continual learning in Mechanics and among the first in Engineering applications. We demonstrate that a recurrent neural network can

sequentially learn multiple tasks, without replaying data from previous tasks and without forgetting – an important distinction when comparing to transfer learning methods and a key enabler of cooperative modeling. The proposed method is based on creating task-related subnetworks that transfer knowledge from each other by sharing neural connections. This is demonstrated to decrease the number of training data required to learn a new task (in this case, a new material law). The approach is robust to different task orders.

As a final note, the authors share their belief that the proposed cooperative data-driven modeling concept has a lot of potential for future developments. More efficient ways of sharing knowledge or selecting subnetworks (when needed), delaying the premature saturation of the network, and accelerating training are only a few possibilities to improve the proposed strategy. Notwithstanding, the prospect of fostering collaborations across different research communities by taking advantage of a machine learning model from a group and adding new capabilities to it such that it solves a new task using less training data and without forgetting how to perform the original task is an exciting new development.



# CONCLUSIONS



# 7

## CONCLUSION AND FUTURE WORK

*This chapter summarizes the methodology and findings of the thesis. We also discuss the limitations of the proposed method and future work for improvements. In particular, we focus on network saturation and task prediction problems.*

## 7.1 CONCLUSION

In the first part of the thesis, we have introduced a general paradigm for solving continual learning problems where the network creates task-specific subnetworks at the training stage and selects the correct one during inference to make a prediction. In Chapter 2, we have presented an iterative pruning algorithm (NNrelief) that uses an input signal to evaluate the importance of each parameter and prune the least significant ones. By pruning the network, we create sparse subnetworks that perform the task as well as the original network, meaning that we created a task-specific region in the network (**RQ1**). We have demonstrated the superiority of the developed method on a range of image datasets. Moreover, we explored the impact of the two most popular optimizers – Adam and SGD, showing that Adam produces sparser subnetworks for architectures without skip-connections (LeNet and VGG). In addition, we have explained why the proposed NNrelief is a more promising pruning approach than a magnitude-based one which is a common baseline.

Chapter 3 illustrates a continual learning algorithm that incorporates the proposed pruning strategy for creating a separate subnetwork for every task. By freezing the parameters assigned for a certain task for the first time and reusing them without retraining for the following tasks, we achieved overlapping subnetworks that do not disturb each other (**RQ2**). We also examine two subnetwork selection strategies – maxoutput and importance scores-based one. However, both of these strategies have one significant limitation: they require a batch of test samples to predict the correct subnetwork. This happens due to the intra-task distribution multi-modality and the similarity between tasks. This issue makes task prediction an extremely challenging task. However, if every task can be modeled with unimodal distribution, we can train a model to predict the correct subnetwork during inference. Thus, in Chapter 4, we consider a surface defect segmentation problem, where each task consists of one type of defects. Therefore, we have trained the streaming LDA model to predict the correct subnetwork first and then predict the defect mask. As a result, the batch size issue was successfully resolved in this problem. This Chapter illustrates how the subnetwork selection challenge can be resolved under simpler problem formulation (**RQ3**).

The second part of the thesis introduces the concept of cooperative learning via continual learning. The idea behind cooperative learning is accumulating knowledge without forgetting and using it when learning new tasks. We have applied the proposed methodology to engineering problems, demonstrating the advantage of cooperative learning using the examples of physics-informed neural networks and constitutive modeling problems. In Chapter 5, we could show that by leveraging past knowledge and adapting the overlapping parts of multiple subnetworks the model can generalize the new given task better. Moreover, we showed that in the case of the reaction PDE, previous subnetworks can learn from the new ones. Secondly, in Chapter 6, we have developed a Cooperative Data-Driven Modeling (CDDM) network which has access only to the current stress-strain data but can predict this relation for every task seen if the information about the task is provided. Thus, in the second part of the thesis, we have addressed **RQ4**, showing the advantages of the proposed paradigm in terms of generalization and the need for the number of training points. It is important to note that the concept of cooperative learning works effectively when the tasks are similar enough that subnetworks can transfer knowledge to each other.

## 7.2 FUTURE WORK

Developing deep learning models that do not forget will have a huge impact on their applicability to real-life problems. In the second part of the thesis, we have considered some applications of continual learning to computational mechanics and physics and the benefits associated with it. However, some challenges still should be addressed for better performance of the proposed methodology. In this section, we discuss further directions for improving the proposed paradigm.

**Network saturation.** In Chapter 3, we have demonstrated the performance of the proposed CP&S algorithm on a variety of examples including the well-known CIFAR-100 dataset split into 20 classification tasks of 5 classes per task. Thus, 20 overlapping subnetworks were constructed by the algorithm, which led to the situation where a newly created subnetwork did not have enough free trainable connection after learning a certain task (in our case, this was task eleven). As a result, we observed a significant degradation of the network's learning capabilities. We called this phenomenon *network saturation*. In this situation, the network does not physically have available connections for new tasks, in contrast to capacity saturation [113, 260] where all parameters are still available but generalization capabilities are declining due to the fixed network architecture. One of the directions of future work should be dedicated to overcoming this issue. A possible solution lies in applying a regularization-based approach (e.g., EWC [84], SI [5] or MAS [85]) to overlapping parts instead of freezing them. This way the model gets flexibility that allows slight updates of the overlapping parameters without significant forgetting. As a result, the network is still able to assimilate new knowledge even if there are no free parameters.

**Task prediction.** In the most general case, every task may consist of objects of different classes. Therefore, modeling intra-task distribution is quite challenging due to the multimodality of the distribution. In Chapter 3, we used two strategies for task prediction to activate the correct subnetwork during the inference stage. The limitation of the considered strategies is the necessity of a batch of test samples for prediction. As shown in Chapter 4, this issue can be eliminated in the special case of segmentation problem if images contain only one type of defect. However, the task prediction strategy should be more general and not require multiple test samples for task identification. Moreover, in the second part of the thesis, we have considered two examples of engineering problems. In both cases, task ID was available during inference, meaning the model knows which subnetwork needs to be activated to make a prediction. Therefore, a logical extension would be to develop a task prediction strategy that is not limited to the classification problem but applies to different types of problems. One possible solution could be exploring out-of-distribution (OOD) detection literature due to the correlation between task prediction and OOD detection [149]. This would allow to first identify if a new test sample is an outlier for a certain task, and then assign it to this task.

**Knowledge transfer in engineering problems.** One of the most interesting and promising features of the continual learning algorithm is its ability to facilitate the learning process for the next tasks by leveraging the learned knowledge from the previous ones. In the second part of the thesis, we have illustrated this phenomenon in two different



terms: the improvement of prediction generalization and the reduction of the need for training data. However, in the case of CDDM (Chapter 6), we have also exemplified cases where there is no knowledge transfer between subnetworks even if a sufficient number of points were provided for training. This may happen if the tasks are not similar enough to facilitate further training by sharing parameters. In this case, it is more efficient to train a new network from scratch or find a subnetwork that does not overlap with all the previous ones. The key question here is to determine to what extent the two problems are similar (or different) and what is the acceptable similarity threshold for cooperative learning.

---

## APPENDICES



## A

## APPENDIX TO CHAPTER 2

## A.1 PRUNING SETUPS

We implement our approach with PyTorch [252]. Table A.1 shows the training parameters for each network.

Table A.1: Training setups.

