Modularity in deep learning

Haozhe Sun

université PARIS-SACLAY

Modularity in Deep Learning

Modularité dans l'Apprentissage Profond

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans Laboratoire interdisciplinaire des sciences du numérique (Université Paris-Saclay, CNRS), sous la direction d'Isabelle GUYON, Professeure, le co-encadrement de Felix MOHR, Professeur et le co-encadrement de Hedi TABIA, Professeur.

Thèse soutenue à Paris-Saclay, le 19 décembre 2023, par

Haozhe SUN

Composition du jury

Membres du jury avec voix délibérative

Lambert SCHOMAKER Professor, University of Groningen	Président & Rapporteur
Amparo ALONSO-BETANZOS Professor, Universidade da Coruña	Rapporteur & Examinatrice
Vincent LEMAIRE Research Scientist, Orange Labs	Examinateur
Mehreen SAEED Research Associate, North Carolina State University	Examinatrice

THESE DE DOCTORAT

NNT : 2023UPASG090

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Modularité dans l'Apprentissage Profond
Mots clés : Apprentissage profond, intelligence artificielle, modularité

Résumé : L'objectif de cette thèse est de rendre l'apprentissage profond plus efficace en termes de ressources en appliquant le principe de modularité. La thèse comporte plusieurs contributions principales : une étude de la littérature sur la modularité dans l'apprentissage profond ; la conception d'OmniPrint et de Meta-Album, des outils qui facilitent l'étude de la modularité des données ; des études de cas examinant les effets de l'apprentissage épisodique, un exemple de modularité des données ; un mécanisme d'évaluation modulaire appelé LTU pour évaluer les risques en matière de protection de la vie privée ; et la méthode RRR pour réutiliser des modèles modulaires pré-entraînés afin d'en construire des versions plus compactes. La modularité, qui implique la décomposition d'une entité en sous-entités, est un concept répandu dans diverses disciplines. Cette thèse examine la modularité sur trois axes de l'apprentissage profond : les données, la tâche et le modèle. OmniPrint et Meta-Album facilitent de comparer les modèles modulaires et d'explorer les impacts de la modularité des données. LTU garantit la fiabilité de l'évaluation de la protection de la vie privée. RRR améliore l'efficacité de l'utilisation des modèles modulaires pré-entraînés. Collectivement, cette thèse fait le lien entre le principe de modularité et l'apprentissage profond et souligne ses avantages dans certains domaines de l'apprentissage profond, contribuant ainsi à une intelligence artificielle plus efficace en termes de ressources.

Title : Modularity in Deep Learning
Keywords : Deep learning, artificial intelligence, modularity

Abstract : This Ph.D. thesis is dedicated to enhancing the efficiency of Deep Learning by leveraging the principle of modularity. It contains several main contributions : a literature survey on modularity in Deep Learning ; the introduction of OmniPrint and Meta-Album, tools that facilitate the investigation of data modularity ; case studies examining the effects of episodic few-shot learning, an instance of data modularity ; a modular evaluation mechanism named LTU for assessing privacy risks ; and the method RRR for reusing pre-trained modular models to create more compact versions. Modularity, which involves decomposing an entity into sub-entities, is a prevalent concept across various disciplines. This thesis examines modularity across three axes of Deep Learning : data, task, and model. OmniPrint and Meta-Album assist in benchmarking modular models and exploring data modularity's impacts. LTU ensures the reliability of the privacy assessment. RRR significantly enhances the utilization efficiency of pre-trained modular models. Collectively, this thesis bridges the modularity principle with Deep Learning and underscores its advantages in selected fields of Deep Learning, contributing to more resource-efficient Artificial Intelligence.

Acknowledgments

I would like to express my sincere and deep gratitude to Isabelle Guyon for her invaluable guidance and experience. Throughout these years, she led me to top-level research and provided me with the opportunity to work with great people and engage with top platforms. I consider myself exceptionally fortunate to have received her endless inspiration and direction. The opportunities she offered me are deeply appreciated.

I am deeply indebted to Hedi Tabia and Felix Mohr. They provided me with a lot of guidance, opportunities, and help during the latter half of my Ph.D. journey, which was crucial in the successful completion of my doctoral studies. I would like to acknowledge the guidance of Anne Auger and Wei-Wei Tu during the beginning of my doctoral program. I would like to sincerely thank the jury members Lambert Schomaker, Amparo Alonso-Betanzos, Vincent Lemaire, and Mehreen Saeed for dedicating their time to reviewing and examining my Ph.D. thesis.

I am immensely thankful to my colleagues and collaborators, Dustin Carrión-Ojeda, Joseph Pedersen, Ihsan Ullah, Adrian El Baz, Romain Egele, Alexandre Heuillet, Birhanu Hailu Belay, Lisheng Sun, Adrien Pavao, Shuyu Dong, Tamon Nakano, Balthazar Donon, Alice Lacan, Mélina Verger, Zhen Xu, and Badr Youbi Idrissi. Our collaborations have resulted in many fruitful accomplishments during my Ph.D. thesis. Working with them has been a joy and a pleasure. I greatly value the time we have spent together and the insightful discussions we have shared.

I extend my heartfelt thanks to all my co-authors, Zhengying Liu, Rafael Muñoz-Gómez, Jiangnan Huang, Phan Anh Vu, Mike Huisman, Jan N. van Rijn, Joaquin Vanschoren, Frank Hutter, Wenwu Zhu, Sergio Escalera, Sebastien Treguer, Edesio Alcobaça, André C. P. L. F. Carvalho, Hong Chen, Fabio Ferreira, Henry Gouk, Chaoyu Guan, Timothy Hospedales, Shell Hu, Ekrem Öztürk, Xin Wang, Mahbubul Alam, Ahmed Farahat, Dipanjan Ghosh, Teresa Gonzalez Diaz, Chetan Gupta, Joël Roman Ky, Xian Yeow Lee, Xin Liu, Manh Hung Nguyen, Emmanuel Pintelas, Stefan Roth, Simone Schaub-Meyer, Lasitha Vidyaratne, Jiamin Wu, and Xiaotian Yin. Collaborating with them has been an enriching experience.

My gratitude extends to many members of the TAU team at the LISN laboratory (INRIA, CNRS), the Humania group, ChaLearn, the MetaDL technical crew, the Meta-Album technical crew (including the data owners/creators), Edouard Belval, and all those who contributed to TextRecognitionDataGenerator. I would like to thank ANR AI chair Humania ANR-19-CHIA-0022 and INRIA for funding my Ph.D. thesis. I extend my thanks to Lab-IA and TAU for providing computational resources and to Google for their donation of computing

cloud units.

This Ph.D. thesis marks the end of my higher education years. I am immensely grateful to my family and friends for their constant support along the way. Most importantly, I must express my deepest gratitude to my parents, who have shaped who I am. Their support during not just my Ph.D. but my entire life has been invaluable, especially in tough times. Their endless patience and backing, witnessing both my highs and lows, have been my pillar.

Summary

The current trend in Artificial Intelligence is to rely heavily on systems capable of learning from examples, such as Deep Learning models, a modern embodiment of artificial neural networks. While numerous applications have made it to market in recent years (including self-driving cars, automated assistants, booking services, chatbots, improvements in search engines, recommendations, advertising, and healthcare applications, to name a few), Deep Learning models are still notoriously hard to deploy. During the training phase, these models typically demand a massive number of training examples and substantial computational resources. On the other hand, during real-time utilization, trained models encounter challenges such as latency, storage constraints, and power consumption.

Modularity is a general principle present in many fields, such as biology, system design, computer science, and graph theory. It is the property of an entity, whereby it can be broken down into a number of sub-entities (referred to as modules). It offers attractive advantages, which include, among others, ease of conceptualization, interpretability, scalability, and module reusability. The goal of this Ph.D. thesis is to contribute to the field of low-resource Deep Learning. The modularity principle, which suggests that complex systems can be broken down into smaller components, serves as the guiding principle. In particular, we focus on reducing the computational resource requirements associated with Deep Learning and enhancing the reusability of existing resources.

The first contribution of this thesis is a literature survey [1] of the notion of modularity present in the field of Deep Learning. We discuss the definition of modularity and review the modularity principle present in Deep Learning around three axes: data, task, and model, which characterize the life cycle of Deep Learning. Data modularity refers to the observation or creation of data groups for various purposes. Task modularity refers to the decomposition of tasks into sub-tasks. Model modularity means that the architecture of a neural network system can be decomposed into modules. We describe different instantiations of the modularity principle, and we contextualize their advantages in different Deep Learning sub-fields.

We have made a series of contributions in the scope of data modularity. We introduce OmniPrint [2], a data synthesizer designed to provide full control over latent factors in the data generation process. Alongside, we introduce Meta-Album [3], a meta-dataset distinguished by its vast diversity in terms of data domains and sources. Both OmniPrint and Meta-Album are characterized by their rich data modularity information, including super-classes, extensive metadata, and varied domains. This information richness helps ex-

plore the effects of data modularity *e.g.,* the effects of using heterogeneous data episodes when training few-shot learning models. Additionally, both the OmniPrint and Meta-Album datasets lend themselves to benchmarking modular models for tasks such as few-shot learning, transfer learning, and meta-learning. In our research, we have used them in organizing few-shot learning challenges. Our subsequent post-challenge analyses [4, 5] investigated the effects of episodic training in the context of few-shot learning.

We have made one contribution in the scope of task modularity. We present a modular evaluation mechanism, termed LTU [6], for assessing the privacy risks of Deep Learning models. This evaluation breaks down into two parts: an Attacker and an Evaluator. While the Evaluator conducts leave-two-unlabeled evaluation rounds, the Attacker simulates privacy attacks.

We have also made one contribution in the scope of model modularity. Reusing models pre-trained on large datasets has become a common practice in computer vision and other Deep Learning fields. However, the high performance of these pre-trained models often comes with large sizes and high inference latency, making them less suitable for resource-constrained machines. To address this, we introduced an approach to reuse these pre-trained models, delivering faster inference speed without compromising on performance. Grounded in the "Reuse, Reduce, and Recycle" philosophy, our approach, RRR [7], employs reduction and recycling techniques. The results indicate that our refined model significantly enhances utilization efficiency while preserving the performance of the original.

In conclusion, this Ph.D. thesis is devoted to connecting the modularity principle and Deep Learning to make Deep Learning more resource-efficient. We investigated how modularity is present in Deep Learning and how to leverage it from different axes (data, task, model) of Deep Learning. We demonstrated the benefits of the modularity principle in some selected subjects of Deep Learning. These studies pave the way for more resource-efficient Artificial Intelligence.

Résumé

La tendance actuelle en Intelligence Artificielle est de s'appuyer fortement sur des systèmes capables d'apprendre à partir d'exemples, tels que les modèles de Deep Learning, une incarnation moderne des réseaux de neurones artificiels. Bien que de nombreuses applications aient été commercialisées ces dernières années (y compris les voitures autonomes, les assistants automatisés, les services de réservation, les chatbots, les améliorations dans les moteurs de recherche, les recommandations, la publicité et les applications de santé, pour n'en nommer que quelques-unes), les modèles de Deep Learning restent notoirement difficiles à déployer. Durant la phase d'apprentissage, ces modèles exigent typiquement un nombre massif d'exemples d'entraînement et d'importantes ressources computationnelles. D'autre part, lors de l'utilisation en temps réel, ces modèles rencontrent des défis tels que la latence, les contraintes de stockage et la consommation d'énergie.

La modularité est un principe général présent dans de nombreux domaines, tels que la biologie, la conception de systèmes, l'informatique et la théorie des graphes. C'est la propriété d'une entité, qui peut être décomposée en un certain nombre de sous-entités (appelées modules). Elle offre des avantages attrayants, qui incluent, entre autres, la facilité de conceptualisation, l'interprétabilité, la scalabilité et la réutilisabilité des modules. L'objectif de cette thèse de doctorat est de contribuer au domaine du Deep Learning à faibles ressources. Le principe de modularité, qui suggère que des systèmes complexes peuvent être décomposés en composants plus petits, sert de principe directeur. En particulier, nous nous concentrons sur la réduction des exigences en ressources computationnelles associées au Deep Learning et sur l'amélioration de la réutilisabilité des ressources existantes.

La première contribution de cette thèse est une revue de la littérature [1] sur la notion de modularité présente dans le domaine du Deep Learning. Nous discutons de la définition de la modularité et examinons le principe de modularité présent dans le Deep Learning autour de trois axes : les données, la tâche et le modèle, qui caractérisent le cycle de vie du Deep Learning. La modularité des données se réfère à l'observation ou à la création de groupes de données à diverses fins. La modularité des tâches se réfère à la décomposition des tâches en sous-tâches. La modularité des modèles signifie que l'architecture d'un système de réseau de neurones peut être décomposée en modules. Nous décrivons différentes incarnations du principe de modularité et contextualisons leurs avantages dans différents sous-domaines du Deep Learning.

Nous avons réalisé une série de contributions dans le domaine de la modularité des données. Nous introduisons OmniPrint [2], un synthétiseur de

données conçu pour offrir un contrôle total sur les facteurs latents dans le processus de génération de données. Parallèlement, nous introduisons Meta-Album [3], un méta-dataset caractérisé par sa grande diversité en termes de domaines et de sources de données. OmniPrint et Meta-Album se caractérisent tous les deux par leur riche information de modularité des données, incluant des super-classes, des métadonnées et des domaines variés. Cette richesse d'information aide à explorer les effets de la modularité des données, par exemple les effets de l'utilisation d'épisodes de données hétérogènes lors de l'entraînement de modèles d'apprentissage avec peu d'exemples. De plus, les données d'OmniPrint et de Meta-Album se prêtent au benchmarking de modèles modulaires pour des tâches telles que l'apprentissage avec peu d'exemples, l'apprentissage par transfert et le méta-apprentissage. Dans notre recherche, nous les avons utilisés pour organiser des compétitions d'apprentissage avec peu d'exemples. Nos analyses post-compétition [4, 5] ont étudié les effets de l'entraînement épisodique dans le contexte de l'apprentissage avec peu d'exemples.

Nous avons réalisé une contribution dans le domaine de la modularité des tâches. Nous présentons un mécanisme d'évaluation modulaire, appelé LTU [6], pour évaluer les risques de confidentialité des modèles de Deep Learning. Cette évaluation se décompose en deux parties : un Attaquant et un Évaluateur. Alors que l'Évaluateur conduit des tours d'évaluation LTU, l'Attaquant simule des attaques de confidentialité.

Nous avons également réalisé une contribution dans le domaine de la modularité des modèles. La réutilisation de modèles pré-entraînés sur de grands ensembles de données est devenue une pratique courante en vision par ordinateur et dans d'autres domaines du Deep Learning. Cependant, la haute performance de ces modèles pré-entraînés s'accompagne souvent de grandes tailles et d'une latence d'inférence élevée, les rendant moins adaptés aux machines à ressources limitées. Pour y remédier, nous avons introduit une approche pour réutiliser ces modèles pré-entraînés, offrant une vitesse d'inférence plus rapide sans compromettre les performances. Ancrée dans la philosophie « Réutiliser, Réduire et Recycler », notre approche, RRR [7], emploie des techniques de réduction et de recyclage. Les résultats indiquent que notre modèle raffiné améliore significativement l'efficacité de l'utilisation tout en préservant la performance du modèle original.

En conclusion, cette thèse de doctorat est consacrée à relier le principe de modularité et le Deep Learning pour rendre le Deep Learning plus efficient en termes de ressources. Nous avons étudié comment la modularité est présente dans le Deep Learning et comment l'exploiter à partir de différents axes (données, tâche, modèle) du Deep Learning. Nous avons démontré les avantages du principe de modularité dans certains sujets sélectionnés du Deep Learning. Ces études ouvrent la voie à une Intelligence Artificielle

plus efficiente en ressources.