Network	dataset	optimizer	learning rate (by epoch)	weight decay
LeNet-5/300-100	MNIST	Adam	$\begin{cases} 10^{-3}, & 1 \leq epoch \leq 30 \\ 10^{-4}, & 31 \leq epoch \leq 60 \end{cases}$	$5 \cdot 10^{-4}$
VGG-like/13	CIFAR-10/100, Tiny-ImageNet	SGD/Adam	$\begin{cases} 10^{-1}/10^{-3}, & 1 \leq epoch \leq 80 \\ 10^{-2}/10^{-4}, & 81 \leq epoch \leq 120 \\ 10^{-3}/10^{-5}, & 121 \leq epoch \leq 150 \end{cases}$	$5 \cdot 10^{-4}$
ResNet-20/56	CIFAR-10/100	SGD	$\begin{cases} 0.1, & 1 \leq epoch \leq 80 \\ 0.01, & 81 \leq epoch \leq 120 \\ 0.001, & 121 \leq epoch \leq 150 \end{cases}$	$5 \cdot 10^{-4}$
		Adam	$\begin{cases} 10^{-3}, & 1 \leq epoch \leq 120 \\ 10^{-4}, & 121 \leq epoch \leq 160 \\ 10^{-5}, & 161 \leq epoch \leq 200 \end{cases}$	$5 \cdot 10^{-4}$

Table A.2 shows the parameters for retraining on every iteration.

Table A.3 shows the pruning parameters and total number of iterations.

Table A.2: Retraining parameters

Network	dataset	optimizer	learning rate (by epoch)	weight decay
LeNet-5/300-100	MNIST	Adam	$\begin{cases} 10^{-3}, & 1 \leq epoch \leq 30 \\ 10^{-4}, & 31 \leq epoch \leq 60 \end{cases}$	$5 \cdot 10^{-4}$
VGG-like/13	CIFAR-10/100, Tiny-ImageNet	SGD/Adam	$\begin{cases} 10^{-1}/10^{-3}, & 1 \leq epoch \leq 20 \\ 10^{-2}/10^{-4}, & 21 \leq epoch \leq 40 \\ 10^{-3}/10^{-5}, & 41 \leq epoch \leq 60 \end{cases}$	$5 \cdot 10^{-4}$
	CIFAR-10	SGD/Adam	$\begin{cases} 10^{-1}/10^{-3}, & 1 \leq epoch \leq 20 \\ 10^{-2}/10^{-4}, & 21 \leq epoch \leq 40 \\ 10^{-3}/10^{-5}, & 41 \leq epoch \leq 60 \end{cases}$	$5 \cdot 10^{-4}$
ResNet-20/56	CIFAR-100	SGD/Adam	$\begin{cases} 10^{-1}/10^{-3}, & 1 \leq epoch \leq 30 \\ 10^{-2}/10^{-4}, & 21 \leq epoch \leq 60 \\ 10^{-3}/10^{-5}, & 41 \leq epoch \leq 80 \end{cases}$	$5 \cdot 10^{-4}$

Table A.3: Pruning parameters

Network	dataset	$(\alpha_{conv}, \alpha_{fc})$	# iterations	best iteration SGD/Adam	it-
LeNet-300-100	MNIST	(-, 0.95)	15	11	
LeNet-5	MNIST	(0.9, 0.95)	20	20	
VGG-like	CIFAR-10	(0.95, 0.95)	6	6	
	Tiny-ImageNet	(0.95, 0.95)	5	5	
ResNet-20 (SGD/Adam)	CIFAR-10	(0.95, 0.99)	10	7/10	
	CIFAR-100	(0.95, 0.99)	10	7/8	
ResNet-56 (SGD/Adam)	CIFAR-10	(0.95, 0.99)	10	6/2	
	CIFAR-100	(0.95, 0.99)	10	6/9	

## A.2 FLOPs COMPUTATION

According to [261], we compute FLOPs as follows:

- for a convolutional layer:  $FLOPs = 2HW(C_{in}K^2 + 1)C_{out}$  where  $H, W$  and  $C_{in}$  are height, width and number of channels of the input feature map,  $K$  is the kernel width (and height due to symmetry), and  $C_{out}$  is the number of output channels.
- for a fully connected layer:  $FLOPs = (2I - 1)O$ , where  $I$  is the input dimensionality and  $O$  is the output dimensionality.

# B

## B

## APPENDIX TO CHAPTER 3

### B.1 ADDITIONAL INFORMATION ON CIFAR-100 EXPERIMENTS

**Task-selection** We present CP&S results with different test batch sizes and task-selection strategies in Figure B.1.

Also, we provide an additional comparison between maxoutput and IS strategies in Figs. B.2 and B.3. In both cases, we observe the advantage of importance scores (IS) over maxoutput strategy in the case of imbalanced tasks.

**Training hyperparameters** In Table B.1, we show the hyperparameters that we used for experiments on CIFAR-100 in Section 3.4. For iTAML, all the parameters are taken from the original work and the results were reproduced using the official GitHub repository. Memory buffer contains 2000 training samples to mitigate forgetting. For CP&S, we used 3 pruning iterations, 1000 training samples per task to estimate importance scores in NNrelief and  $\alpha_{conv} = 0.9$ . For retraining (after pruning sep), we use 40 epochs with Learning Rate (LR) 0.01 multiplied by 0.2 on epochs 15, 25 and 40.

Table B.1: Hyperparameters for (ResNet-18)/3 training on CIFAR-100 (5/10/20 tasks).

Method	# epochs	optimizer	LR	LR scheduler	weight decay
iTAML	70	RAdam [262]	0.01	on epochs 20, 40, 60 multiply LR by 0.2	0
CP&S (ours)	70	Adam	0.01	on epochs 20, 40, 60 multiply LR by 0.2	0.0005

In Table B.2, we present the training hyperparameters for experiments in Section 3.5. To reproduce the results, we use PODNet and AFC GitHub repositories using the hyperparameters from the original works. All the previous works use 2000 training samples in the fixed-size memory buffer to mitigate forgetting. For CP&S, we used 1 pruning

B

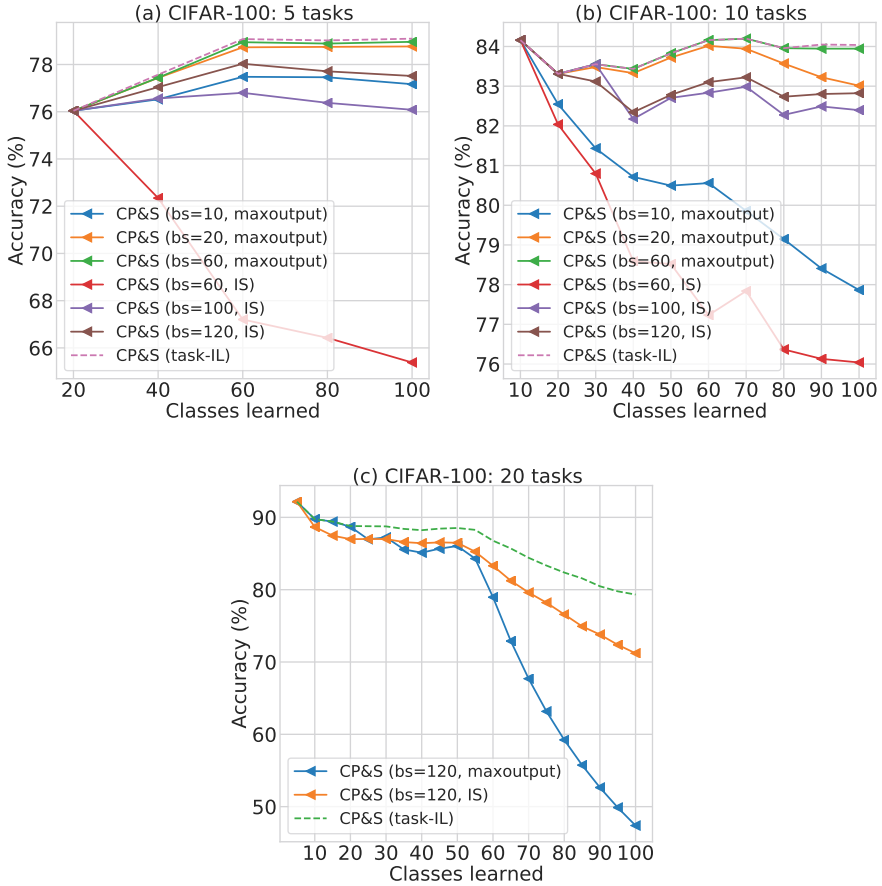


Figure B.1: The performance of CP&S with different batch sizes and task-selection strategies.

iteration, 1000 training samples per task to estimate importance scores in NNrelief and  $\alpha_{conv} = 0.9$ . For retraining (after the pruning step), we use 50 epochs with LR 0.001 multiplied by 0.1 on epochs 20 and 40.