Contents

List of Figures

List of Tables

List of Algorithms

List of papers published during this thesis

For a documentary purpose, we list papers that are co-authored by the author of this thesis and were published during this thesis.

1. **Haozhe Sun**, Wei-Wei Tu, and Isabelle Guyon. "OmniPrint: A Configurable Printed Character Synthesizer." In Proceedings of the Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks, edited by J. Vanschoren and S. Yeung, Vol. 1, 2021. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/38b3eff8baf56627478ec76a704e9b52-Paper-round1.pdf. https://doi.org/10.48550/arXiv.2201.06648.
2. **Haozhe Sun**, Isabelle Guyon, Felix Mohr, and Hedi Tabia. "RRR-Net: Reusing, Reducing, and Recycling a Deep Backbone Network." In 2023 International Joint Conference on Neural Networks (IJCNN), 1–9, 2023. https://doi.org/10.1109/IJCNN54540.2023.10191770.
3. **Haozhe Sun**, and Isabelle Guyon. "Modularity in Deep Learning: A Survey." In Intelligent Computing, edited by Kohei Arai, 561–95. Cham: Springer Nature Switzerland, 2023. https://doi.org/10.1007/978-3-031-37963-5_40.
4. Ihsan Ullah, Dustin Carrión-Ojeda, Sergio Escalera, Isabelle Guyon, Mike Huisman, Felix Mohr, Jan N. van Rijn, **Haozhe Sun**, Joaquin Vanschoren, and Phan Anh Vu. "Meta-Album: Multi-Domain Meta-Dataset for Few-Shot Image Classification." In Advances in Neural Information Processing Systems (NeurIPS), edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, 35:3232–47. Curran Associates, Inc., 2022. https://proceedings.neurips.cc/paper_files/paper/2022/file/1585da86b5a3c4fb15520a2b368205
Paper-Datasets_and_Benchmarks.pdf. https://doi.org/10.48550/arXiv.2302.08909.
5. Joseph Pedersen, Rafael Muñoz-Gómez, Jiangnan Huang, **Haozhe Sun**, Wei-Wei Tu, and Isabelle Guyon. "LTU Attacker for Membership Inference." Algorithms 15, no. 7 (2022). https://doi.org/10.3390/a15070254.
6. Adrian El Baz, Ihsan Ullah, Edesio Alcobaça, André C. P. L. F. Carvalho, Hong Chen, Fabio Ferreira, Henry Gouk, Chaoyu Guan, Isabelle Guyon, Timothy Hospedales, Shell Hu, Mike Huisman, Frank Hutter, Zhengying Liu, Felix Mohr, Ekrem Öztürk, Jan N. van Rijn, **Haozhe Sun**, Xin Wang, and Wenwu Zhu. "Lessons Learned from the NeurIPS 2021 MetaDL Challenge: Backbone Fine-Tuning without Episodic Meta-Learning Dominates for Few-Shot Learning Image Classification." In Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track, edited by Douwe Kiela, Marco Ciccone, and Barbara Caputo, 176:80–96. Proceedings of Machine Learning Research. PMLR, 2022. https://proceedings.mlr.press/v176/el-baz22a.html. https://doi.org/10.48550/arXiv.2206.08138.

7. Dustin Carrión-Ojeda, Mahbubul Alam, Sergio Escalera, Ahmed Farahat, Dipanjan Ghosh, Teresa Gonzalez Diaz, Chetan Gupta, Isabelle Guyon, Joël Roman Ky, Xian Yeow Lee, Xin Liu, Felix Mohr, Manh Hung Nguyen, Emmanuel Pintelas, Stefan Roth, Simone Schaub-Meyer, **Haozhe Sun**, Ihsan Ullah, Joaquin Vanschoren, Lasitha Vidyaratne, Jiamin Wu, and Xiaotian Yin. "NeurIPS'22 Cross-Domain MetaDL Challenge: Results and Lessons Learned." In Proceedings of the NeurIPS 2022 Competitions Track, edited by Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, 220:50–72. Proceedings of Machine Learning Research. PMLR, 2022. https://proceedings.mlr.press/v220/carojeda23a.html. https://inria.hal.science/hal-04198139/.

1 - The modularity principle

1.1 . Motivation

The modularity principle is omnipresent in many fields and disciplines. At its heart, modularity is about breaking down complexity into more manageable parts.

In both nature and society, many complex systems are built on modularity [13, 14]. Such systems are often composed of a large number of components. Recognizing and studying the modular components or subsystems can simplify the understanding of these complex systems. This is why modularity is foundational in system design and engineering [15, 16, 17, 18]. A modular approach makes systems more flexible, scalable, and easy to maintain. It also allows efficiently replacing, upgrading or reusing individual modules without disrupting the whole system.

Software development deeply adopts the principle of modularity [19, 20]. This is evident in programming languages, frameworks, and design patterns which often highlight modularity. Breaking software down into modules, such as functions, classes, or packages, improves clarity, allows for reuse, and simplifies maintenance. Moreover, modularity promotes teamwork in the sense that it enables developers to work on separate modules in parallel without much interference.

The principle of modularity is also present in mathematics because mathematical knowledge can be understood as having a modular structure [21, 22]. Mathematics is based on fundamental ideas that support more complex theories. Proven theorems act as modules for subsequent proofs. Once a concept is understood, it can be reused in other contexts. Mathematicians often break down complex problems into simpler sub-problems. This point of view simplifies computations and depicts properties with better clarity, capturing the beauty of mathematics.

Modularity is not just a human-engineered concept; it is also fundamental to biological systems. For example, in biology, modular construction is evident in animals like colonial cnidarians, bryozoans, and colonial ascidians. They are subdivided into repeated modules, each capable of acquiring, processing, and sharing resources [23]. Moreover, physical constraints and evolutionary pressures [24, 25] have made modularity as a foundational feature of biological brains [26, 27, 28, 29, 30, 31, 32, 33]. As evidenced in numerous studies, the cerebral cortex exhibits this modularity, partitioning itself into specialized areas responsible for distinct functions, such as processing various sensory signals or enabling reasoning [34]. This organization is not just functional but also structural: neuron wiring patterns within the cortex exhibit clustering, en-

suring neurons collaborating on a function are closely situated [35, 36]. Such modularity is exemplified in the visual cortex, where the processing of various visual perception properties is allocated to anatomically and functionally distinct regions operating autonomously [37, 38].

Similar to biological brains, modularity is present in graph theory when identifying communities [39, 40, 41, 42]. Vertices in graphs often cluster together to form these communities. Within a community, vertices are more tightly connected to each other than to vertices outside of it. Recognizing this modularity pattern is crucial for understanding the networks.

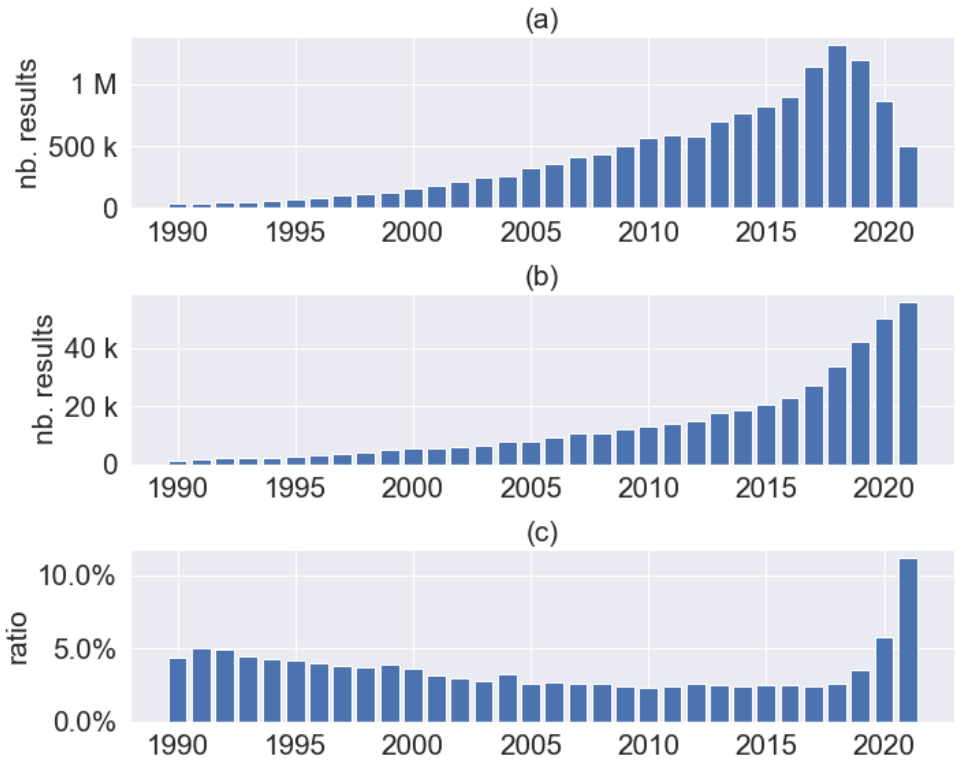


Figure 1.1 – **Trends in Publications on "Modular Deep Learning" from 1990 to 2021.** We conducted a search on Google Scholar to track the frequency of certain keywords in academic papers. The horizontal axis represents the publication year. (a) represents the total count of search results for the terms "deep learning" and "neural network". (b) represents the total count of search results for the terms "modular deep learning" and "modular neural network". (c) shows the proportion $\frac{(b)}{(a)}$, indicating the presence of "modular" terminology within the academic papers about "deep learning" and "neural network", as indexed by Google Scholar.

Back in the early years of Deep Learning research, around the end of the last century, the researchers began exploring the idea of introducing modu-

larity into neural networks [43, 44, 45, 46]. This interest has been revived in recent years [47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57].

To grasp the depth of this renewed interest, we used Google Scholar to count the number of related publications over the years. The findings are quite revealing. As shown in Figure 1.1, it is clear that there has been an increase in Deep Learning research papers referencing the term "modular". This trend indicates that the Deep Learning community is increasingly valuing the advantages of modularity, emphasizing its emergence as a compelling research direction.

1.2 . Definition of modularity

Modularity, as a general principle, is both a descriptive property and an organizational scheme. It serves as a way to represent entities such as data, tasks, and models, allowing for their conceptual or practical manipulation [58, 19, 20, 18].

Modularity is intuitively simple to understand as a concept. However, defining modularity is, in itself, a challenging problem. The notion of modularity is present in literature across many different fields such as biology [59, 60, 26, 27, 28, 34, 29, 30, 61, 31, 33, 38, 25, 24], complex systems [13, 14], mathematics [21, 22], system design [20, 15, 16, 17, 18], computer science [19, 32], graph theory [39, 40, 41, 42]. While many researchers have a strong intuition about what it means for an entity to be modular, there has yet to be a universal agreement on what defines modularity [62]. The same is true even within the field of neural networks.

> Modularity of neural networks is a bit like the notion of beauty in art: everyone agrees that it's important, but nobody can say exactly what it means.
>
> (Béna *et al.* [63])

Despite their common use, defining intuitive concepts is not always evident, as seen with terms like intelligence, complex system, among others [64, 65, 66, 67, 13, 68, 69]. After all, the definition is a matter of social conventions, different communities may have different conventions or requirements. There is no simple solution to determine a single definition to use in all cases [62]. However, defining the object of interest is a prerequisite to make progress in a scientific field.

In order to define the notion of modularity, we adopt the methodology of Legg *et al.* [64]: (1) we collect a list of definitions of modularity from the literature across different disciplines, and (2) we scan through them to find out common defining features. It is worth mentioning that, while many works discuss modularity, not all provide an explicit definition. Our list exclusively includes those that are clearly articulated. The exhaustive list is available in Appendix A. A selected subset of this list is presented below.

> Modularity can be defined as subdivision of a complex object into simpler objects. The subdivision is determined either by the structure or function of the object and its subparts.
>
> (Schmidt *et al.* [70])

> Modularity is the property of a system whereby it can be broken down into a number of relatively independent, replicable, and

composable subsystems (or modules).

(Amer *et al.* [52])

> Modularity as a system design principle is apprehended here as the extent to which processes can be decomposed into modules to be executed in parallel and/or in series.

(Modrak *et al.* [17])

From the list of modularity definitions in Appendix A, we identify a set of defining features: decomposition, cohesion, independence, functional specialization, combinability, and replicability. While decomposition relates to the overall entity, the other features are characteristics of the decomposed sub-entities (modules).

- **Decomposition** highlights that the original entity can be broken down into smaller components, known as sub-entities or modules.
- **Cohesion** indicates that elements within a module tend to have tighter relations (*e.g.,* connections, similarities) compared to elements spanning different modules.
- **Independence** suggests minimal or no causal effects [71, 72] between different modules. Each module can operate or change independently without influencing the others.
- **Functional specialization** means that modules are specialized to implement a particular functionality.
- **Combinability**, also termed as composability or recombinability, denotes the capability of modules (possibly from distinct original entities) to merge. This implies an interoperable interface (a communication protocol or a set of rules) that enables different modules to exchange information and collaborate seamlessly.
- **Replicability** means that once developed, modules can be replicated in a large number. Examples are the consistent structures in body parts of organisms (*e.g.,* cells, legs, fingers), and the duplication of basic units in electronics [70] (*e.g.,* transistors, microprocessors, memory chips, integrated circuits).

The features we have listed are not mutually exclusive. In fact, differentiating them sharply can often be challenging, as they frequently coexist. However, they each highlight distinct aspects. Decomposition acts as the foundation for the others, since the subsequent features relate to the properties of the decomposed modules. Cohesion presupposes the presence of relationships between the elements of the entity, highlighting the irregular distribution of these relationships. Independence underscores the limited interactions between separate modules. Functional specialization presupposes that

the entire entity has a specific function, with each module specialized to a particular sub-function. It's worth noting that not all entities have a defined function *e.g.,* a graph of vertices and edges may lack a defined purpose. Combinability points to the existence of a interoperable interface, allowing for the combination of modules, even those from different entities. Lastly, replicability stresses the ability to duplicate modules.

Some properties can emerge from the enumerated defining features, including the *reusability* and *replaceability* of modules. Decomposing an entity into modules paves the way for both reusing and replacing these modules. Features like independence, functional specialization, combinability, and replicability enhance the reusability of modules. Meanwhile, replaceability is a consequence of combinability. When an interoperable interface is in place, maintaining the entity becomes easier as faulty components can be substituted with new ones.

After examining the list of modularity definitions in Appendix A, we summarize the defining features mentioned in these definitions in Table 1.1. It is clear that the challenge in defining modularity arises from its association with numerous different features. Authors across diverse fields and communities typically retain different subsets of these features to label an entity as modular. Yet, there is one consistent feature in all these definitions: decomposition.

	[70]	[52]	[17]	[21]	[38]	[73]	[15]	[16]	[39]	[26]	[59]	[19]	[25]	[74]	[75]	[55]	[76]	[77]	[78]	[79]
Decomposition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cohesion	×	×	×	✓	✓	×	×	✓	✓	×	✓	✓	✓	×	×	×	×	×	×	×
Independence	×	✓	×	×	×	×	×	×	×	×	×	✓	×	×	✓	✓	✓	✓	✓	×
Functional specialization	×	×	×	×	×	×	×	×	×	×	×	×	✓	✓	×	×	×	×	×	×
Combinability	×	✓	×	×	×	✓	✓	×	×	×	×	×	×	×	×	×	×	✓	×	×
Replicability	×	✓	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

Table 1.1 – **Summary of common defining features from different definitions of modularity.** Rows denote the common features. Columns denote the reference for the modularity definitions. We see that "Decomposition" is the only feature required by all definitions of modularity.