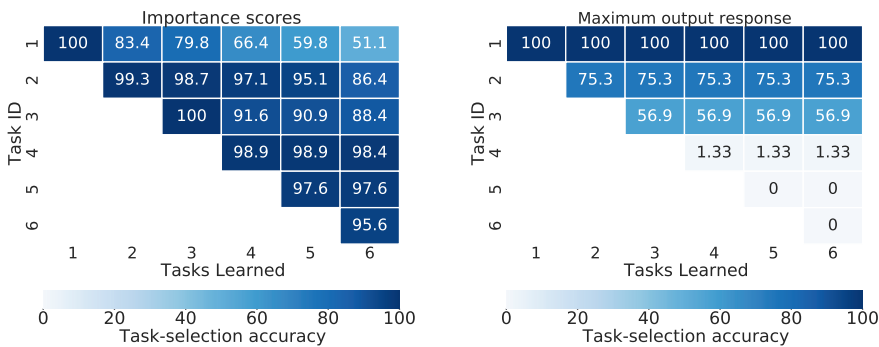


Figure B.2: Task-selection accuracy using Importance Scores (IS) (left) as opposed to maxoutput (right) on CIFAR-100 with class imbalance (50 classes in the first task and 10 classes in each of the following five tasks) for CP&S. The test batch size is 60 images in both cases.

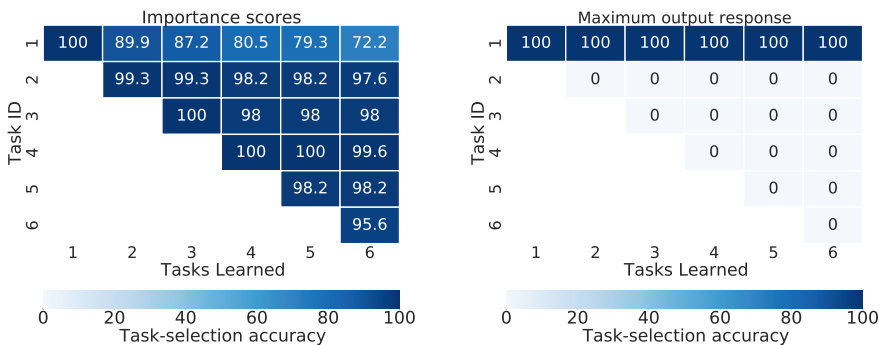


Figure B.3: Task-selection accuracy using Importance Scores (IS) (left) as opposed to maxoutput (right) on CIFAR-100 with class imbalance (50 classes in the first task and 10 classes in each of the following five tasks) for CP&S-frozen. The test batch size is 60 images in both cases.

Table B.2: Hyperparameters for ResNet-32 training on CIFAR-100 (6 tasks).

Method	# epochs	optimizer	LR	LR scheduler	weight decay
iCaRL	70	SGD	2.0	on epochs 49 and 63 multiply LR by 0.2	0.00005
LUCIR	160	SGD	0.1	on epochs 80, 120 multiply LR by 0.1	0.0005
PODNet	160	SGD	0.1	on epochs 80, 120 multiply LR by 0.1	0.0005
AFC	160	SGD	0.1	on epochs 80, 120 multiply LR by 0.1	0.0005
CP&S (ours)	160	Adam	0.001	on epochs 80, 120 multiply LR by 0.1	0.0005



## B.2 IMAGENET-100/1000 RESULTS

For ImageNet-100/1000, we present exact numbers from which the plots are constructed for CP&S in Tables B.3 and B.4.

Table B.3: ImageNet-100 results with different test batch sizes and task-IL scenario trained with SGD and Adam.

optimizer	batch size	1	2	3	4	5	6	7	8	9	10
Adam	20	98.20	98.80	98.67	98.50	98.48	98.60	98.63	98.63	98.50	98.38
	10	98.20	98.80	98.67	98.41	98.25	98.15	97.90	97.94	97.23	97.00
	5	98.20	98.02	97.03	95.86	94.23	92.51	92.08	92.12	90.53	89.39
	task-IL	98.20	98.80	98.67	98.50	98.48	98.60	98.63	98.63	98.50	98.38
SGD	20	99.00	98.90	98.67	98.60	97.90	98.09	98.05	98.25	94.20	92.62
	task-IL	99.00	98.90	98.67	98.60	98.20	98.33	98.26	98.43	98.20	98.06

Table B.4: ImageNet-1000 results with different test batch sizes and task-IL scenario trained with SGD.

optimizer	batch size	1	2	3	4	5	6	7	8	9	10
SGD	50	94.38	94.96	94.52	94.42	94.40	94.45	94.40	94.16	93.88	93.77
	20	94.38	94.96	94.52	94.42	94.40	94.40	94.34	94.12	93.77	93.66
	10	94.38	94.96	94.42	94.09	93.91	93.70	93.40	92.92	92.19	91.97
	5	94.38	94.31	92.74	91.53	90.41	89.24	88.07	86.59	84.82	83.92
	task-IL	94.38	94.96	94.52	94.42	94.40	94.45	94.40	94.16	93.88	93.77

## B.3 CUB-200-2011 ADDITIONAL COMPARISON

In this section, we provide an additional comparison for ResNet-18 on CUB-200-2011 dataset using 5 test images per batch to predict the task-ID in Fig. B.4.

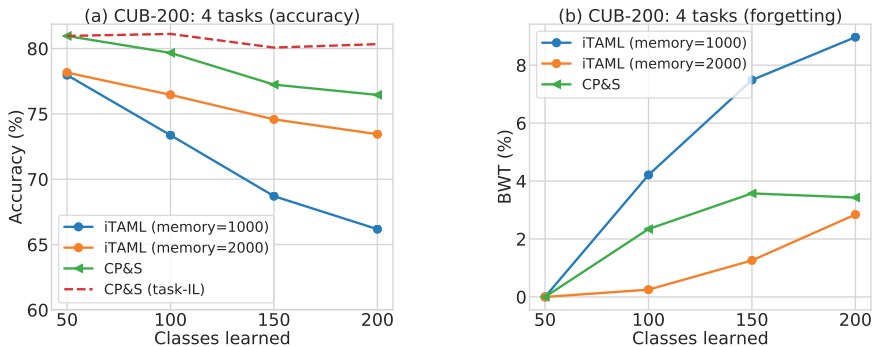


Figure B.4: Comparison with iTAML on four tasks constructed from CUB-200-2011. Notation: “memory” is the number for images from previous tasks; “task-IL” refers to task-IL scenario as an upper-bound for CP&S.