In this thesis, we choose a general definition of modularity. Such a definition offers the benefit of maintaining its validity across various fields and communities. Drawing insights from Table 1.1, we adopt Definition 1.2.1 to define the modularity principle. This definition is foundational, it is the prerequisite for the other defining features mentioned above.

Definition 1.2.1 (Modularity)**.** Modularity is the property of an entity whereby it can be broken down into a number of sub-entities. Sub-entities are referred to as modules.

Definition 1.2.1 involves entities. In this thesis, we define entities as anything that can be represented by a set of atomic elements (indivisible ele-

ments) or as a system. Here, a system is understood as a set of atomic elements accompanied by relationships between them. This leads to Definition 1.2.2.

Definition 1.2.2 (Entity)**.** An entity is a 2-tuple $(\mathcal{E}, \mathcal{C})$, where $\mathcal{E}$ is a set of atomic elements a_i, $\mathcal{C}$ is a set of functions $\mathcal{R}_j$. $\forall j$, $\mathcal{R}_j : \mathcal{E} \times \mathcal{E} \to \mathcal{B}_j$, where the codomain $\mathcal{B}_j$ is an arbitrary set.

We assume that $\mathcal{E}$ is non-empty, as an empty $\mathcal{E}$ would mean the entity doesn't exist. However, $\mathcal{C}$ can be the empty set $\varnothing$. When $\mathcal{C} = \varnothing$, the entity is simply a set. If $\mathcal{C} \neq \varnothing$, the entity is a system. For all j, $\mathcal{R}_j$ is a function of two elements, termed a relationship. The codomain $\mathcal{B}_j$ determines the kind of relationships $\mathcal{R}_j$ can represent between atomic elements, which may include connections, similarities, distances, interactions, dependencies, or module composition constraints. A system can have multiple relationships defined over its atomic elements, with each relationship characterizing a different aspect of the entity. For instance, $\mathcal{R}_1$ with $\mathcal{B}_1 = \mathbb{R}$ can measure the connection's intensity or similarities between atomic elements. Another example of relationship is $\mathcal{R}_2$ with $\mathcal{B}_2 = \{0, 1\}$, introducing a module composition constraint. In this case, two atomic elements $a, a' \in \mathcal{E}$ can be put into the same module only if $\mathcal{R}_2(a, a') = 1$. The way relationships are defined is subjective and depends on the study's purposes.

Generally, a module $\mathcal{M}$ is a subset of the set of atomic elements, denoted as $\mathcal{M} \subseteq \mathcal{E}$. When relationships impose module composition constraints, as in the example above, a subset of atomic elements can qualify as a module only if all these atomic elements fulfill these constraints.

A relationship between modules can be defined from a relationship between atomic elements. For instance, if the codomain of the relationship between atomic elements is a subset of the set of real numbers $\mathbb{R}$, then the relationship between modules might be defined as *e.g.,* the minimum, maximum, or average relationships of atomic elements from the respective modules. This concept is closely related to the linkage criteria utilized in hierarchical clustering methods [80].

An entity $(\mathcal{E}, \mathcal{C})$ is modular if and only if $|\mathcal{E}| > 1$. By recursively applying Definition 1.2.1, a modular entity is one that can be decomposed into sub-entities. Each sub-entity can, in turn, be further decomposed into sub-sub-entities, and so on. This recursive decomposition continues until atomic elements (the smallest indivisible modules) are obtained.

Many modular entities exhibit a hierarchical structure [13]. In these structures, multiple modules from a lower hierarchical level combine to constitute a single module at a higher level. Modules in the lower level of the hierarchy are more fine-grained than those at the higher levels. Within a given hierarchical level, the decomposition can result in modules that either distinctly

partition the entity (known as a hard division) or share components with other modules (known as a soft division). The resulting modules from decomposition can either be homogeneous, where all modules are similar, or heterogeneous, with dissimilar modules.

> Since all systems are characterized by some degree of coupling (whether loose or tight) between components, and very few systems have components that are completely inseparable and cannot be recombined, almost all systems are, to some degree, modular.
>
> (Schilling [73])

Definition 1.2.1 turns out to be very general. Under this definition, almost anything can be deemed modular to some extent, except for those entities composed solely of one indivisible element.

The determination of what qualifies as an indivisible element is often subject to perspective and context. There can be instances where we establish a threshold beyond which further decomposition is no longer desirable, aligning with the purposes of the study. On the other hand, our perceptions of what is divisible can also be shaped by advancements in our knowledge. For instance, atoms were once believed to be the smallest indivisible units of matter. However, with the evolution of particle physics, we have discovered that atoms themselves can be broken down into smaller particles [81].

What makes modularity compelling are the properties of the modules. These properties offer a range of practical benefits. For example, the capability to reuse modules allows for efficient use of existing assets and resources [21, 46, 47, 51, 53, 70, 82, 83, 18, 84, 85, 86, 87, 88, 89, 90]; the ability to replace modules contributes to enhancing the maintainability of the entire entity [20, 82, 83]; the independence of modules provides flexibility for the whole entity [19, 76, 91, 92, 75].

Having adopted a general definition of modularity, it becomes essential to contextualize this principle within Deep Learning. The next chapter offers a perspective on how modularity is perceived in the context of Deep Learning. We will provide a taxonomy of modularity in Deep Learning and use it as a framework to review the existing literature.

1.3 . Thesis outline

The organization of this thesis manuscript is as follows.

Chapter 1 is the introduction chapter. It provides the background and motivation of modularity. It also presents and discusses our definition of modularity to establish the foundational framework of the entire thesis.

Chapter 2 discusses how the modularity principle is present in the field of Deep Learning. We view the modularity principle from three different axes of Deep Learning: data, task, and model, which characterize the life cycle of Deep Learning. Chapter 2 provides a literature survey of the modularity principle in Deep Learning around these three axes.

Chapter 3 presents our contributions in the scope of data modularity research. It features the development of OmniPrint, a data synthesizer, and Meta-Album, a meta-dataset. This chapter also presents cases studies that use these tools to investigate the effects of data modularity in the context of few-shot learning.

Chapter 4 presents our contribution in the scope of task modularity research. It features the design of a modular evaluation mechanism addressing the privacy concerns of Deep Learning models.

Chapter 5 presents our contribution in the scope of model modularity research. It features our exploration of techniques to efficiently reusing pre-trained models with the goal of creating compact, faster models without sacrificing performance. Grounded in the "Reuse, Reduce, and Recycle" principle, our study presents techniques for the efficient re-utilization of a pre-trained ResNet152 model.

Finally, Chapter 6 presents the conclusion and suggestions for future work.

2 - Survey on modularity in Deep Learning

As discussed in the previous chapter, modularity is a general principle present across a multitude of disciplines, including the field of Deep Learning. It refers to the idea that complex systems can be decomposed into smaller components that can be easily understood and manipulated. This principle has significant implications for the design and development of Deep Learning systems and has been the subject of research in recent years.

In this chapter, we seek to explore the modularity principle in the context of Deep Learning, examining it from three different yet related axes: data, task, and model. Each of these three axes represents a critical component of the life cycle of Deep Learning and plays a key role in the effectiveness and efficiency of Deep Learning.

This chapter discusses how the modularity principle is presented across these three axes and provides a literature survey. The organization of this survey chapter is illustrated in Figure 2.1. The majority of the material presented in this chapter has been the subject of publication [1].

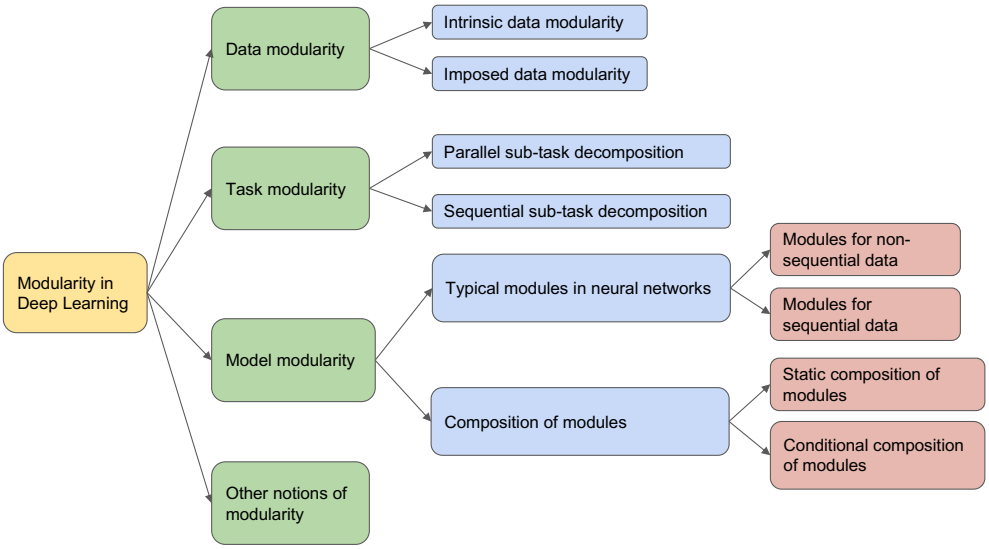


Figure 2.1 – **Organization of this survey chapter.** The first three subsections discuss how the modularity principle is instantiated in the three axes: data, task, and model architecture. We then cover other modularity notions for completeness.

2.1 . Data modularity

2.1.1 . Definition of data modularity

Data is an entity used to represent knowledge and information. In the context of machine learning and Deep Learning, it can take various forms *e.g.,* image, audio sound, and text. Data samples can be interpreted as points in a high dimensional space (fixed-length dense vectors) [93, 94, 95]. A collection of data samples is a dataset. Datasets can be used to train or test Deep Learning models, referred to as training or test datasets. In these scenarios, data is the input of Deep Learning models (neural networks) [96].

Data modularity is the observation or creation of data groups; it refers to how a dataset can be divided into different modules for various purposes. In the framework of Definition 1.2.2, the entity $(\mathcal{E}, \mathcal{C})$ can be defined to be the dataset. The choice of the set of atomic elements $\mathcal{E}$ depends on the point of view. The atomic elements could be defined to be the individual data samples, in which case the set of relationships $\mathcal{C}$ could include the distances or correlations between samples, enabling the definition of data sample clusters. This logic extends to the embedding of data samples. It should be noted that each data sample can indeed be further divided into smaller components *e.g.,* feature vectors of reduced length in the case of tabular data or image patches in the case of image data. The modularization could happen both at the sample dimension and the feature dimension. This is why the atomic elements could also be defined to be every single dimension of data samples, which are scalar numbers stored in physical computers (often represented by floating-point numbers). In this scenario, the set of relationships $\mathcal{C}$ should include a relationship for the module composition constraints, indicating which scalar numbers should be grouped to form a data sample.

We identify two types of data modularity: intrinsic data modularity and imposed data modularity. Intrinsic data modularity refers to the dataset divisions that are inherently present in the data at the time of its creation or generation. This includes divisions stemming from class labels and latent generative factors. Imposed data modularity refers to the dataset divisions that are introduced or imposed on the data after its creation or generation by a Deep Learning practitioner or data analyst. This includes the organization of data samples into random mini-batches or the assembly of data samples from a collection of classes which are selected randomly. We review the presence of intrinsic and imposed data modularity in the field of Deep Learning in Section 2.1.3 and Section 2.1.4, respectively.

2.1.2 . Advantages of data modularity

This section enumerates the advantages associated with data modularity. Bolded text references the features of the modularity principle introduced in Section 1.2. [1]

Decomposition

The division of a dataset into smaller modules can significantly enhance the ease with which the data can be organized and manipulated. It can offer several advantages in terms of data analysis, including the possibility to perform more targeted analyses on subsets of the data and the reduction in time and memory requirements. It can facilitate the identification of complex patterns and relationships between data samples that might otherwise be difficult to discern.

Cohesion

Data modularization can influence the training efficiency of learning machines. For example, curriculum learning [97] groups data samples based on a measure of difficulty or complexity of data samples, which allows presenting training data to a Deep Learning model in a progressively more challenging sequence so that the model can learn more efficiently. On the other hand, several recent works in the area of few-shot learning [98, 99] propose to decompose the overall training dataset into a number of smaller training datasets, referred to as episodes. Each episode contains a subset of the classes to be learned. Besides varying class subsets across episodes, it is also possible to vary latent generative factors, ensuring greater similarity of samples within an episode than between them [2].

Functional specialization

Some algorithms leverage the concept of data modularity to process data. For example, Qiao *et al.* [100] divides the training samples to heterogeneous subsets, each of which is processed by a different network. This modular strategy capitalizes on the divide-and-conquer principle.

Combinability

Data modules from different datasets can be combined and repurposed to create new datasets or meta-datasets [101, 3]. This not only facilitates the creation of more challenging datasets but also enables the simulation of more realistic scenarios that may involve diversified data sources.

1. We only list the modularity features for which relevant examples or scenarios have been identified in the context of data modularity.

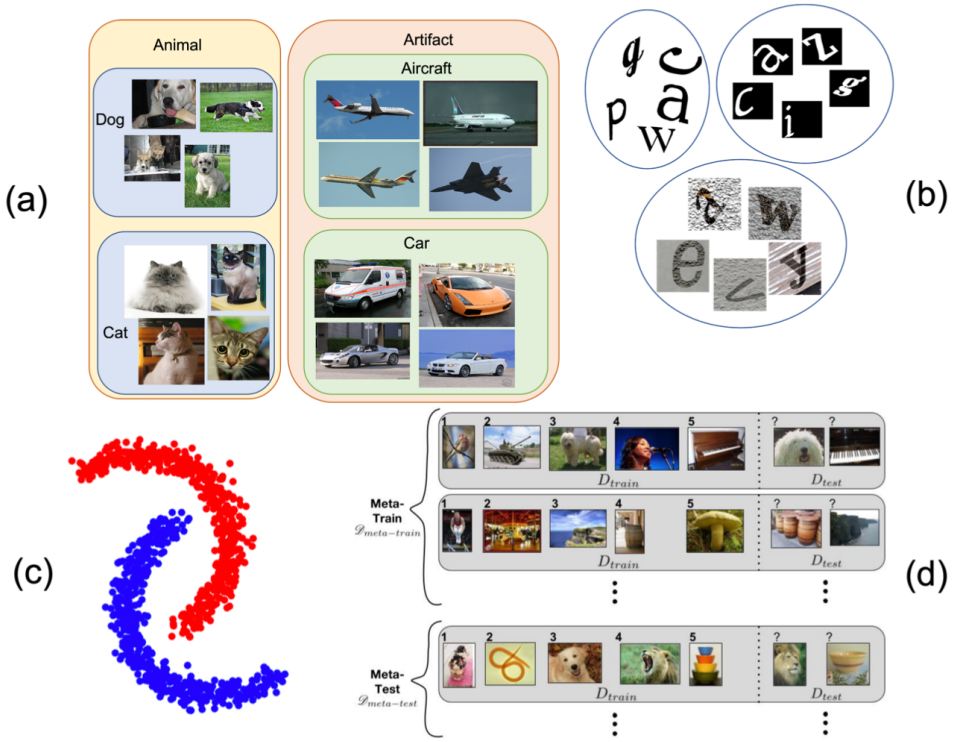


Figure 2.2 – **Illustration of data modularity.** (a) intrinsic data modularity based on super-classes, images, and class hierarchy in ImageNet [8]; (b) intrinsic data modularity based on styles characterized by a set of metadata, the upper-left circle contains black-on-white characters, the upper-right circle contains white-on-black characters, the lower circle contains characters with natural foreground and background, all characters are drawn from the same set of classes (small-case Latin characters), these three circles illustrate the division of a character dataset based on its metadata; (c) intrinsic manifolds in the form of a moon dataset, where each data manifold can be considered as a module; (d) few-shot learning episodes, reprinted from [9]. (a), (b) and (c) are examples of intrinsic data modularity, (d) is an example of imposed data modularity.