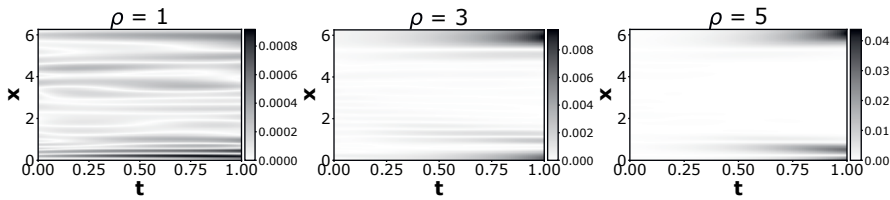
## C

## C

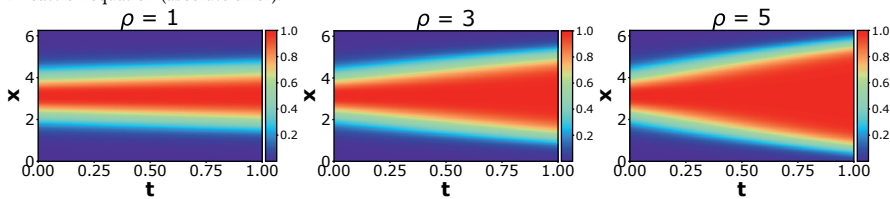
## APPENDIX TO CHAPTER 5

## C.1 ADDITIONAL INFORMATION ABOUT 1-D REACTION PDE

Figure C.1 illustrates the absolute error and exact solutions for 1-D convection PDE considering the values of parameter  $\rho = 1, 3, 5$ .



(a) 1-D reaction equation (absolute error).



(b) 1-D reaction equation (exact solution).

Figure C.1: iPINNs on 1-D reaction equation.

In the case of periodic boundary conditions that we have considered, the analytical solution is [190]:

$$u_{\text{analytical}}(x, t) = \frac{h(x) e^{\rho t}}{h(x) e^{\rho t} + 1 - h(x)},$$

where  $h(x)$  is the initial condition.

## C.2 ADDITIONAL INFORMATION ABOUT 1-D CONVECTION PDE

Figure C.2 illustrates the absolute error and exact solutions for 1-D convection PDE considering the values of parameter  $\beta = 1, 20, 40$ .

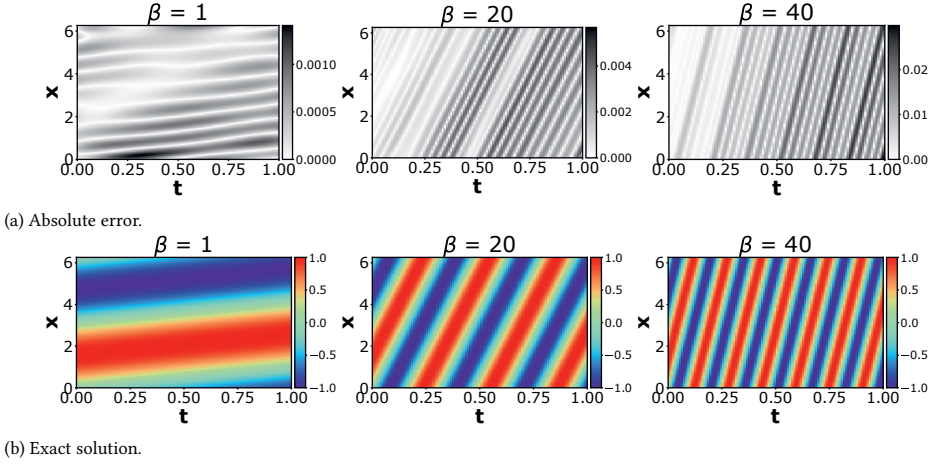


Figure C.2: iPINNs on 1-D convection equation.

# D

## APPENDIX TO CHAPTER 6

**D**

### D.1 ADDITIONAL INFORMATION ON THE FIRST CASE STUDY

An elastoplastic von Mises material with hardening is investigated to create a path-dependent problem. A fixed-sized square is utilized as a domain and holes of varying sizes and locations are placed inside the domain to create different tasks (see Figure D.1). For all the tasks the bottom part of the domain is fixed and the top part is deformed according to a uniform displacement in  $u_{x_1;t}$  and  $u_{y_1;t}$ .

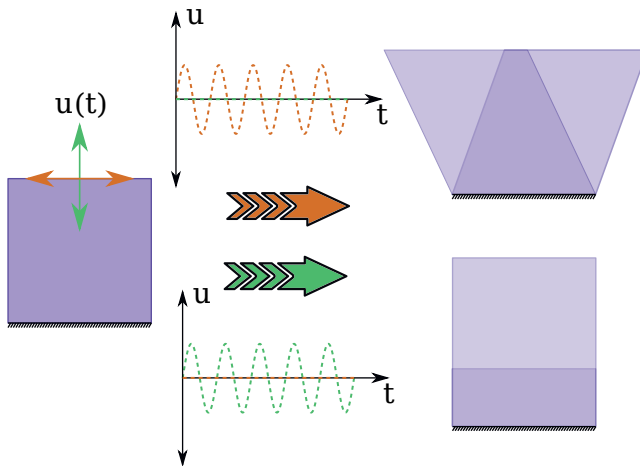


Figure D.1: A square domain fixed on the bottom and displaced on the top. Displacement is done in a pseudo-time.



---

# BIBLIOGRAPHY

## REFERENCES

- [1] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.
- [2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- [4] 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*, pages 770–778, 2016.
- [5] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.
- [6] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.
- [7] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *Journal of Intelligent & Robotic Systems*, 105(1):9, 2022.
- [8] Matthias Perkonigg, Johannes Hofmanninger, Christian J Herold, James A Brink, Oleg Pinykh, Helmut Prosch, and Georg Langs. Dynamic memory to alleviate catastrophic forgetting in continual learning with medical imaging. *Nature communications*, 12(1):5678, 2021.
- [9] Chaitanya Baweja, Ben Glocker, and Konstantinos Kamnitsas. Towards continual learning in medical imaging. In *Medical imaging meets NIPS workshop, 32nd conference on neural information processing systems (NIPS)*, 2018.
- [10] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.
- [11] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *2nd International Conference on Learning Representations, ICLR*, 2014.

- [12] Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.
- [13] Wickliffe C Abraham and Anthony Robins. Memory retention—the synaptic stability versus plasticity dilemma. *Trends in neurosciences*, 28(2):73–78, 2005.
- [14] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects, 2013.
- [15] Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.
- [16] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.
- [17] Guido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.
- [18] M Jehanzeb Mirza, Marc Masana, Horst Possegger, and Horst Bischof. An efficient domain-incremental learning approach to drive in all weather conditions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3001–3011, 2022.
- [19] Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D. Bagdanov, and Joost van de Weijer. Class-incremental learning: Survey and performance evaluation on image classification. *IEEE transactions on pattern analysis and machine intelligence*, 2020.
- [20] David Attwell and Simon B Laughlin. An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, 21(10):1133–1145, 2001.
- [21] Yulia Lerner, Christopher J Honey, Lauren J Silbert, and Uri Hasson. Topographic mapping of a hierarchy of temporal receptive windows using a narrated story. *Journal of Neuroscience*, 31(8):2906–2915, 2011.
- [22] Aleksandr Dekhovich, David MJ Tax, Marcel HF Sluiter, and Miguel A Bessa. Neural network relief: a pruning algorithm based on neural activity. *Machine Learning*, 113:2597–2618, 2024.
- [23] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- [24] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

- [25] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.
- [26] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.
- [27] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.
- [28] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.
- [29] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.
- [30] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.
- [31] Marco Ancona, Cengiz Öztireli, and Markus Gross. Shapley value as principled metric for structured network pruning. *arXiv preprint arXiv:2006.01795*, 2020.
- [32] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.
- [33] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.
- [34] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019.
- [35] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.
- [36] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.
- [37] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.
- [38] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.