2.1.3 . Intrinsic data modularity

Intrinsic data modularity refers to the dataset divisions that are inherently present in the data at the time of its creation or generation.

Any supervised learning datasets can be divided according to the classes (labels); data points belonging to the same class are supposed to be close to each other in a hidden space, which allows for solutions of classification algorithms. Classes sharing common semantics can be further grouped to form super-classes. For example, ImageNet [8] has a class hierarchy (see Figure 2.2 (a)) which is used by Meta-Dataset [101]. Omniglot dataset [85] and OmniPrint datasets [2] contain character images organized in scripts, each script

(super-class) contains several characters (classes); Meta-Album dataset [3] is a meta-dataset including 40 datasets, where each dataset can be considered as a super-class. The super-classes provide information about class similarity, allowing splitting datasets according to the semantics [102].

In addition to the classes or super-classes, data points can also be grouped by one or several metadata such as time, location, and gender. Such metadata is available with the Exif data of photos. The OmniPrint data synthesizer generates data together with a comprehensive set of metadata, including font, background, foreground, margin size, shear angle, rotation angle, etc. [2] (see Figure 2.2 (b)). The NORB dataset collected stereo image pairs of 50 uniform-colored toys under 36 angles, 9 azimuths, and 6 lighting conditions, where the angles, azimuths, and lighting conditions serve as the metadata [103].

Some datasets contain intrinsic clusters in the high-dimensional feature space. Such intrinsic clusters can stem from the underlying data generative process, where latent categorical variables determine the natural groups of data. An illustrative example is a Gaussian Mixture distribution where data points are assumed to be generated from a mixture of a finite number of Gaussian distributions with unknown parameters [104]. Some datasets have intrinsic manifolds; an illustrative example is the moons dataset as shown in Figure 2.2 (c), where the two manifolds interlace while preserving an identifiable division, each manifold can be considered as a module. Both of the above examples fall into the category of data clustering. When data samples are interconnected in the form of a graph [105, 106], this is called graph partitioning. One question which arises is how to determine the optimal clustering of a dataset. Luxburg *et al.* [107] argue that there are no optimal domain-independent clustering algorithms and that clustering should always be studied in the context of its end-use.

Multi-modal Deep Learning aims to build models that can process and relate information from multiple modalities. Here the modality refers to the way in which something happens or is experienced *e.g.,* data in the form of image, text, audio [108]. Multi-modal datasets fall into the category of intrinsic data modularity in the sense that the data in each modality can be considered a module. For example, VQA v2.0 dataset [109] consists of open-ended questions about images; SpeakingFaces dataset [110] consists of aligned thermal and visual spectra image streams of fully-framed faces synchronized with audio recordings of each subject speaking.

2.1.4 . Imposed data modularity

Imposed data modularity refers to the dataset divisions that are introduced or imposed on the data after its creation or generation by a Deep Learning practitioner or data analyst.

When training Deep Learning models [96], human practitioners usually

divide the whole training dataset into mini-batches, which can be seen as a kind of imposed data modularity. The gradient is computed using one mini-batch of data for each parameter update; one training epoch means passing through all the mini-batches. This iterative learning regime is called stochastic gradient descent [111]. Mini-batches reduce the memory requirement for backpropagation, which makes training large Deep Learning models possible. On the other hand, batch size also influences learning behavior. Smith *et al.* [112] showed that the benefits of decaying the learning rate could be obtained by instead increasing the training batch size. Keskar *et al.* [113] showed that learning with large batch sizes usually gives worse generalization performance.

Instead of using a sequence of mini-batches sampled uniformly at random from the entire training dataset, curriculum learning [97] uses non-uniform sampling of mini-batches such that the mini-batch sequence exhibits an increasing level of difficulty. A related concept is active learning [114], which assumes that different data points in a dataset have different values for the current model update; it tries to select the data points with the highest value to construct the actual training set.

In few-shot learning [98, 99], the model performance is usually tested on few-shot episodes. Few-shot episodes are typically formed by drawing N classes from the class pool and K data samples for each selected class, called N-way-K-shot episodes [98, 99] (Figure 2.2 (d)).

Data augmentation is a way to generate more training data by applying transformations to existing data [115]. The transformed versions of the same data point can be seen as a module. Some transformations, such as rotation and translation, form a group structure [116]. The effect of such data augmentation can be understood as averaging over the orbits of the group that keeps the data distribution approximately invariant and leads to variance reduction [117].

In addition to splitting the dataset into subsets of samples, each data sample can be split into subdivisions of features, referred to as feature partitioning. A dataset can be represented as a matrix where each row represents one data sample; each column represents one feature dimension. It can then be divided along the sample and feature dimensions. Schmidt *et al.* [70] process each feature partition with a different model. For image classification tasks, input images can be split into small patches that can be processed in parallel [118, 119].

2.2 . Task modularity

2.2.1 . Definition of task modularity

Deep Learning models are utilized to solve a wide range of tasks, ranging from the classification of entities to the generation of realistic images. Solving a task boils down to achieving a corresponding objective. It's worth noting that a task simply defines an objective; the definition of a task is separate from the means used to accomplish it. In the context of Deep Learning, an objective is usually modeled by an explicit differentiable objective function, commonly referred to as a loss function, which facilitates end-to-end training. This viewpoint can be extended to any task, even if the objective function is implicit and lacks a differentiable form. For instance, the task of "purchasing a cup of tea" may be defined by an indicator function that imposes a penalty if tea cannot be purchased and a bonus if it can.

Tasks in Deep Learning are often linked with data, but they are different from data. Different tasks can be defined using the same dataset. For instance, the MNIST dataset [120] can be utilized for image classification benchmarking [121] or pixel sequence classification benchmarking [122, 91]. Similarly, the OmniPrint-meta[1-5] datasets [2] can be employed for few-shot learning benchmarking or domain adaptation benchmarking.

Task modularity, also known as sub-task decomposition, refers to the fact that a task can be broken down into a number of sub-tasks. In the framework of Definition 1.2.2, the entity $(\mathcal{E}, \mathcal{C})$ is the task in question, which may be subject to decomposition into sub-tasks. These sub-tasks, if they cannot be further decomposed, are regarded as atomic elements. The relationships $\mathcal{C}$ can include dependencies between sub-tasks *e.g.,* when some sub-tasks rely on the outputs of others as their inputs.

Task modularity can be categorized into two regimes: parallel decomposition and sequential decomposition, as illustrated in Figure 2.3. Parallel decomposition means that the sub-tasks can be executed simultaneously, whereas sequential decomposition means that sub-tasks need to be executed in a specific order, with some sub-tasks dependent on the completion of preceding ones. It is common in practice for these two regimes to be combined. For example, a sub-task from a sequential decomposition might be further decomposed into parallel sub-tasks, resulting in a directed acyclic graph workflow.

To provide concrete examples, consider the task of building a house. This task can be decomposed into several sub-tasks *e.g.,* laying the foundation, constructing the walls, and installing the roof. These sub-tasks exhibit a sequential relationship because the walls cannot be constructed until the foundation is laid, and the roof cannot be installed until the walls are constructed. On the other hand, the sub-task of constructing the walls can be further de-

composed into parallel sub-tasks, such as building the front wall, back wall, and side walls, as these can be constructed concurrently. We review the presence of parallel and sequential sub-task decomposition in the field of Deep Learning in Section 2.2.3 and Section 2.2.4, respectively.

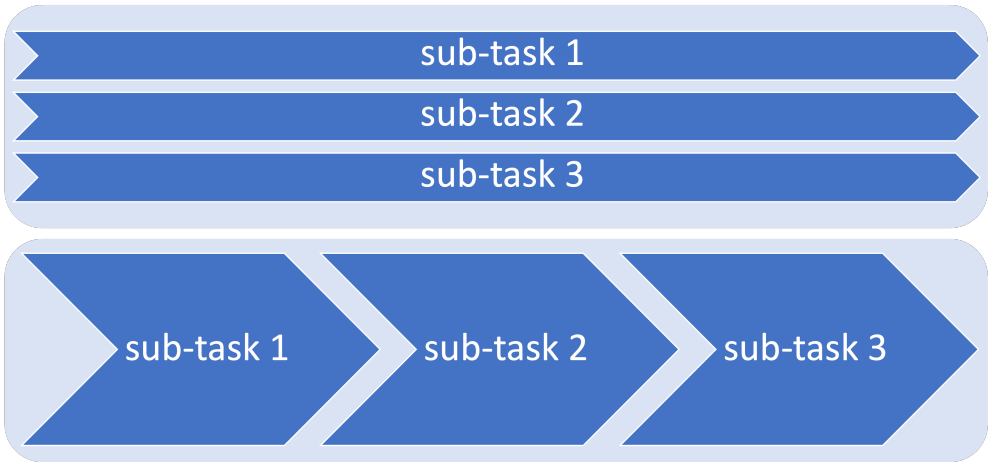


Figure 2.3 – **Illustration of sub-task decomposition.** The upper figure illustrates the parallel decomposition of a task. The lower figure illustrates the sequential decomposition of a task.

2.2.2 . Advantages of task modularity

This section enumerates the advantages associated with task modularity. Bolded text references the features of the modularity principle introduced in Section 1.2. [2]

Decomposition

Sub-task decomposition involves breaking down a complex task into smaller, more manageable sub-tasks. It facilitates conceptualization because it allows the identification and isolation of individual sub-tasks, each with its own requirements and constraints. It also facilitate the identification of shared patterns across sub-tasks.

On the other hand, sub-task decomposition facilitates problem-solving following the divide-and-conquer principle [123, 100, 44, 124]. Each sub-task is solved individually. Once the sub-tasks have been solved, the solutions are then combined to yield a solution to the original complex task.

2. We only list the modularity features for which relevant examples or scenarios have been identified in the context of task modularity.

Independence

Sub-task decomposition improves the flexibility and maintainability of a task, thereby decreasing the cost associated with problem-solving. In situations where the overall task cannot be accomplished, sub-task decomposition offers the advantage of enabling a targeted diagnosis of the failed sub-tasks [55], thereby saving resources for the other sub-tasks.

Sub-task decomposition can promote clarity and integrity. A well-defined decomposition simplifies resource allocation, enabling targeted allocation of specific computing resources or personnel to sub-tasks. In scenarios where information security is paramount, such as in the design of evaluation mechanisms [6], establishing independent sub-tasks can prevent data leakage, thereby safeguarding fairness and integrity.

Functional specialization

Sub-task decomposition facilitates the integration of expert knowledge, which can further facilitate the problem-solving. Expert knowledge refers to the knowledge from human experts in a particular field or the assumptions that are already known before the problem-solving process. On one hand, the way to divide the complex task can be guided by expert knowledge. On the other hand, by decomposing a complex task into sub-tasks, experts can more easily apply their specialized knowledge to each individual sub-task.

Reusability

Sub-task decomposition can promote reuse if the sub-tasks are designed to be combinable and reusable. In such cases, solutions to sub-tasks may be transferable to other tasks [125, 126, 127, 128, 129, 130]. This contributes to the overall efficiency because the need to develop new solutions from scratch for each new task is reduced.

2.2.3 . Parallel sub-task decomposition

A parallel sub-task decomposition is called homogeneous if the decomposed sub-tasks are similar. One typical example is dividing a multi-class classification problem into multiple smaller classification problems [58]. Given a neural network trained to perform a multi-class classification problem, Csordás *et al.* [53] use parameter masks to identify subsets of parameters solely responsible for individual classes on their own. Kim *et al.* [131] learn to split a neural network into a tree structure to handle different subsets of classes. They assume that different classes use different features, and this tree-structured neural network design guarantees that the deeper layers do not overlap features between class subsets. Pan *et al.* [82, 83] and Kingetsu *et al.* [132] decompose a multi-class classification model into reusable, replaceable and combinable modules, where each module is a binary classifier. Such modules

can be recombined without retraining to obtain a new multi-class classifier. These methods can be useful in situations where the classes to be classified frequently change. Abbas *et al.* [133] use transfer learning and class decomposition to improve the performance of medical image classification. Such sub-task decomposability is an implicit prerequisite of the model editing problem [134, 135, 136, 137, 138]. Model editing aims to modify a specific sub-task learned by a trained neural network without damaging model performance on other inputs, *e.g.,* it aims to patch the mistake of the model for a particular sample. If the task cannot be decomposed into disentangled sub-tasks, then model editing cannot be achieved.

A parallel sub-task decomposition is termed heterogeneous if the decomposed sub-tasks are dissimilar; such decomposition is usually problem-dependent and requires expert knowledge of the task at hand. Belay *et al.* [139] decompose the recognition task of Amharic characters into a vowel recognition task and a consonant recognition task to reduce overall task complexity. Cao *et al.* [140] decompose the full self-attention into question-wide and passage-wide self-attentions to speed up inference for question answering tasks. Ding *et al.* [141] divide the facial recognition task into multiple facial component recognition tasks. Zhou *et al.* [142] decompose the neural network learning process into structure learning and parameter learning, allowing for the automatic learning of equivariance from data. Gatys *et al.* [143] break down the natural image synthesis task into two components: content and style. This separation enables the combinatorial recombination of content and style to produce new images.

Parallel sub-task decomposition allows sub-tasks to iterate and collaborate with one another. This is exemplified in some Deep Learning methods that leverage a cooperative division of sub-tasks. For instance, in Generative Adversarial Networks (GANs) [144, 145, 146, 147, 148, 149], the data generation task is split between a generator, which creates data, and a discriminator, responsible for evaluating its authenticity. This dual setup transforms the generation process into a continuous feedback loop, with each component challenging the other to improve over time. The paradigm of teacher-student neural networks [150] further illustrates this cooperative interaction. It views the learning process as a collaborative effort between the guiding teacher and the learning student. Deep reinforcement learning brings one example with the Actor-Critic approach [151, 152, 153, 154]. Here, two sub-tasks work in tandem: the actor takes actions in the environment, while the critic evaluates these actions. Based on the feedback from the critic, the actor refines its policy to optimize its future actions, creating a cycle of action and evaluation. Continual learning with deep replay buffer [155] offers another example. As the learner processes new data, the replay buffer reintroduces samples from previous experience, acting like a memory mechanism. This ensures the model revis-

its and reinforces old knowledge even as it acquires new knowledge.

2.2.4 . Sequential sub-task decomposition

Sequential sub-task decomposition reflects the sequential pipeline of the task. A simple example is the division of a machine learning task into a preprocessing stage (data cleaning and normalization) and a model inference stage [156].

In reinforcement learning, a complex task can usually be decomposed [128] into a sequence of sub-tasks or steps. An illustrative example is to imagine that the task of manufacturing an artifact Z requires purchasing the raw material X, forging X to produce parts Y, and then assembling the parts Y into the end product Z. Both X and Y can take different values independently ($X \in \{x_1, x_2, x_3, ...\}, Y \in \{y_1, y_2, y_3, ...\}$). Different values of X and Y can be recombined, which forms a combinatorial number of possible scenarios to learn. This pipeline can be factorized into three stages: (1) raw material purchase, (2) forging to produce parts, and (3) assembling of parts. Reinforcement learning agents would learn more efficiently if the learning happens at the granularity of the factorized stages instead of the overall task [130]. Furthermore, such a factorization enables the independence of credit assignment [71]; the failure of the overall task can be traced back to the problematic stages, while the other stages can remain untouched. For example, if the raw material is of bad quality, then the purchase sub-task needs to be improved; the forging sub-task and the assembling sub-task do not need to be changed [55].