- [39] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in neural information processing systems*, pages 3288–3298, 2017.
- [40] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.
- [41] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240, 2018.
- [42] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.
- [43] 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*, pages 7765–7773, 2018.
- [44] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.
- [45] Subutai Ahmad and Luiz Scheinkman. How can we be so dense? the benefits of using highly sparse representations. *arXiv preprint arXiv:1903.11257*, 2019.
- [46] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 111–120, 2019.
- [47] Ningyi Liao, Shufan Wang, Liyao Xiang, Nanyang Ye, Shuo Shao, and Pengzhi Chu. Achieving adversarial robustness via sparsity. *Machine Learning*, 111(2):685–711, 2022.
- [48] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.
- [49] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference of Learning Representation (ICLR)*, 2017.
- [50] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.
- [51] Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. *Advances in Neural Information Processing Systems*, 30:3177–3186, 2017.

- [52] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.
- [53] Yash Garg and K Selçuk Candan. iSparse: Output informed sparsification of neural network. In *Proceedings of the 2020 International Conference on Multimedia Retrieval*, pages 180–188, 2020.
- [54] Namhoon Lee, Thalaisyasingam Ajanthan, and Philip HS Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019.
- [55] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.
- [56] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.
- [57] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.
- [58] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.
- [59] Lili Geng and Baoning Niu. Pruning convolutional neural networks via filter similarity analysis. *Machine Learning*, 111(9):3161–3180, 2022.
- [60] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34:24604–24616, 2021.
- [61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [62] Sergey Zagoruyko. 92.45 on cifar-10 in torch, 2015. URL <http://torch.ch/blog/2015/07/30/cifar.html>, 2015.
- [63] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.
- [64] 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*, pages 1026–1034, 2015.

- [65] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pages 1379–1387, 2016.
- [66] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.
- [67] Dushyant Mehta, Kwang In Kim, and Christian Theobalt. On implicit filter level sparsity in convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 520–528, 2019.
- [68] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019.
- [69] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.
- [70] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, Boston, MA, 1998.
- [71] Guy Oren and Lior Wolf. In defense of the learning without forgetting for task incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2209–2218, 2021.
- [72] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, pages 3400–3409, 2017.
- [73] Juan-Manuel Perez-Rua, Xiatian Zhu, Timothy M Hospedales, and Tao Xiang. Incremental few-shot object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13846–13855, 2020.
- [74] Jianren Wang, Xin Wang, Yue Shang-Guan, and Abhinav Gupta. Wanderlust: Online continual object detection in the real world. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10829–10838, 2021.
- [75] Shipeng Yan, Jiale Zhou, Jiangwei Xie, Songyang Zhang, and Xuming He. An em framework for online incremental learning of semantic segmentation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 3052–3060, 2021.
- [76] Arthur Douillard, Yifu Chen, Arnaud Dapogny, and Matthieu Cord. Plop: Learning without forgetting for continual semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4040–4050, 2021.
- [77] Minh Hieu Phan, Son Lam Phung, Long Tran-Thanh, Abdesselam Bouzerdoum, et al. Class similarity weighted knowledge distillation for continual semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16866–16875, 2022.

- [78] Asieh Zadbood, Janice Chen, Yuan Chang Leong, Kenneth A Norman, and Uri Hasson. How we transmit memories to other brains: constructing shared neural representations via communication. *Cerebral cortex*, 27(10):4988–5000, 2017.
- [79] Peter R Huttenlocher. Morphometric study of human cerebral cortex development. *Neuropsychologia*, 28(6):517–527, 1990.
- [80] Peter Lennie. The cost of cortical computation. *Current biology*, 13(6):493–497, 2003.
- [81] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.
- [82] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyang Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2019.
- [83] Xiaolei 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)*, pages 2262–2268. IEEE, 2018.
- [84] 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. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.
- [85] 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)*, pages 139–154, 2018.
- [86] 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*, pages 2001–2010, 2017.
- [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*, pages 831–839, 2019.
- [88] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 583–592, 2019.
- [89] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 86–102. Springer, 2020.
- [90] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *6th International Conference on Learning Representations, ICLR*, 2018.

- [91] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems*, 2020.
- [92] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021.
- [93] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *Advances in neural information processing systems*, 30, 2017.
- [94] 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)*, pages 532–547, 2018.
- [95] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, 2017.
- [96] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *European Conference on Computer Vision*, pages 488–501. Springer, 2012.
- [97] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.
- [98] 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*, pages 374–382, 2019.
- [99] Minsoo Kang, Jaeyoo Park, and Bohyung Han. Class-incremental learning by knowledge distillation with adaptive feature consolidation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16071–16080, 2022.
- [100] 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*, pages 1131–1140, 2020.
- [101] Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for incremental learning. In *Advances in Neural Information Processing Systems*, 2019.
- [102] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3014–3023, 2021.

- [103] Fu Lee Wang, Da-Wei Zhou, Han-Jia Ye, and De chuan Zhan. Foster: Feature boosting and compression for class-incremental learning. In *European Conference on Computer Vision*, 2022.
- [104] Jathushan Rajasegaran, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. itaml: An incremental task-agnostic meta-learning approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13588–13597, 2020.
- [105] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.
- [106] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. In *NeurIPS Workshop on Real Neurons & Hidden Units*, 2019.
- [107] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [108] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.
- [109] Eun Sung Kim, Jung Uk Kim, Sangmin Lee, Sang-Keun Moon, and Yong Man Ro. Class incremental learning with task-selection. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1846–1850. IEEE, 2020.
- [110] David Lopez-Paz and Marc’Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, volume 30, pages 6467–6476, 2017.
- [111] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [112] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.
- [113] Shenyang Huang, Vincent Francois-Lavet, and Guillaume Rabusseau. Understanding capacity saturation in incremental learning. In *Canadian Conference on AI*, 2021.
- [114] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

- [115] Michela Prunella, Roberto M Scardigno, Domenico Buongiorno, Antonio Brunetti, Nicola Longo, Raffaele Carli, Mariagrazia Dotoli, and Vitoantonio Bevilacqua. Deep learning for automatic vision-based recognition of industrial surface defects: a survey. *IEEE Access*, 2023.
- [116] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.
- [117] Shin-Min Chao and Du-Ming Tsai. An anisotropic diffusion-based defect detection for low-contrast glass substrates. *Image and Vision Computing*, 26(2):187–200, 2008.
- [118] Kechen Song and Yunhui Yan. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Applied Surface Science*, 285:858–864, 2013.
- [119] Yong-Ju Jeon, Doo-chul Choi, Sang Jun Lee, Jong Pil Yun, and Sang Woo Kim. Defect detection for corner cracks in steel billets using a wavelet reconstruction method. *JOSA A*, 31(2):227–237, 2014.
- [120] Hongbin Jia, Yi Lu Murphey, Jinajun Shi, and Tzyy-Shuh Chang. An intelligent real-time vision system for surface defect detection. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 239–242. IEEE, 2004.
- [121] Kuldeep Agarwal, Rajiv Shivpuri, Yijun Zhu, Tzyy-Shuh Chang, and Howard Huang. Process knowledge based multi-class support vector classification (pk-msvm) approach for surface defects in hot rolling. *Expert Systems with Applications*, 38(6):7251–7262, 2011.
- [122] Rajalingappaa Shanmugamani, Mohammad Sadique, and B Ramamoorthy. Detection and classification of surface defects of gun barrels using computer vision and machine learning. *Measurement*, 60:222–230, 2015.
- [123] Yu He, Kechen Song, Qinggang Meng, and Yunhui Yan. An end-to-end steel surface defect detection approach via fusing multiple hierarchical features. *IEEE transactions on instrumentation and measurement*, 69(4):1493–1504, 2019.
- [124] Domen Tabernik, Samo Šela, Jure Skvarč, and Danijel Skočaj. Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, 31(3):759–776, 2020.
- [125] Masood Aslam, Tariq M Khan, Syed Saud Naqvi, Geoff Holmes, and Rafea Naffa. Ensemble convolutional neural networks with knowledge transfer for leather defect classification in industrial settings. *IEEE Access*, 8:198600–198614, 2020.
- [126] Hao Wu and Quanquan Lv. Hot-rolled steel strip surface inspection based on transfer learning model. *Journal of Sensors*, 2021:1–8, 2021.