The sequential pipeline is omnipresent in practical applications *e.g.,* optical character recognition (OCR) [157, 158, 159, 160], natural language processing (NLP) [161]. When facing a multi-script (multi-language) recognition task, the pipeline can consist of a script identification stage and a script-specific recognition stage [162, 163], which decouples the domain classifier and the domain-specific solver. The text-in-the-wild recognition task [158] usually consists of decoupled text detector (to localize the bounding box of the text) and recognizer (recognize the text from the bounding box) [158]. Traditional OCR methods also decompose the word recognition task into a character segmentation task and a character recognition task [164, 165, 166, 167]. Traditional NLP pipeline includes sentence segmentation, word tokenization, part-of-speech tagging, lemmatization, filtering stop words, and dependency parsing [161]. In bioinformatics, the scientific workflow (data manipulations and transformations) groups similar or strongly coupled workflow steps into modules to facilitate understanding and reuse [18].

2.3 . Model modularity

2.3.1 . Definition of model modularity

Model modularity denotes the presence of sub-entities (referred to as modules) within the architecture of a neural network system. A neural network system may be comprised of a single neural network or a set of neural networks which interact with each other.

In the framework of Definition 1.2.2, the entity $(\mathcal{E}, \mathcal{C})$ is the neural network system under consideration. The choice of the set of atomic elements $\mathcal{E}$ depends on the point of view. The atomic elements could be defined to be the individual neurons within an artificial neural network, in which case the relationships could be defined to be the dependencies between neurons along the feedforward pass. It is worth noting that neurons can also be broken down into smaller components *e.g.,* scalar parameters. Indeed several works in the Deep Learning literature [53, 63, 168, 132] have defined modules via parameter masking, indicating that modularization of neural networks can be performed at the granularity of scalar parameters. In consequence, the atomic elements could also be defined to be the scalar parameters within a neural network system. In this case, the set of relationships $\mathcal{C}$ would need to include a relationship concerning the module composition constraints. This relationship would serve to specify the organizational arrangement of each scalar parameter along the feedforward pass within the computational graph of the neural networks.

Model modularity is different from task modularity. A task defines an objective to be attained, task modularity focuses on decomposing the objective into sub-objectives. Model modularity focuses on the architecture of the neural network system, it decomposes the solution into sub-solutions.

2.3.2 . Advantages of model modularity

This section enumerates the advantages associated with model modularity. Bolded text references the features of the modularity principle introduced in Section 1.2. [3]

Replicability

Model modularity provides ease of conceptual design and implementation. For example, modern neural networks consist of repeated layer/block patterns (modules). Such examples include fully-connected neural networks [96], vanilla convolutional neural networks, ResNets [10, 169], Inception [170] and models searched by Neural Architecture Search (NAS) [171, 172]. The design with homogeneous modules allows for a more concise description of the model

3. We only list the modularity features for which relevant examples or scenarios have been identified in the context of model modularity.

architecture in the sense of Kolmogorov complexity (short description length) [173, 174]. For example, instead of specifying how each primitive operation (*e.g.,* sum, product, concatenation) interacts in a computational graph, the model can be described as a collection of modules that interact with each other [58]. The standardization of such neural network building blocks (fully-connected layers, convolutional layers) also enabled the development of highly optimized hardware and software ecosystems for fast computation [175, 176, 177, 178, 179].

Modular neural network systems also facilitate model scaling in two ways. (1) Modular models like fully-connected models and ResNet can be scaled up (or down) by simply stacking more (or less) modules to increase (or decrease) the model capacity to fit larger (or smaller) datasets [10]. (2) Modular methods based on sparsely activated Mixture-of-Experts [48] decouple computation cost from model size. They allow drastically increasing the model capacity without increasing compute cost because only a small fraction of the model is evaluated on each forward pass [180, 48, 181, 182, 183, 184]. The extreme example of these sparsely activated models is Switch Transformer [185] which contains 1.6 trillion parameters, pushing the competition of large model sizes [186, 187] to the next level.

Functional specialization

Together with task modularity, model modularity offers ease of expert knowledge integration [47, 144, 152, 155, 139] and interpretability [50, 84, 124, 188]. Interpretability can have different forms. For example, each neural network module could be assigned a specific interpretable sub-task. On the other hand, selective module evaluation provides insights on how different samples/tasks are related [45, 48, 47, 51] in the context of conditional computation [189].

Reusability

The model decomposition into modules promotes reusability and knowledge transfer [190]. Though each neural network is typically trained to perform a specific task, its (decomposed) modules could be shared across tasks if appropriate mechanisms promote such reusability. The simplest example would be the classical fine-tuning paradigm of large pretrained models [191, 102, 192, 193]. This paradigm typically freezes the pretrained model and only retrains its last classification layer to adapt it to the downstream task. Pretrained models are typically pretrained on large datasets [194, 195, 196]. The large amount and diversity of training data make pretrained models' intermediate features reusable for other downstream tasks. More recently, the finer-grained reusability of neural network systems has attracted the attention of researchers. Such methods assume that the tasks share underlying

patterns and keep an inventory of reusable modules (each module is a small neural network) [47, 50, 51, 56]. Each module learns different facets (latent factors or atomic skills) of the knowledge required to solve each task. The selective/sparse use and dynamic reassembling/recombination of these modules can promote sample efficiency [84] and combinatorial generalization [47, 197, 51, 198].

Combinatorial generalization is also known as compositional generalization, "infinite use of finite means" [199], and systematic generalization. It aims to generalize to unseen compositions of known functions/factors/words [200, 201, 202, 203, 125, 204, 129], it is the ability to systematically recombine previously learned elements to map new inputs made up from these elements to their correct output [205]. For example, new sentences consist of new compositions of a known set of words. Combinatorial generalization is argued to be important to achieve human-like generalization [68, 84, 206, 207, 67, 208, 132, 77, 85, 209, 210, 211]. Learning different facets of knowledge with different modules in a reusable way could be one solution to combinatorial generalization. Modular systems have been shown effective for combinatorial generalization [212] in various fields *e.g.,* natural language processing [213, 84, 67, 208, 214], visual question answering [47, 197, 215], object recognition [129, 85, 87, 204], and robotics [51, 127, 216, 88].

Independence

The modularization of neural network systems promotes knowledge retention. If different knowledge is localized into different modules, targeted knowledge updates and troubleshooting [132, 82, 83] will be possible. This can alleviate gradient interference of different tasks [217, 218, 219] and catastrophic forgetting [56, 220, 221, 222, 223, 224, 51, 225, 226, 227, 228].

2.3.3 . Typical modules in Deep Learning models

This section reviews some typical modules in the Deep Learning literature.

Almost all systems are modular to some degree [73], neural network systems can almost always be decomposed into subsystems (modules) [229] following different points of view. More specifically, they usually consist of a hierarchical structure in which a module of a higher hierarchy level is made of modules of a lower hierarchy level. The elementary layer of modern neural networks (*e.g.,* fully-connected layer, convolutional layer) can be seen as a module on its own. On the other hand, any neural network as a whole can also be considered as a module *e.g.,* in the context of ensemble [230], Mixture-of-Experts [45], and Generative Adversarial Networks (GAN) [144]. Some literature [53, 63, 168, 132] define modules as sub-neural networks where part of the parameters are masked out (set to 0). In these cases, overlapping modules can be obtained when the masks overlap.

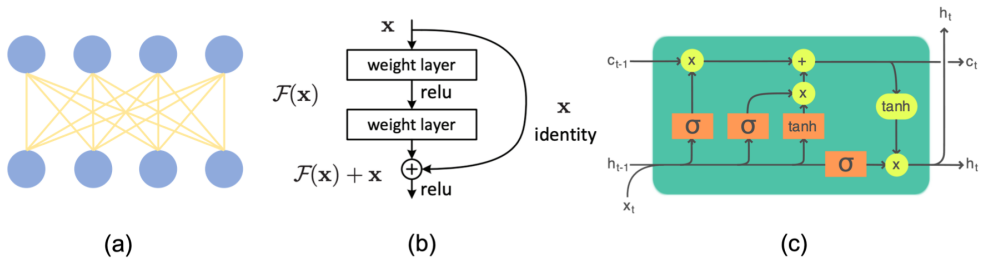


Figure 2.4 – **Examples of a module.** (a) a fully-connected layer; (b) a basic ResNet module, reprinted from [10]; (c) an LSTM module, reprinted from [11].

2.3.3.1 Modules for non-sequential data

Fully-connected layers (Figure 2.4 (a)) imitate the connections between neurons in biological neural networks but connect every input neuron to every output neuron [96]. In practice, a fully-connected layer is implemented as a matrix multiplication between input data and learnable parameters. Convolutional layers introduce the inductive bias of translation equivariance. Conceptually, a convolutional layer (with a single output channel) can be obtained from a fully-connected layer by enforcing local connectivity and parameter sharing [96]. Local connectivity means that each neuron only connects to a subset of neurons of the previous layer; parameter sharing means that the same learnable parameters are used across receptive fields. In practice, a convolutional layer is implemented as a collection of kernels/filters shifted over the input data [175, 176]. Each kernel performs a dot product between input data and learnable parameters. Depending on the number of dimensions over which kernels are shifted, a convolutional layer is termed *e.g.,* 1D, 2D, 3D. 2D convolutional layers are widely used in computer vision tasks [231, 95]. Locally connected layers are similar to convolutional layers except that they remove the constraint of parameter sharing (across kernels). It helps if one wants to impose local receptive fields while there is no reason to think each local kernel should be the same [96]. Low-rank locally connected layers relax spatial equivariance and provide a trade-off between locally connected layers and convolutional layers. The kernel applied at each position is constructed as a linear combination of a basis set of kernels with spatially varying combining weights. Varying the number of basis kernels allows controlling the degree of relaxation of spatial equivariance [232]. Standard convolutional layers offer translation equivariance; a line of research focuses on generalizing this to other equivariances (rotation, reflection), referred to as group convolutional layers [233, 234, 235, 236, 237, 238, 239, 240]. On the other hand, depthwise separable convolutional layers [241, 242, 243] factorize a standard convolutional layer into a depthwise convolutional layer and a pointwise convolutional layer, which reduces model size and computation.

Multiple layers can be grouped into a building block (a module of a higher hierarchy level). Such examples include the building blocks of ResNet [10], Inception [170, 244], ResNeXt [169], Wide ResNet [245]. Inception [170, 244] has parallel kernels of multiple sizes within each block and merges their results to extract information at varying scales. Inception also includes several techniques to reduce computation cost *e.g.,* factorizing large kernels into smaller kernels and using 1×1 convolution to reduce dimensionality. A ResNet block [10] (Figure 2.4 (b)) contains a sequence of convolutional layers; it adds a skip-connection (also known as residual connection, identity mapping) from the beginning to the end of the block to alleviate vanishing gradients. Many variants of the ResNet block have been proposed. For example, Wide ResNet [245] increases the block width; ResNeXt [169] aggregates parallel paths within each block.

The block design could be automatically searched instead of handcrafted. In order to narrow down the model search space, some Neural Architecture Search methods [172, 246, 171, 247] automatically search the optimal design pattern for a block (also known as a cell) while fixing the block composition scheme (also known as meta-architecture). Once the block design patterns are searched, the full model is instantiated by repeating the searched blocks following the predefined block composition scheme. For example, NAS-Bench-101 [247] defines the block search space as all possible directed acyclic graphs on V vertices ($V \leqslant 7$) while limiting the maximum number of edges to 9.

McNeely-White *et al.* [248] report that the features learned by Inception and ResNet are almost linear transformations of each other, even though these two architectures have a remarkable difference in the architectural design philosophy. This result explains why the two architectures usually perform similarly and highlights the importance of training data. This result is corroborated by Bouchacourt *et al.* [249], who argue that invariance generally stems from the data itself rather than from architectural bias.

2.3.3.2 Modules for sequential data

When the input data is sequential *e.g.,* time series, text, audio, video, Recurrent Neural Networks (RNN) [250] come into play. The RNN module processes the sequential data one at a time; the output (also known as the hidden state) of the RNN module at the previous time step is recursively fed back to the RNN module, which allows it to aggregate information across different time steps. The vanilla RNN module suffers from short-term memory issues; it cannot effectively preserve information over long sequences. To overcome this issue, gated recurrent unit (GRU) [251] and long short-term memory (LSTM) [252] module use gates to control which information should be stored or forgotten in the memory, which allows better preservation of long-term information. In GRU and LSTM modules, gates are neural networks with

trainable parameters. While GRU modules are faster to train than LSTM modules, their performance comparison varies depending on the scenario. GRU surpasses LSTM in long text and small dataset scenarios while LSTM outperforms GRU in other scenarios [253].

Contrary to RNN, GRU, and LSTM, which process sequential data one at a time, self-attention layers [254] process the data sequence in parallel. For each data point in a data sequence (*e.g.,* each time step of a time series), a self-attention layer creates three transformed versions, referred to as query vector, key vector, and value vector, through linear transformations. Between each pair of data points, the dot product between the query vector and the key vector of the pair reflects how much those two data points are related within the sequence. These dot products are then normalized and combined with the corresponding value vectors to get the new representation of each data point in the sequence. An enhanced version of self-attention layers is multi-head self-attention layers, which extract different versions of query vector, key vector, and value vector for each data point. Multi-head self-attention layers improve performance by capturing more diverse representations. A transformer block combines multi-head self-attention layers, fully-connected layers, normalization layers, and skip-connections. Models built upon transformer blocks have achieved state-of-the-art performance in a wide range of tasks such as natural language processing [255] and speech synthesis [256]. Transformer models can be applied to image modality by transforming each input image into a sequence of small image patches [119]. Despite the lack of image-specific inductive bias (translation equivariance, locality), vision transformers can achieve state-of-the-art performance when combined with a large amount of training data [119, 257, 258].

2.3.4 . Composition of modules

Section 2.3.3 presents typical modules in the literature. Section 2.3.4 discusses how to organize these modules to form a model (or a module of a higher hierarchy level).

2.3.4.1 Static composition of modules

Static composition means that the composed structure does not vary with input; the same structure is used for all input samples or tasks.

One straightforward way to compose modules is sequential concatenation (Figure 2.5 (a)). It implies that multiple (typically homogeneous) modules are sequentially concatenated into a chain to form a model, where a module's output is the next module's input. Examples of sequential concatenation include fully-connected models [96] and ResNet models [10]. This composition scheme typically does not assume an explicit sub-task decomposition; the chain of concatenated modules can instead be seen as a series of information

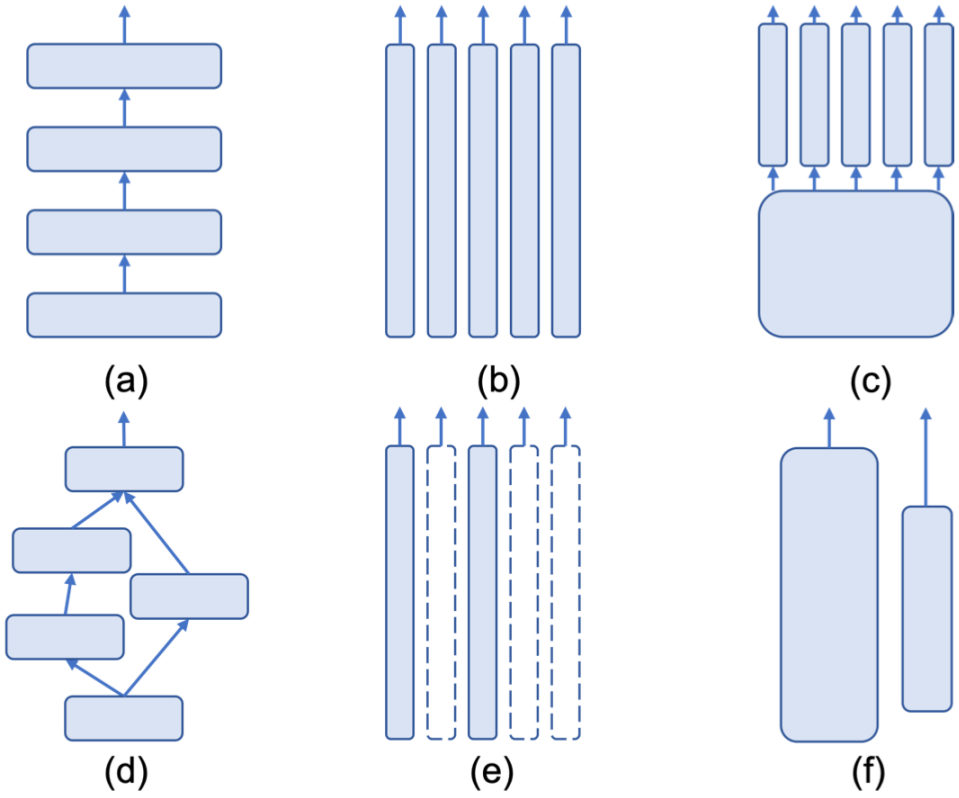


Figure 2.5 – **Illustration of module composition.** (a) Sequential concatenation. (b) Ensembling. (c) Tree-structure composition. (d) General Directed Acyclic Graph. (e) Conditional composition. (f) Cooperation composition.

extraction steps [102, 259, 260], extracted features transition from low-level to high-level.