- [127] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III* 18, pages 234–241. Springer, 2015.
- [128] Di He, Ke Xu, and Peng Zhou. Defect detection of hot rolled steels with a new object detection framework called classification priority network. *Computers & Industrial Engineering*, 128:290–297, 2019.
- [129] Guorong Song, Kechen Song, and Yunhui Yan. Edrnet: Encoder–decoder residual network for salient object detection of strip steel surface defects. *IEEE Transactions on Instrumentation and Measurement*, 69(12):9709–9719, 2020.
- [130] Ruiyang Hao, Bingyu Lu, Ying Cheng, Xiu Li, and Biqing Huang. A steel surface defect inspection approach towards smart industrial monitoring. *Journal of Intelligent Manufacturing*, 32:1833–1843, 2021.
- [131] Yibin Huang, Congying Qiu, and Kui Yuan. Surface defect saliency of magnetic tile. *The Visual Computer*, 36:85–96, 2020.
- [132] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [133] Yue Pan and Limao Zhang. Dual attention deep learning network for automatic steel surface defect segmentation. *Computer-Aided Civil and Infrastructure Engineering*, 37(11):1468–1487, 2022.
- [134] Hüseyin Üzen, Muammer Türkoğlu, Berrin Yanikoglu, and Davut Hanbay. Swin-mfinet: Swin transformer based multi-feature integration network for detection of pixel-level surface defects. *Expert Systems with Applications*, 209:118269, 2022.
- [135] Xiaoming Lv, Fajie Duan, Jia-jia Jiang, Xiao Fu, and Lin Gan. Deep metallic surface defect detection: The new benchmark and detection network. *Sensors*, 20(6):1562, 2020.
- [136] Xinglong Feng, Xianwen Gao, and Ling Luo. X-sdd: A new benchmark for hot rolled steel strip surface defects detection. *Symmetry*, 13(4):706, 2021.
- [137] Tao Liu and Wei Ye. A semi-supervised learning method for surface defect classification of magnetic tiles. *Machine Vision and Applications*, 33(2):35, 2022.
- [138] Robert Coop, Aaron Mishtal, and Itamar Arel. Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting. *IEEE transactions on neural networks and learning systems*, 24(10):1623–1634, 2013.
- [139] Guorong Song, Kechen Song, and Yunhui Yan. Saliency detection for strip steel surface defects using multiple constraints and improved texture features. *Optics and Lasers in Engineering*, 128:106000, 2020.



- [140] Karin van Garderen, Sebastian van der Voort, Fatih Incekara, Marion Smits, and Stefan Klein. Towards continuous learning for glioma segmentation with elastic weight consolidation. In *International Conference on Medical Imaging with Deep Learning –Extended Abstract Track*, 2019.
- [141] Marvin Klingner, Andreas Bär, Philipp Donn, and Tim Fingscheidt. Class-incremental learning for semantic segmentation re-using neither old data nor old labels. In *2020 IEEE 23rd international conference on intelligent transportation systems (ITSC)*, pages 1–8. IEEE, 2020.
- [142] Sungmin Cha, YoungJoon Yoo, Taesup Moon, et al. Ssul: Semantic segmentation with unknown label for exemplar-based class-incremental learning. *Advances in neural information processing systems*, 34:10919–10930, 2021.
- [143] Yiqiao Qiu, Yixing Shen, Zhuohao Sun, Yanchong Zheng, Xiaobin Chang, Weishi Zheng, and Ruixuan Wang. Sats: Self-attention transfer for continual semantic segmentation. *Pattern Recognition*, 138:109383, 2023.
- [144] Hasan Tercan, Philipp Deibert, and Tobias Meisen. Continual learning of neural networks for quality prediction in production using memory aware synapses and weight transfer. *Journal of Intelligent Manufacturing*, 33(1):283–292, 2022.
- [145] Benjamin Maschler, Thi Thu Huong Pham, and Michael Weyrich. Regularization-based continual learning for anomaly detection in discrete manufacturing. *Procedia CIRP*, 104:452–457, 2021.
- [146] Benjamin Maschler, Sophia Tatiyosyan, and Michael Weyrich. Regularization-based continual learning for fault prediction in lithium-ion batteries. *Procedia CIRP*, 112:513–518, 2022.
- [147] Wenbo Sun, Raed Al Kontar, Judy Jin, and Tzyy-Shuh Chang. A continual learning framework for adaptive defect classification and inspection. *arXiv preprint arXiv:2203.08796*, 2022.
- [148] Aleksandr Dekhovich, David MJ Tax, Marcel HF Sluiter, and Miguel A Bessa. Continual prune-and-select: class-incremental learning with specialized subnetworks. *Applied Intelligence*, 53(14):17849–17864, 2023.
- [149] Gyuhak Kim, Changnan Xiao, Tatsuya Konishi, Zixuan Ke, and Bing Liu. A theoretical study on solving continual learning. *Advances in Neural Information Processing Systems*, 35:5065–5079, 2022.
- [150] Matthias Dorfer, Rainer Kelz, and Gerhard Widmer. Deep linear discriminant analysis. *arXiv preprint arXiv:1511.04707*, 2015.
- [151] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 220–221, 2020.

- [152] Gido M Van De Ven, Zhe Li, and Andreas S Tolias. Class-incremental learning with generative classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3611–3620, 2021.
- [153] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [154] Sanjoy Dasgupta and Daniel Hsu. On-line estimation with the multivariate gaussian distribution. In *International Conference on Computational Learning Theory*, pages 278–292. Springer, 2007.
- [155] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International workshop on machine learning in medical imaging*, pages 379–387. Springer, 2017.
- [156] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [157] Celia Fernández Madrazo, Ignacio Heredia, Lara Lloret, and Jesús Marco de Lucas. Application of a convolutional neural network for image classification for the analysis of collisions in high energy physics. In *EPJ Web of Conferences*, volume 214, page 06017. EDP Sciences, 2019.
- [158] Alexander Bogatskiy, Brandon Anderson, Jan Offermann, Marwah Roussi, David Miller, and Risi Kondor. Lorentz group equivariant neural network for particle physics. In *International Conference on Machine Learning*, pages 992–1002. PMLR, 2020.
- [159] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, 2020.
- [160] Omar Khatib, Simiao Ren, Jordan Malof, and Willie J Padilla. Learning the physics of all-dielectric metamaterials with deep lorentz neural networks. *Advanced Optical Materials*, page 2200097, 2022.
- [161] Gonçalo Marques, Deevyankar Agarwal, and Isabel de la Torre Díez. Automated medical diagnosis of covid-19 through efficientnet convolutional neural network. *Applied soft computing*, 96:106691, 2020.
- [162] Tapas Si, Jayri Bagchi, and Péricles BC Miranda. Artificial neural network training using metaheuristics for medical data classification: an experimental study. *Expert Systems with Applications*, 193:116423, 2022.
- [163] DR Sarvamangala and Raghavendra V Kulkarni. Convolutional neural networks in medical image understanding: a survey. *Evolutionary intelligence*, 15(1):1–22, 2022.