Ensembling composition [230, 261, 262], on the other hand, organizes modules in a parallel manner (Figure 2.5 (b)). The principle of ensembling is to aggregate (*e.g.,* averaging) the results of multiple modules (weaker learners) to obtain a more robust prediction. The rationale is that different modules are expected to provide complementary and diverse views of input data. Each module's data is processed independently without relying on the other modules at inference time. The regularization method Dropout [263], which randomly deactivates neurons during training, can be seen as an implicit ensemble method of overlapping modules.

Sequential composition and parallel composition can be combined, *e.g.,* in the form of a tree structure (Figure 2.5 (c)). A typical scenario of tree-structure composition is a model with a shared feature extractor and multiple task-specific heads [264, 265]. All the above composition schemes are special cases of DAG (Directed Acyclic Graph, Figure 2.5 (d)). The general DAG composition scheme is typically found in models searched by Neural Architecture Search [266,

267, 46].

Cooperation composition (Figure 2.5 (f)) often coexists with task modularity, it suggests that each module is a standalone neural network with a specific purpose. These networks collaborate during training or inference, forming a system of multiple distinct neural networks. Unlike ensembling composition, modules within the cooperation composition are typically heterogeneous. Such examples include siamese networks [268, 269, 270], Generative Adversarial Networks (GAN) [144, 145], and deep reinforcement learning methods guided by the Actor-Critic approach [151, 152, 153, 154]. Continual learning methods that rely on a generative replay buffer [155] or incorporate new modules to enhance model capacity for future tasks [220, 221, 222, 56] also exemplify cooperation composition.

2.3.4.2 Conditional composition of modules

Conditional composition (Figure 2.5 (e)) is complementary to static composition in the sense that the composed modules are selectively (conditionally, sparsely, or dynamically) activated (used or evaluated) for each particular input. The input conditioning can happen at the granularity of individual sample [47, 50, 45] as well as task [84, 219, 168, 271, 124]. In the literature, this paradigm is also termed conditional computation [189, 272].

The idea of conditional computation can be traced back to Mixture-of-Experts (MoE) introduced in the last century. An MoE is a system composed of multiple separate neural networks (modules), each of which learns to handle a sub-task of the overall task [124, 273] *e.g.,* a subset of the complete training dataset. A gating network computes the probability of assigning each example to each module [45, 274] or a sparse weighted combination of modules [48]. Two issues of MoE are module collapse [48, 50, 206] and shrinking batch size [48], both of which are related to the balance of module utilization. Module collapse means under-utilization of modules or lack of module diversity. Due to the self-reinforcing behavior of the gating network during training, premature modules may be selected and thus trained even more. The gating network may end up converging to always selecting a small subset of modules while the other modules are never used. Shrinking batch size means the batch size is reduced for each conditionally activated module. Large batch sizes are necessary for modern hardware to make efficient inferences because they alleviate the cost of data transfers [48].

MoE can be generalized to *e.g.,* stacked MoE [275, 50, 276, 49, 277] or hierarchical MoE [48, 278] (Figure 2.6). Eigen *et al.* [275] first explored stacked MoE; they introduced the idea of using multiple MoE with their own gating networks. In order to train stacked MoE, Kirsch *et al.* [50] use generalized Viterbi Expectation-Maximization algorithm, Rosenbaum *et al.* [276] employ a multi-agent reinforcement learning algorithm, Fernando *et al.* [49] use a ge-

netic algorithm. MoE systems do not always have explicit gating networks; for instance, Fernando *et al.* [49] rely on the results of the genetic algorithm to decide the module routing scheme.

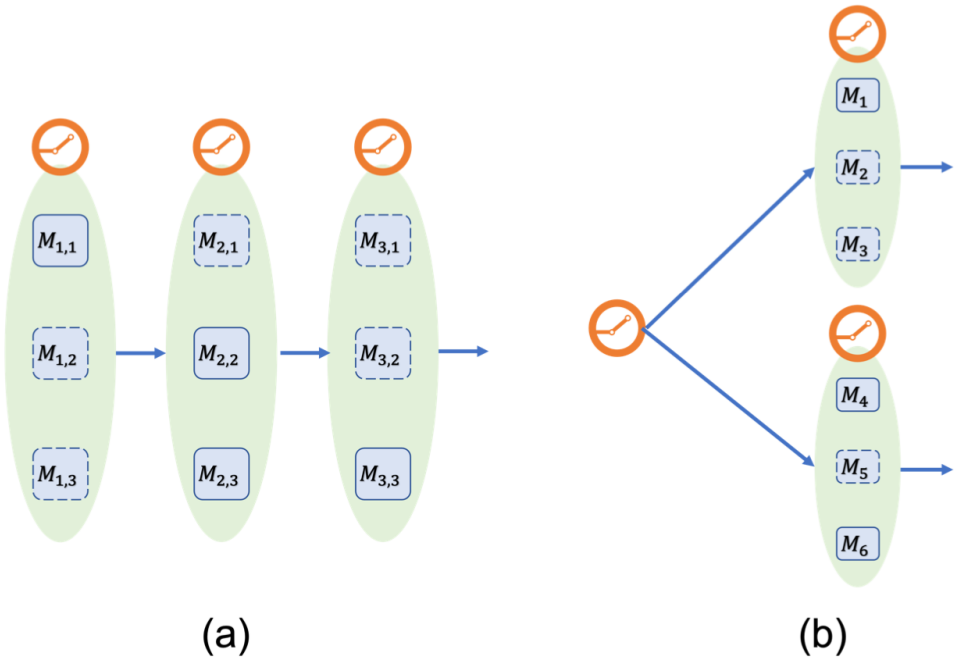


Figure 2.6 – **Extension of Mixture-of-Experts (MoE).** (a) A stacked MoE, which stacks multiple MoE layers into a chain. (b) A hierarchical MoE, where a primary gating network chooses a sparse weighted combination of "modules", each of which is an MoE with its own gating network.

Inspired by MoE, some Deep Learning methods keep an inventory of reusable specialized modules that can be conditionally reassembled for each input. This approach has been advocated to promote knowledge transfer, sample efficiency, and generalization. For example, in visual question answering, Neural Module Networks [47, 279, 197] dynamically reassemble modules into a neural network to locate the attention (region of interest) on the questioned image. The question's parsing guides the reassembling process so that the reassembled model reflects the structure and semantics of the question. For this particular task, the compositionality of modules comes from the compositionality of visual attention. Following the question's syntax, the reassembled modules sequentially modify the attention onto the questioned image. For example, the module associated with the word "cat" locates the image region containing a cat, and the module associated with the word "above" shifts up the attention. Zhang *et al.* [280] investigated adding new abilities to a generic network by directly transplanting the module corresponding to the new ability, dubbed network transplanting.

Some work relies on the hypothesis that the tasks at hand share some commonalities *i.e.,* hidden factors are shared across tasks. Each hidden factor can be learned by a separate module from the module inventory for transfer learning and meta-learning. For example, Alet *et al.* [51] use simulated annealing to meta-learn an inventory of modules reusable across tasks to achieve combinatorial generalization. The parameters of an inventory of modules are optimized during meta-training; the trained modules are reassembled during the meta-test with an optional parameter fine-tuning process. They demonstrated the utility of their method for robotics tasks. Ponti *et al.* [84] assume that each task is associated with a subset of latent discrete skills from a skill inventory. They try to generalize more systematically to new tasks by disentangling and recombining different facets of knowledge. More precisely, they jointly learn a skill-specific parameter vector for each latent skill and a binary task-skill allocation matrix. For each new task, the new model's parameter vector is created as the average of the skill-specific parameter vectors corresponding to the skills present in the new task (in addition to a shared base parameter vector).

The conditional composition scheme also has other forms. For example, Teerapittayanon *et al.* [281] save computation on easy input data via early exiting; deeper layers will be skipped if the intermediate feature's prediction confidence passes a predefined threshold. Fuengfusin *et al.* [282] train models whose layers can be removed at inference time without significantly reducing the performance to allow adaptive accuracy-latency trade-off. Similarly, Yu *et al.* [283] train models which are executable at customizable widths (the number of channels in a convolutional layer). Xiong *et al.* [284] sparsely activate convolutional kernels within each layer for each particular input sample, which provides an example of the conditional composition of overlapping modules.

2.4 . Other notions of modularity in Deep Learning

There remain some other notions of modularity in the Deep Learning literature.

In graph theory, the term "modularity" refers to a measure commonly used in community detection. It measures the density of connections within a community compared to between communities [39]. This measure can be applied to graph clustering problems in the form of modularity optimization [285, 286, 287, 288]. Inspired by this measure, Filan *et al.* [54] investigate the parameter clustering pattern that emerged from the training of a neural network. They view a neural network as an undirected weighted graph (edge weights are the absolute value of network parameters) and apply spectral clustering on the obtained graph. They observe that some neural networks trained on image classification tasks have some clustering properties of their parameters: edge weights are stronger within one cluster than between clusters. Watanabe *et al.* [289] have obtained similar results. Béna *et al.* [63] adapted the graph-theoretic modularity measure to define structural modularity and define functional specialization through three heuristic measures. The functional specialization can be intuitively understood as the extent to which a sub-network can do a sub-task independently. To investigate the relationship between structural and functional modularity, they design a scenario where a model with two parallel modules (with an adjustable number of interconnections) is used to predict whether the parity of the two digits is the same or different. They show that enforcing structural modularity via sparse connectivity between two communicating modules does lead to functional specialization of the modules. However, this phenomenon only happens at extreme levels of sparsity. With even a moderate number of interconnections, the modules become functionally entangled. Mittal *et al.* [206] observed that weighted combinations of parallel modules with a good module specialization are good in terms of the overall system performance, however end-to-end training itself is not enough to achieve a good module specialization.

The term "modularity" is related to the notion of independence in some literature. For example, Galanti *et al.* [76] use modularity to refer to the ability of hypernetworks [290] to learn a different function for each input instance. A line of research has been carried out on learning disentangled representation. Intuitively, disentangled representation aims to reverse the underlying data generating process and retrieve its latent factors into the learned representation (Figure 2.7). One of the desirable properties of a disentangled representation [291, 292, 78] is "modularity". In this context, a modular representation is a representation where each dimension of the representation conveys information about at most one latent generative factor.

Moreno-Muñoz *et al.* [293] introduce modular Gaussian processes for trans-

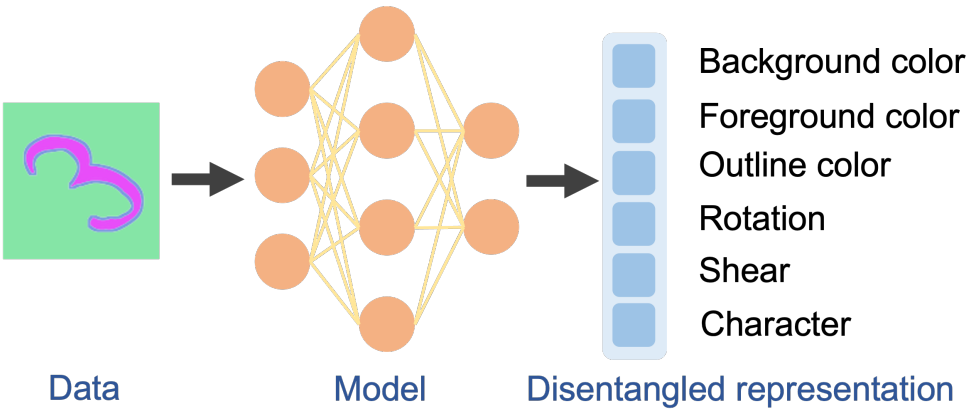


Figure 2.7 – **Illustration of a disentangled representation.**

fer learning. They build a dictionary of Gaussian process modules where they train each module on a subset of data. In some literature, some plug-in modules have been proposed to enable a specific functionality to a neural network. For example, the STN module [294] helps learn various invariances; the test-time adjustment module [295] replaces the model's last classification layer with template matching to promote domain generalization.

2.5 . Conclusion

In this chapter, we introduced a taxonomy of modularity in Deep Learning around three axes: data, task, and model. Each of these axes represents a component of the Deep Learning life cycle. For each axis, we provided the definition and contextualization of modularity:

- **Data modularity**: This concept refers to observing or creating data groups, where a dataset is segmented into modules.
- **Task modularity**: This concept refers to the breakdown of a task into sub-tasks, also known as sub-task decomposition.
- **Model modularity**: This concept refers to the presence of sub-entities, termed modules, within the architecture of a neural network system. This system may comprise a single neural network or multiple neural networks that interact with each other.

Using this taxonomy as the guiding framework, we conducted a review of the Deep Learning literature, exploring how modularity has been approached and implemented in the literature. Our exploration shed light on the advantages of modularity: ease of conceptualization and manipulation, reduction of the cost of problem-solving, enhanced reusability of existing assets, prevention of interference among individual components, and scalability, among others. These advantages align with the features of modularity, such as decomposition, independence, functional specialization, replicability, which were discussed in the previous chapter (see Section 1.2). Our survey reveals that modularity is pervasive in Deep Learning.

In the chapters that follow, we will organize the presentation of our contributions around these three axes. Chapter 3 presents our research contributions in the scope of data modularity, Chapter 4 presents our research contribution in the scope of task modularity, and Chapter 5 presents our research contribution in the scope of model modularity.

3 - Contributions in the scope of data modularity

Benchmarks and datasets have been fostering progress in Deep Learning. We have contributed to this progress by designing and creating two tools for data and benchmarks: (1) OmniPrint, a data synthesizer, and (2) Meta-Album, a meta-dataset for image classification. OmniPrint [2] was published at NeurIPS 2021 Datasets and Benchmarks Track, and Meta-Album [3] was published at NeurIPS 2022 Datasets and Benchmarks Track. Both tools exhibit data modularity properties in the form of super-classes, rich metadata, and diverse domains, thereby helping the investigation of data modularity effects, for example, on model training efficiency. Moreover, OmniPrint allows full control over the data-generating process's latent factors, while Meta-Album provides a large diversity in terms of data domain and sources. They lend themselves to benchmarking modular models for a wide variety of tasks, including few-shot learning, transfer learning, and meta-learning.