- [164] Tadaaki Hosaka. Bankruptcy prediction using imaged financial ratios and convolutional neural networks. *Expert systems with applications*, 117:287–299, 2019.
- [165] Pengfei Yu and Xuesong Yan. Stock price prediction based on deep neural networks. *Neural Computing and Applications*, 32(6):1609–1628, 2020.
- [166] Periklis Gogas and Theophilos Papadimitriou. Machine learning in economics and finance. *Computational Economics*, 57(1):1–4, 2021.
- [167] Jun Wang and Xin Gan. Neurodynamics-driven portfolio optimization with targeted performance criteria. *Neural Networks*, 157:404–421, 2023.
- [168] Miguel A Bessa, R Bostanabad, Zeliang Liu, A Hu, Daniel W Apley, C Brinson, Wei Chen, and Wing Kam Liu. A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, 320:633–667, 2017.
- [169] Ivan Sosnovik and Ivan Oseledets. Neural networks for topology optimization. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 34(4):215–223, 2019.
- [170] Aaditya Chandrasekhar and Krishnan Suresh. Tounn: topology optimization using neural networks. *Structural and Multidisciplinary Optimization*, 63(3):1135–1149, 2021.
- [171] Nerea Portillo Juan and Vicente Negro Valdecantos. Review of the application of artificial neural networks in ocean engineering. *Ocean Engineering*, 259:111947, 2022.
- [172] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.
- [173] Andrew J Meade Jr and Alvaro A Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9):19–44, 1994.
- [174] R Yentis and ME Zaghoul. Vlsi implementation of locally connected neural network for solving partial differential equations. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(8):687–690, 1996.
- [175] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural network methods in quantum mechanics. *Computer Physics Communications*, 104(1-3):1–14, 1997.
- [176] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [177] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

- [178] Henning Wessels, Christian Weißenfels, and Peter Wriggers. The neural particle method—an updated lagrangian physics informed neural network for computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 368:113127, 2020.
- [179] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12, 2022.
- [180] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics express*, 28(8):11618–11633, 2020.
- [181] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.
- [182] Peter R Wiecha, Arnaud Arbouet, Christian Girard, and Otto L Muskens. Deep learning in nano-photonics: inverse design and beyond. *Photonics Research*, 9(5):B182–B200, 2021.
- [183] Yuexing Bai, Temuer Chaolu, and Sudaο Bilige. The application of improved physics-informed neural network (ipinn) method in finance. *Nonlinear Dynamics*, 107(4):3655–3667, 2022.
- [184] Ameya D Jagtap and George E Karniadakis. Extended physics-informed neural networks (xpinn): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In *AAAI Spring Symposium: MLPS*, 2021.
- [185] Zongren Zou and George Em Karniadakis. L-hydra: Multi-head physics-informed neural networks. *arXiv preprint arXiv:2301.02152*, 2023.
- [186] Xu Liu, Xiaoya Zhang, Wei Peng, Weien Zhou, and Wen Yao. A novel meta-learning initialization method for physics-informed neural networks. *Neural Computing and Applications*, 34(17):14511–14534, 2022.
- [187] Michael Penwarden, Shandian Zhe, Akil Narayan, and Robert M Kirby. A meta-learning approach for physics-informed neural networks (pinns): Application to parameterized pdes. *Journal of Computational Physics*, 477:111912, 2023.
- [188] Fabio Cermelli, Antonino Geraci, Dario Fontanel, and Barbara Caputo. Modeling missing annotations for incremental learning in object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3700–3710, 2022.
- [189] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

- [190] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.
- [191] Franz M Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C Geiger. Understanding the difficulty of training physics-informed neural networks on dynamical systems. *arXiv preprint arXiv:2203.13648*, 2022.
- [192] Rambod Mojjani, Maciej Balajewicz, and Pedram Hassanzadeh. Lagrangian pinns: A causality-conforming solution to failure modes of physics-informed neural networks. *arXiv preprint arXiv:2205.02902*, 2022.
- [193] Colby L Wight and Jia Zhao. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020.
- [194] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, 2012.
- [195] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.
- [196] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18, 2019.
- [197] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.
- [198] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.
- [199] Sina Amini Niaki, Ehsan Haghighat, Trevor Campbell, Anoush Poursartip, and Reza Vaziri. Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. *Computer Methods in Applied Mechanics and Engineering*, 384:113959, 2021.
- [200] Souvik Chakraborty. Transfer learning based multi-fidelity physics informed deep neural network. *Journal of Computational Physics*, 426:109942, 2021.
- [201] Chen Xu, Ba Trung Cao, Yong Yuan, and Günther Meschke. Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Computer Methods in Applied Mechanics and Engineering*, 405:115852, 2023.

- [202] Xinhai Chen, Chunye Gong, Qian Wan, Liang Deng, Yunbo Wan, Yang Liu, Bo Chen, and Jie Liu. Transfer learning for deep neural network-based partial differential equations solving. *Advances in Aerodynamics*, 3(1):1–14, 2021.
- [203] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.
- [204] Tomasz Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, pages 203–224. Springer Singapore, Singapore, 2021.
- [205] Ameya D Jagtap and George Em Karniadakis. How important are activation functions in regression and classification? a survey, performance comparison, and future directions. *arXiv preprint arXiv:2209.02681*, 2022.
- [206] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.
- [207] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.
- [208] Subana Shanmuganathan. Artificial neural network modelling: An introduction. In *Artificial neural network modelling*, pages 1–14. Springer, 2016.
- [209] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016.
- [210] Jonathan Schmidt, Mário RG Marques, Silvana Botti, and Miguel AL Marques. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):1–36, 2019.
- [211] M.A. Bessa, R. Bostanabad, Z. Liu, A. Hu, Daniel W. Apley, C. Brinson, W. Chen, and Wing Kam Liu. A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, 320:633–667, June 2017.
- [212] German Capuano and Julian J Rimoli. Smart finite elements: A novel machine learning application. *Computer Methods in Applied Mechanics and Engineering*, 345:363–381, 2019.
- [213] Prakash Thakolkaran, Akshay Joshi, Yiwen Zheng, Moritz Flaschel, Laura De Lorenzis, and Siddhant Kumar. Nn-euclid: deep-learning hyperelasticity without stress data. *arXiv preprint arXiv:2205.06664*, 2022.
- [214] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

- [215] J. Ghaboussi, J. H. Garrett, and X. Wu. Knowledge-based modeling of material behavior with neural networks. *Journal of Engineering Mechanics*, 117(1):132–153, 1991.
- [216] Rubén Ibanez, Emmanuelle Abisset-Chavanne, Jose Vicente Aguado, David Gonzalez, Elias Cueto, and Francisco Chinesta. A manifold learning approach to data-driven computational elasticity and inelasticity. *Archives of Computational Methods in Engineering*, 25:47–57, 2018.
- [217] RE Jones, JA Templeton, CM Sanders, and JT Ostien. Machine learning models of plastic flow based on representation theory. *CMES-Computer Modeling in Engineering & Sciences*, 117(3), 2018.
- [218] Lu Trong Khiem Nguyen and Marc-André Keip. A data-driven approach to nonlinear elasticity. *Computers & Structures*, 194:97–115, 2018.
- [219] I.B.C.M. Rocha, P. Kerfriden, and F.P. van der Meer. On-the-fly construction of surrogate constitutive models for concurrent multiscale mechanical analysis through probabilistic machine learning. *Journal of Computational Physics: X*, 9:100083, 2021.
- [220] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, and M. A. Bessa. Deep learning predicts path-dependent plasticity. *Proceedings of the National Academy of Sciences*, 116(52):26414–26420, December 2019.
- [221] F Ghavamian and A Simone. Accelerating multiscale finite element simulations of history-dependent materials using a recurrent neural network. *Computer Methods in Applied Mechanics and Engineering*, 357:112594, 2019.
- [222] Nikolaos N Vlassis, Ran Ma, and WaiChing Sun. Geometric deep learning for computational mechanics part i: Anisotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering*, 371:113299, 2020.
- [223] Diab W Abueidda, Seid Koric, Nahil A Sobh, and Huseyin Sehitoglu. Deep learning for plasticity and thermo-viscoplasticity. *International Journal of Plasticity*, 136:102852, 2021.
- [224] Jan N Fuhg and Nikolaos Bouklas. The mixed deep energy method for resolving concentration features in finite strain hyperelasticity. *Journal of Computational Physics*, 451:110839, 2022.
- [225] Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.
- [226] Filippo Masi, Ioannis Stefanou, Paolo Vannucci, and Victor Maffi-Berthier. Thermodynamics-based artificial neural networks for constitutive modeling. *Journal of the Mechanics and Physics of Solids*, 147:104277, 2021.