We also present case studies on episodic few-shot learning as a contribution in the scope of data modularity. We interpret few-shot learning data episodes as an instance of data modularity. More specifically, we view the data episodes in few-shot learning as data modules, where data samples are organized in a particular manner (see Section 2.1). We aim to investigate the effects of such modularization of data samples.

This chapter is structured into three sections: Section 3.1 is dedicated to OmniPrint, Section 3.2 is dedicated to Meta-Album, and Section 3.3 presents case studies on episodic few-shot learning. Most of the content of this chapter has been the subject of publications, including two papers accepted at NeurIPS data and benchmark track [2, 3] and two papers for the post-analysis of NeurIPS competitions [4, 5].

3.1 . OmniPrint: generation of data with intrinsic modularity

One of the most popular benchmarks is MNIST [120], which is used all over the world in tutorials, textbooks, and classes. Many variants of MNIST have been created including: Omniglot [85]. This dataset includes characters from many different scripts. Among machine learning techniques using such benchmark datasets, Deep Learning techniques are known to be very data hungry. Thus, while there is an increasing number of available datasets, there is a need for larger ones. But, collecting and labeling data is time consuming and expensive, and systematically varying environment conditions [296] is difficult and necessarily limited. Therefore, resorting to artificially generated data is useful to drive the research in Deep Learning. This motivated us to create *OmniPrint*, as an extension to Omniglot, geared to the generation of an unlimited amount of printed characters. Some sample images synthesized by OmniPrint are shown in Figure 3.1.

Figure 3.1 – Sample images synthesized by OmniPrint.

Of all Deep Learning problems, we direct our attention to classification and regression problems in which a vector $\mathbf{y}$ (discrete or continuous labels) must be predicted from a real-valued input vector $\mathbf{x}$ of observations (in the case of OmniPrint, an image of a printed character). Additionally, data are plagued by nuisance variables $\mathbf{z}$, another vector of discrete or continuous labels, called *metadata* or *covariates*. $\mathbf{z}$ is a form of intrinsic data modularity as it segments the dataset, and this segmentation is inherently present in the data at the time of its generation. In the problem at hand, $\mathbf{z}$ may include various character distortions, such as shear, rotation, line width variations, and changes in background. Using capital letters for random variable and lowercase for their associated realizations, a data generating process supported by OmniPrint consists in three steps:

$$\mathbf{z} \sim \mathbb{P}(\mathbf{Z}) \tag{3.1}$$
$$\mathbf{y} \sim \mathbb{P}(\mathbf{Y}|\mathbf{Z}) \tag{3.2}$$
$$\mathbf{x} \sim \mathbb{P}(\mathbf{X}|\mathbf{Z}, \mathbf{Y}) \tag{3.3}$$

Oftentimes, $\mathbf{Z}$ and $\mathbf{Y}$ are independent, so $\mathbb{P}(\mathbf{Y}|\mathbf{Z}) = \mathbb{P}(\mathbf{Y})$. This type of data generating process is encountered in many scenarios such as image, video, sound, and text applications (in which objects or concepts are target values $\mathbf{y}$ to be predicted from percepts $\mathbf{x}$); medical diagnoses of genetic disease (for which $\mathbf{x}$ is a phenotype and $\mathbf{y}$ a genotype); analytical chemistry (for which $\mathbf{x}$ may be chromatograms, mass spectra, or other instrument measurements, and $\mathbf{y}$ compounds to be identified), etc. Advancements achieved through the utilization of OmniPrint as a benchmarking tool for Deep Learning systems could stimulate progress in other related scenarios.

Casting the problem in such a generic way allows researchers to target a variety of Deep Learning research topics. Indeed, character images provide excellent benchmarks for Deep Learning problems because of their relative simplicity, their visual nature, while opening the door to high-impact real-life applications. No publicly available data synthesizer fully suits our purposes: generating realistic quality images $\mathbf{x}$ of small sizes (to allow fast experimentation) for a wide variety of characters $\mathbf{y}$ (to study extreme number of classes), and wide variety of conditions parameterized by $\mathbf{z}$ (to study invariance to realistic distortions). A conjunction of technical features is required to meet our specifications: pre-rasterization manipulation of anchor points; post-rasterization distortions; natural background and seamless blending; foreground filling; anti-aliasing rendering; importing new fonts and styles.

Modern fonts (*e.g.,* TrueType or OpenType) are made of straight line segments and quadratic Bezier curves, connecting anchor points. Thus it is easy to modify characters by moving anchor points. This allows users to perform vectors-space pre-rasterization geometric transforms (rotation, shear, etc.) as well as distortions (*e.g.,* modifying the length of ascenders of descenders), without incurring aberrations due to aliasing, when transformations are done in pixel space (post-rasterization). To our knowledge, OmniPrint is the first text image synthesizer geared toward ML research, supporting pre-rasterization transforms. This allows Omniprint to imitate handwritten characters, to some degree.

The generative process of OmniPrint is illustrated in Figure 3.2. Briefly, here are some highlights of the pipeline:

1. **Parameter configuration file:** We support both TrueType or OpenType font files. Style parameters include rotation angle, shear, stroke width, foreground, text outline and other transformation-specific parameters.

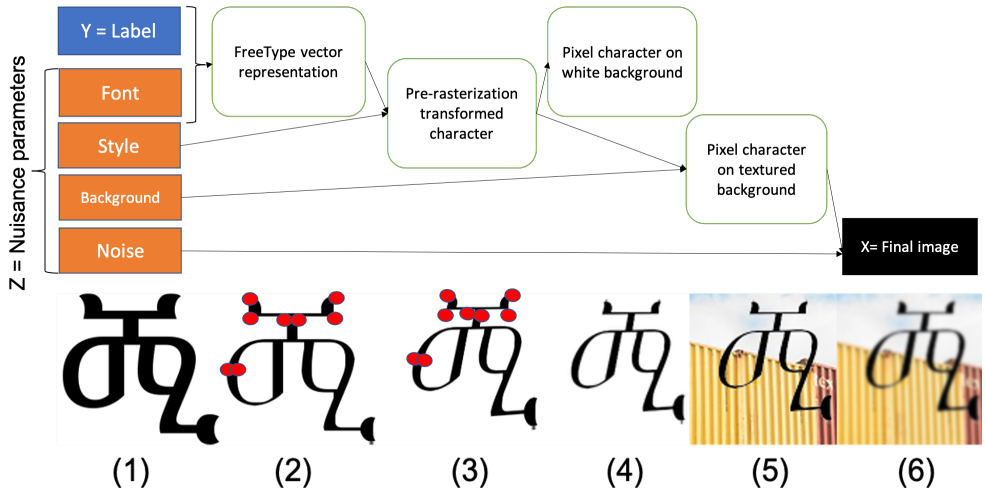


Figure 3.2 – **Basic character image generative process in OmniPrint.** The generative process produces images $\mathbf{X}$ as a function of $\mathbf{Y}$ (label or character class) and $\mathbf{Z}$ (nuisance parameter). Only a subset of anchor point (red dots) are shown in steps (2) and (3). A subset of nuisance parameters are chosen for illustration. $\mathbf{Z}$ represents a form of intrinsic data modularity *i.e.,* data samples with similar $\mathbf{Z}$ values can be organized into clusters.

2. **FreeType vector representation:** The chosen text, font and style parameters are used as the input to the FreeType rasterization engine [297].
3. **Pre-rasterization transformed character:** FreeType also performs all the pre-rasterization (vector-based) transformations, which include linear transforms, stroke width variation, random elastic transformation and variation of character proportion. The RGB bitmaps output by FreeType are called the foreground layer.
4. **Pixel character on white background:** Post-rasterization transformations are applied to the foreground layer. The foreground layer is kept at high resolution at this stage to avoid introducing artifacts. The RGB image is then resized to the desired size with anti-aliasing techniques. The resizing pipeline consists of three steps: (1) applying Gaussian filter to smooth the image; (2) reducing the image by integer times; (3) resizing the image using Lanczos resampling. The second step of the resizing pipeline is an optimization technique proposed by the PIL library [298].
5. **Pixel character on textured background:** The resized foreground layer is then pasted onto the background at the desired position.
6. **Final image:** Some other post-rasterization transformations may be applied after adding the background *e.g.,* Gaussian blur of the whole image. Before outputting the synthesized text image, the image mode can be changed if needed (*e.g.,* changed to grayscale or binary images).

Labels $\mathbf{Y}$ (isolated characters of text) and nuisance parameters $\mathbf{Z}$ (font, style, background, etc.) are output together with image $\mathbf{X}$. $\mathbf{Z}$ serve as "metadata" to help diagnose learning algorithms. The role of $\mathbf{Y}$ and (a subset of) $\mathbf{Z}$ may be exchanged to create a variety of classification problems (*e.g.,* classifying alphabets or fonts), or regression problems (*e.g.,* predicting rotation angles or shear).

We rely on the Unicode 9.0.0 standard [299], which consists of a total of 128172 characters from more than 135 scripts, to identify characters by "code point". A code point is an integer, which represents a single character or part of a character; some code points can be chained to represent a single character *e.g.,* the small Latin letter o with circumflex ô can be either represented by a single code point 244 or a sequence of code points (111, 770), where 111 corresponds to the small Latin letter o, 770 means combining circumflex accent. In this work, we use NFC normalized code points [300] to ensure that each character is uniquely identified.

We have included 27 scripts: Arabic, Armenian, Balinese, Bengali, Chinese, Devanagari, Ethiopic, Georgian, Greek, Gujarati, Hebrew, Hiragana, Katakana, Khmer, Korean, Lao, Latin, Mongolian, Myanmar, N'Ko, Oriya, Russian, Sinhala, Tamil, Telugu, Thai, Tibetan. For each of these scripts, we manually selected characters. Besides skipping unassigned code points, control characters, incomplete characters, we also filtered Diacritics, tone marks, repetition marks, vocalic modification, subjoined consonants, cantillation marks, etc. For Chinese and Korean characters, we included the most commonly used ones. In total, we have selected 12729 characters from 27 scripts (and some special symbols) and 935 fonts.

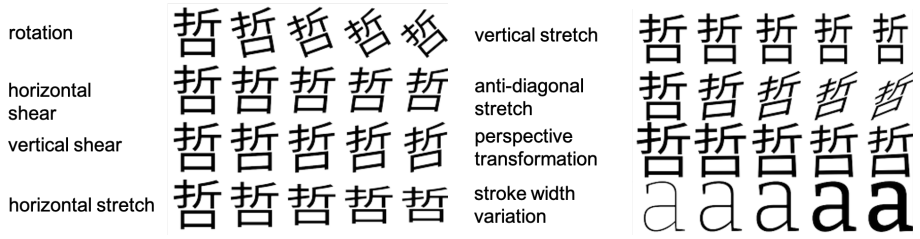


Figure 3.3 – **Some available transformations.**

We are interested in all "label-preserving" transformations on text images as well as their compositions. A transformation is said to be label-preserving if applying it does not alter the semantic meaning of the text image, as interpreted by a human reader. Examples of available transformations are shown in Figure 3.3.

Available transformations are classified as **geometric transformations** (number 1-4: each class is a subset of the next class), **local transformations**, and **noises**:

1. **Isometries: rotation, translation.** Isometries are bijective maps between two metric spaces that preserve distances, they preserve lengths, angles and areas. In our case, rotation has to be constrained to a certain range in order to be label-preserving, the exact range of rotation may vary in function of scripts. Reflection is not desired because it is usually not label-preserving for text images. For human readers, a reflected character may not be acceptable or may even be recognized as another character.
2. **Similarities: uniform scaling.** Similarities preserve angles and ratios between distances. Uniform scaling includes enlarging or reducing.
3. **Affine transformations: shear, stretch.** Affine transformations preserve parallelism. Shear (also known as skew, slant, oblique) can be done either along horizontal axis or vertical axis. Stretch is usually done along the four axes: horizontal axis, vertical axis, main diagonal and anti-diagonal axis. Stretch can be seen as non-uniform scaling. Stretch along horizontal or vertical axis is also referred to as parallel hyperbolic transformation, stretch along main diagonal or anti-diagonal axis is also referred to as diagonal hyperbolic transformation [301].
4. **Perspective transformations.** Perspective transformations (also known as homographies or projective transformations) preserve collinearity. This transformation can be used to imitate camera viewpoint *i.e.,* 2D projection of 3D world.
5. **Local transformations**: Independent random vibration of the anchor points. Variation of the stroke width *e.g.,* thinning or thickening of the strokes. Variation of character proportion *e.g.,* length of ascenders and descenders.
6. **Noises** related to imaging conditions *e.g.,* Gaussian blur, contrast or brightness variation.

We used OmniPrint to generate two sets of datasets: OmniPrint-meta[1-5] and OmniPrint-MD. OmniPrint-meta[1-5] datasets, which are detailed in Section 3.3.1, provide a prototype of benchmarking few-shot learning and transfer learning algorithms. OmniPrint-MD datasets inherit similar design as OmniPrint-meta[1-5] and is part of Meta-Album, a meta-dataset of image classification (see Section 3.2). Details of OmniPrint-MD datasets can be found in the publication [3].

3.2 . Meta-Album: a multi-domain meta-dataset for image classification

Figure 3.4 – **Meta-Album sample images.**

We introduce *Meta-Album*, an image classification meta-dataset consisting of 40 datasets from 10 domains. Meta-Album covers a variety of domains, including ecology, manufacturing, textures, object classification, and character recognition, as well as a variety of scales: microscopic, macroscopic (human scale), or distant (remote sensing). Figure 3.4 shows example images from Meta-Album. While mostly re-purposing public datasets from heterogeneous sources to maximally vary recording conditions, we also introduce

Table 3.1 – Comparison between Meta-Album and other large-scale or (meta-) datasets

Dataset/ Meta-Dataset	# of domains	# of datasets	# of images	min/max classes per domain	min/max images per class	size on disk	multi-domain	lightweight (<20GB)	uniform # of images per class	uniform image size	repeated extensions
Meta-Dataset	7	10	53 068 000	43/1 696	3/140 000	210 GB	✓	✗	✗	✗	✗
VTAB	3	19	2 244 000	2/397	40/1 000	100 GB	✓	✗	✗	✗	✗
MS-COCO	1	1	328 000	80/80	9/10 777	44 GB	✗	✗	✗	✗	✗
Mini Imagenet	1	1	60 000	100/100	600/600	1 GB	✗	✓	✓	✓	✗
Omniglot	1	1	32 000	1 623/1 623	20/20	148 MB	✗	✓	✓	✓	✗
CUB-200	1	1	6 000	200/200	20/39	647 MB	✗	✓	✗	✗	✗
CIFAR-100	3	1	60 000	15/50	600/600	161 MB	✗	✓	✓	✓	✗
Meta-Album *Micro*	**10**	**40**	**32 000**	**19/20**	**40/40**	**380 MB**	✓	✓	✓	✓	✓
Meta-Album *Mini*	**10**	**40**	**220 950**	**19/706**	**40/40**	**3.9 GB**	✓	✓	✓	✓	✓
Meta-Album *Extended*	**10**	**40**	**1 583 624**	**19/706**	**1/187 384**	**15 GB**	✓	✓	✗	✓	✓

new datasets. These include *e.g.,* the 4 datasets for the character recognition domain, termed as *OmniPrint-MD* datasets [3], which were newly synthesized by OmniPrint (presented in Section 3.1) with different styles. All datasets are preprocessed, annotated, and formatted uniformly. Meta-Album comprises 3 different versions to be used for different scenarios and different amounts of computational resources. These 3 versions are Micro $\subset$ Mini $\subset$ Extended:

- **Micro**: a minimal version with 20 randomly selected classes and 40 images per class;
- **Mini**: a medium version with all classes having at least 40 images per class, including 40 randomly selected images per class;
- **Extended** a full version that consists of all classes and all images per class.