- [227] Faisal As' ad, Philip Avery, and Charbel Farhat. A mechanics-informed artificial neural network approach in data-driven constitutive modeling. *International Journal for Numerical Methods in Engineering*, 123(12):2738–2759, 2022.
- [228] Annan Zhang and Dirk Mohr. Using neural networks to represent von mises plasticity with isotropic hardening. *International Journal of Plasticity*, 132:102732, 2020.
- [229] Peyman Saidi, Hadi Pirgazi, Mehdi Sanjari, Saeed Tamimi, Mohsen Mohammadi, Laurent K Béland, Mark R Daymond, and Isaac Tamblyn. Deep learning and crystal plasticity: A preconditioning approach for accurate orientation evolution prediction. *Computer Methods in Applied Mechanics and Engineering*, 389:114392, 2022.
- [230] Colin Bonatti, Bekim Berisha, and Dirk Mohr. From cp-fft to cp-rnn: Recurrent neural network surrogate model of crystal plasticity. *International Journal of Plasticity*, page 103430, 2022.
- [231] Zeliang Liu, CT Wu, and M391298807188754 Koishi. A deep material network for multiscale topology learning and accelerated nonlinear modeling of heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 345:1138–1168, 2019.
- [232] Grace CY Peng, Mark Alber, Adrian Buganza Tepole, William R Cannon, Suvaranu De, Salvador Dura-Bernal, Krishna Garikipati, George Karniadakis, William W Lytton, Paris Perdikaris, et al. Multiscale modeling meets machine learning: What can we learn? *Archives of Computational Methods in Engineering*, 28(3):1017–1037, 2021.
- [233] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.
- [234] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David Parkes, and Sai Srivatsa Ravindranath. Optimal auctions through deep learning. In *International Conference on Machine Learning*, pages 1706–1715. PMLR, 2019.
- [235] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.
- [236] Claudia Draxl and Matthias Scheffler. Nomad: The fair concept for big data-driven materials science. *Mrs Bulletin*, 43(9):676–682, 2018.
- [237] Annika Jacobsen, Ricardo de Miranda Azevedo, Nick Juty, Dominique Batista, Simon Coles, Ronald Cornet, Mélanie Courtot, Mercè Crosas, Michel Dumontier, Chris T Evelo, et al. Fair principles: interpretations and implementation considerations, 2020.
- [238] Rich Caruana. Learning many related tasks at the same time with backpropagation. *Advances in neural information processing systems*, 7, 1994.



- [239] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.
- [240] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.
- [241] Zeliang Liu, C. T. Wu, and M. Koishi. Transfer learning of deep material network for seamless structure–property predictions. *Computational Mechanics*, 64(2):451–465, August 2019.
- [242] Emma Lejeune and Bill Zhao. Exploring the potential of transfer learning for metamodels of heterogeneous material deformation, October 2020.
- [243] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [244] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.
- [245] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, pages 1045–1048, 2010.
- [246] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, pages 338–342, 2014.
- [247] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.
- [248] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- [249] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [250] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [251] Guang Chen. Recurrent neural networks (RNNs) learn the constitutive law of viscoelasticity. *Computational Mechanics*, page 11, 2021.
- [252] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

- [253] Ling Wu, Nanda Gopala Kilingar, Ludovic Noels, et al. A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths. *Computer Methods in Applied Mechanics and Engineering*, 369:113234, 2020.
- [254] Anders Logg, Kent-Andre Mardal, and Garth Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2012.
- [255] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.
- [256] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [257] Marc Masana, Bartłomiej Twardowski, and Joost Van de Weijer. On class orderings for incremental learning. *arXiv preprint arXiv:2007.02145*, 2020.
- [258] Jiaxiang Yi and Miguel Anibal Bessa. rvesimulator: An automated representative volume element simulator for data-driven material discovery. In *AI for Accelerated Materials Design-NeurIPS 2023 Workshop*, 2023.
- [259] Michael Smith. *ABAQUS/Standard User's Manual, Version 6.9*. Dassault Systèmes Simulia Corp, United States, 2009.
- [260] Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. *Neural computation*, 32(1):1–35, 2020.
- [261] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.
- [262] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *8th International Conference on Learning Representations, ICLR*, 2020.



# GLOSSARY

**AI** Artificial Intelligence.

**CL** Continual Learning.

**Class-IL** Class-incremental Learning.

**DL** Deep Learning.

**DNN** Deep Neural Network.

**Domain-IL** Domain-incremental Learning.

**LDA** Linear Discriminant Analysis.

**ML** Machine Learning.

**OOD** Out-of-distribution.

**PDE** Partial Differential Equation.

**PIML** Physics-informed Machine Learning.

**PINNs** Physics-informed Neural Networks.

**RVE** Representative Volume Element.

**SciML** Scientific Machine Learning.

**Task ID** Task identifier.

**Task-IL** Task-incremental Learning.




# LIST OF PUBLICATIONS

## JOURNAL PUBLICATIONS AND PREPRINTS

- 5. *Aleksandr Dekhovich* and Miguel A. Bessa. Continual learning for surface defect segmentation by subnetwork creation and selection. *Journal of Intelligent Manufacturing*, 1–15, 2024. <https://doi.org/10.1007/s10845-024-02393-4>
- 4. *Aleksandr Dekhovich*, David M.J. Tax, Marcel H.F. Sluiter, and Miguel A. Bessa. Neural network relief: a pruning algorithm based on neural activity. *Machine Learning*, **113**, 2597–2618, 2024. <https://doi.org/10.1007/s10994-024-06516-z>
- 3. *Aleksandr Dekhovich*, Marcel H.F. Sluiter, David M.J. Tax, and Miguel A. Bessa. iPINNs: Incremental learning for physics-informed neural networks. *arXiv preprint arXiv:2304.04854*, 2023.
- 2. *Aleksandr Dekhovich*, O. Taylan Turan, Jiaxiang Yi, and Miguel A. Bessa. Cooperative data-driven modeling. *Computer Methods in Applied Mechanics and Engineering*, **417**:116432, 2023. <https://doi.org/10.1016/j.cma.2023.116432>
- 1. *Aleksandr Dekhovich*, David M.J. Tax, Marcel H.F. Sluiter, and Miguel A. Bessa. Continual prune-and-select: class-incremental learning with specialized subnetworks. *Applied Intelligence*, **53**, 17849–17864, 2023. <https://doi.org/10.1007/s10489-022-04441-z>

## PEER-REVIEWED CONFERENCE PAPERS (NON-ARCHIVAL)

- 2. *Aleksandr Dekhovich*, Marcel H.F. Sluiter, David M.J. Tax, and Miguel A. Bessa. Incremental learning for physics-informed neural networks. In *Machine Learning and the Physical Sciences Workshop, NeurIPS 2023*, 2023.
- 1. *Aleksandr Dekhovich*, O. Taylan Turan, Jiaxiang Yi, and Miguel A. Bessa. Cooperative data-driven modeling: continual learning of different material behavior. In *Workshop on "Machine Learning for Materials", ICLR 2023*, 2023.

 Included in this thesis.



ISBN 978-94-6469-983-8



9 789464 699838 >