The variety of versions positions Meta-Album anywhere amongst small-scale datasets such as Omniglot [85], miniImageNet [302, 9] and CUB [303], which usually have at most 70000 images in total and weigh at most a few GB, or very large-scale benchmarks such as Meta-dataset [101] and VTAB [304], which have more than 50 million images, weigh at least a few hundreds GB, and require high-end super-computer clusters. We compare Meta-Album with previous benchmarks/datasets in Table 3.1.

The principal distinguishing feature of Meta-Album is that it has, by far, the largest number of domains and datasets, collected in different conditions. Secondly, while other benchmarks usually provide only raw data, we format all images uniformly as 128×128 pixel maps, which has two benefits: reducing the storage/memory footprint and facilitating the benchmarking of methods

Table 3.2 – Summary of the first 30 Meta-Album datasets (Mini version)

Domain ID	Domain Name	Set #	Dataset ID	Dataset Name	# Classes	# Images	Original source
		0	*BRD*	Birds	315	12 600	Birds 400 [305]
LR_AM	Large Animals	1	*DOG*	Dogs	120	4 800	Stanford Dogs [306]
		2	*AWA*	Animals with Attributes	50	2 000	AWA [307]
		0	*PLK*	Plankton	86	3 440	WHOI [308]
SM_AM	Small Animals	1	*INS_2*	Insects 2	102	4 080	Pest Insects [309]
		2	*INS*	Insects	104	4 160	SPIPOLL [310]
		0	*FLW*	Flowers	102	4 080	Flowers [311]
PLT	Plants	1	*PLT_NET*	PlantNet	25	1 000	PlantNet [312]
		2	*FNG*	Fungi	25	1 000	Danish Fungi [313]
		0	*PLT_VIL*	PlantVillage	38	1 520	PlantVillage [314, 315]
PLT_DIS	Plant Diseases	1	*MED_LF*	Medicinal Leaf	25	1 000	Medicial Leaf [316]
		2	*PLT_DOC*	PlantDoc	27	1 080	Plant Doc [317]
		0	*BCT*	Bacteria	33	1 320	DiBas [318]
MCR	Microscopy	1	*PNU*	PanNuke	19	760	PanNuke [319, 320]
		2	*PRT*	Subcel. Human Protein	21	840	Protein Atlas [321]
		0	*RESISC*	RESISC	45	1 800	RESISC45 [322]
REM_SEN	Remote Sensing	1	*RSICB*	RSICB	45	1 800	RSICB128 [323]
		2	*RSD*	RSD	38	1 520	RSD46 [324, 325]
		0	*CRS*	Cars	196	7 840	Cars [326]
VCL	Vehicles	1	*APL*	Airplanes	21	840	Multi-type Aircraft [327]
		2	*BTS*	Boats	26	1 040	MARVEL [328]
		0	*TEX*	Textures	64	2 560	KTH-TIPS [329, 330] Kylberg [331] UIUC [332]
MNF	Manufacturing	1	*TEX_DTD*	Textures DTD	47	1 880	Texture DTD [333]
		2	*TEX_ALOT*	Textures ALOT	250	10 000	Texture ALOT [334]
		0	*SPT*	100 Sports	73	2 920	100 Sports [335]
HUM_ACT	Human Actions	1	*ACT_40*	Stanford 40 Actions	39	1 560	Stanford 40 Actions [336]
		2	*ACT_410*	MPII Human Pose	29	1 160	MPII Human Pose [337]
		0	*MD_MIX*	OmniPrint-MD-mix	706	28 240	
OCR	Optical Char. Recog.	1	*MD_5_BIS*	OmniPrint-MD-5-bis	706	28 240	OmniPrint [2]
		2	*MD_6*	OmniPrint-MD-6	703	28 120	

independent of preprocessing steps. To that end, we optimized cropping and resizing to reduce dimensions as much as possible without degrading performance too much. In addition, Meta-Album includes datasets that have a large number of classes and class hierarchy annotations when available, with a minimum number of classes and examples per class: at least 20 classes (except two datasets having only 19 classes) with a minimum of 40 examples per class. Finally and importantly, we selected datasets that are not typically used in transfer learning or meta-learning benchmarks, *e.g.,* for pre-training backbone networks, such as ImageNet (which is included in *e.g.,* Meta-Dataset), or for conducting other meta-learning or transfer learning experiments, such as Omniglot, CIFAR-100, SVHN, or MNIST (which are included in *e.g.,* VTAB). This avoids giving an unfair advantage to methods that were developed using such commonly used datasets.

The 40 datasets of Meta-Album are grouped into 10 domains: large animals, small animals, plants, plant diseases, microscopy, remote sensing, vehicles, manufacturing, human actions, and optical character recognition (OCR). The datasets in each domain are presented in Table 3.2.

3.3 . Episodic few-shot learning: an instance of data modularity

3.3.1 . Episodic few-shot learning and OmniPrint

Few-shot learning is a machine learning problem in which new classification problems must be learned from just a few training samples per class (shots). This problem is particularly important in domains in which few labeled training samples are available, and/or in which training on new classes must be done quickly episodically, for example if an agent is constantly exposed to new environments [98, 338].

Recently, interest in few-shot learning has been revived (*e.g.,* [98, 99, 69]) and a novel setting proposed. The overall problem is divided into many subproblems, called *episodes*. Data are split for each episode into a pair (support set, query set). The support set plays the role of a training set and the query set that of a test set. In the simplified research setting, each episode is supposed to have the same number N of classes (characters), also called **"ways"**. For each episode, learning machines receive K training samples per class, also called **"shots"**, in the "support set"; and a number of test samples from the same classes in the "query set". This yields a **N-way-K-shot episode**. In some few-shot learning datasets, classes have hierarchical structures [101, 85, 194] *i.e.,* classes sharing certain semantics are grouped into super-classes. In such cases, episodes can coincide with super-classes, and may have a variable number of "ways". As discussed in Section 2.1.1, few-shot learning episodes are an instance of *imposed data modularity*.

To perform meta-learning, data are divided between a **meta-training set** and a **meta-test set**. In the meta-training set, the support and query set labels are visible to learning machines; in contrast, in the meta-test set, only support set labels are visible to learning machines; query set labels are concealed and only used to evaluate performance.

We use OmniPrint to investigate few-shot learning. Indeed, alphabets from many countries are seldom studied and have no dedicated OCR products available. A few-shot learning recognizer could remedy this situation by allowing users to add new alphabets with *e.g.,* a single sample of each character of a given font, yet generalize to other fonts or styles. Alphabets or partitions of alphabets in OmniPrint provide an instance of super-classes as discussed above.

Using OmniPrint to benchmark few-shot learning methods was inspired by Omniglot [85], a popular benchmark in this field. A typical way of using Omniglot is to pool all characters from different alphabets and sample subsets of N characters to create episodes (*e.g.,* $N = 5$ and $K = 1$ results in a 5-way-1-shot problem). While Omniglot has fostered progress, it can hardly push further the state-of-the-art since recent methods, *e.g.,* MAML [98] and Prototypical Networks [99] achieve a classification accuracy of 98.7% and 98.8%

respectively in the 5-way-1-shot setting. Furthermore, Omniglot was not intended to be a realistic dataset: the characters were drawn online and do not look natural. In contrast OmniPrint provides realistic data with a variability encountered in the real world, allowing us to create more challenging tasks.

Table 3.3 – **OmniPrint-meta[1-5] datasets** of progressive difficulty. Elastic means random elastic transformations. Fonts are sampled from all the fonts available for each character set. Transformations include random rotation (within -30 and 30 degrees), horizontal shear and perspective transformation.

X	Elastic	# Fonts	Transformations	Foreground	Background
1	**Yes**	1	No	Black	White
2	**Yes**	**Sampled**	No	Black	White
3	**Yes**	**Sampled**	**Yes**	Black	White
4	**Yes**	**Sampled**	**Yes**	**Colored**	Colored
5	**Yes**	**Sampled**	**Yes**	**Colored**	**Textured**

Figure 3.5 – **OmniPrint-meta[1-5] sample data:** Top: The same character of increasing difficulty. Bottom: Samples of characters showing the diversity of the 54 super-classes.

We created 5 datasets called OmniPrint-meta[1-5] of progressive difficulty, from which few-shot learning episodes can be carved (Table 3.3 and Figure 3.5). These 5 datasets imitate the setting of Omniglot, for easier comparison and to facilitate replacing it as a benchmark. The OmniPrint-meta[1-5] datasets share the same set of 1409 characters (classes) from 54 super-classes, with 20

Table 3.4 – **N-way-K-shot classification results** on the five OmniPrint-meta[1-5] datasets.

Setting		meta1	meta2	meta3	meta4	meta5
N=5 K=1	Naive	66.1 ± 0.7	43.9 ± 0.2	34.9 ± 0.3	20.7 ± 0.1	22.1 ± 0.2
	Proto [99]	$\mathbf{97.6 \pm 0.2}$	83.4 ± 0.7	75.2 ± 1.3	62.7 ± 0.4	61.5 ± 0.7
	MAML [98]	95.0 ± 0.4	$\mathbf{84.7 \pm 0.7}$	$\mathbf{76.7 \pm 0.4}$	$\mathbf{63.4 \pm 1.0}$	$\mathbf{63.5 \pm 0.8}$
N=5 K=5	Naive	88.7 ± 0.3	67.5 ± 0.5	52.9 ± 0.4	21.9 ± 0.1	26.2 ± 0.3
	Proto [99]	$\mathbf{99.2 \pm 0.1}$	93.6 ± 0.9	88.6 ± 1.1	79.2 ± 1.3	77.1 ± 1.5
	MAML [98]	97.7 ± 0.2	$\mathbf{93.9 \pm 0.5}$	$\mathbf{90.4 \pm 0.7}$	$\mathbf{83.8 \pm 0.5}$	$\mathbf{83.8 \pm 0.4}$
N=20 K=1	Naive	25.2 ± 0.2	14.3 ± 0.1	10.3 ± 0.1	5.2 ± 0.1	5.8 ± 0.0
	Proto [99]	$\mathbf{92.2 \pm 0.4}$	$\mathbf{66.0 \pm 1.8}$	$\mathbf{52.8 \pm 0.7}$	35.6 ± 0.9	35.2 ± 0.7
	MAML [98]	83.3 ± 0.7	65.8 ± 1.3	52.7 ± 3.2	$\mathbf{42.0 \pm 0.3}$	$\mathbf{42.1 \pm 0.5}$
N=20 K=5	Naive	40.6 ± 0.1	23.7 ± 0.1	16.0 ± 0.1	5.5 ± 0.0	6.8 ± 0.1
	Proto [99]	$\mathbf{97.2 \pm 0.2}$	$\mathbf{84.0 \pm 1.1}$	74.1 ± 0.9	56.9 ± 0.4	54.6 ± 1.3
	MAML [98]	93.1 ± 0.3	83.0 ± 1.0	$\mathbf{75.9 \pm 1.3}$	$\mathbf{61.4 \pm 0.4}$	$\mathbf{63.6 \pm 0.5}$

samples each, but they differ in transformations and styles. Transformations and distortions are cumulated from dataset to dataset, each one including additional transformations to make characters harder to recognize. We synthesized 32×32 RGB images of isolated characters.

We performed learning experiments on OmniPrint-meta[1-5] datasets with classical few-shot-learning baseline methods: Prototypical Networks [99] and MAML [98] (Table 3.4). The naive baseline trains a neural network from scratch for each meta-test episode with 20 gradient steps. MAML and Prototypical Networks were trained during 300 epochs, where each epoch is defined to be 6 batches of episodes, each batch contains 32 episodes. During meta-training, the model checkpoints were evaluated on meta-validation episodes every 5 epochs. The model having the highest accuracy on meta-validation episodes during training is selected to be tested on meta-test episodes. We split the data (1409 classes) into 900 characters for meta-training, 149 characters for meta-validation, 360 characters for meta-test. Performance is evaluated with the average classification accuracy over 1000 randomly generated meta-test episodes. The reported accuracy and 95% confidence intervals are computed with 5 independent runs (5 random seeds). The backbone neural network architecture is the same for each combination of method and dataset except for the last fully-connected layer, if applicable. It is the concatenation of three modules of Convolution-BatchNorm-Relu-Maxpool.

Our findings include that, for 5-way classification of OmniPrint-meta[1-5], MAML outperforms Prototypical Networks, except for OmniPrint-meta1; for 20-way classification, Prototypical Networks outperforms MAML in easier datasets and are surpassed by MAML for more difficult datasets. One counter-intuitive discovery is that the modeling difficulty estimated from learn-

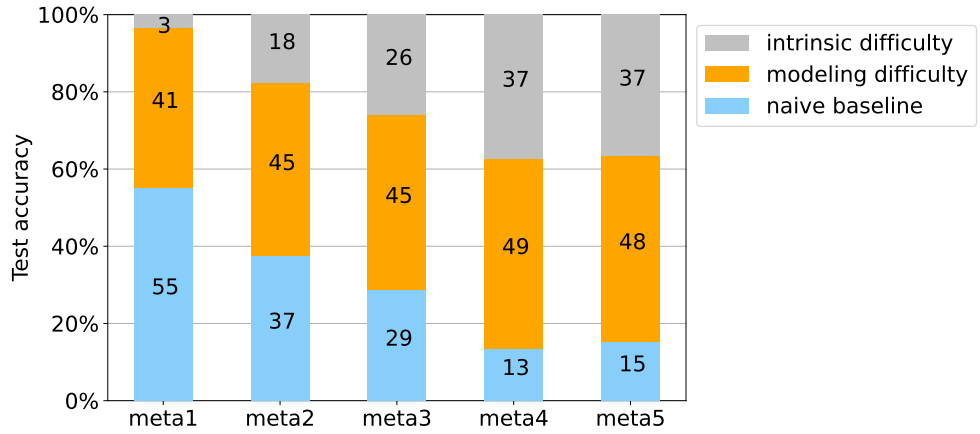


Figure 3.6 – **Difficulty of OmniPrint-meta[1-5] (few-shot learning)**: We averaged the results of N-way-K-shot experiments of Table 3.4. The height of the blue bar represents the performance of the naive baseline (low-end method). The top of the orange bar is the max of the performance of Prototypical Networks and MAML (high-end methods). Difficulty progresses from meta1 to meta4, but is (surprisingly) similar between meta4 and meta5.

ing machine performance (Figure 3.6) does not coincide with human judgement. One would expect that OmniPrint-meta5 should be more difficult than OmniPrint-meta4, because it involves natural backgrounds, making characters visually harder to recognize, but the learning machine results are similar.

We also used OmniPrint to investigate the influence of the training data size for few-shot learning *i.e.,* the number of meta-training episodes. For this purpose, we generated a larger version of OmniPrint-meta3 with 200 images per class (OmniPrint-meta3 has 20 images per class), to study the influence of the number of meta-training episodes. We compared the behavior of MAML [98] and Prototypical Network [99] on a log scale of number of meta-training episodes (from 10^0 to 10^5 meta-training episodes). The experiments visualized in Figure 3.7 show that the learning curves cross and Prototypical Network [99] ends with higher performance than MAML [98] when the number of meta-training episodes increases. Generally Prototypical Network performs better on this larger version of OmniPrint-meta3 than it did on the smaller version. This outlines that few-shot learning algorithms such as Prototypical Network can better leverage the availability of large data size than MAML.

OmniPrint provides extensively annotated metadata, recording all distortions. Thus more general paradigms of few-shot learning can be considered than the classical few-shot learning setting stated above. Such paradigms may include concept drift or covariate shift. In the former case, distortion parameters, such as rotation or shear, could slowly vary in time; in the latter case episodes could be defined to group samples with similar values of distortion

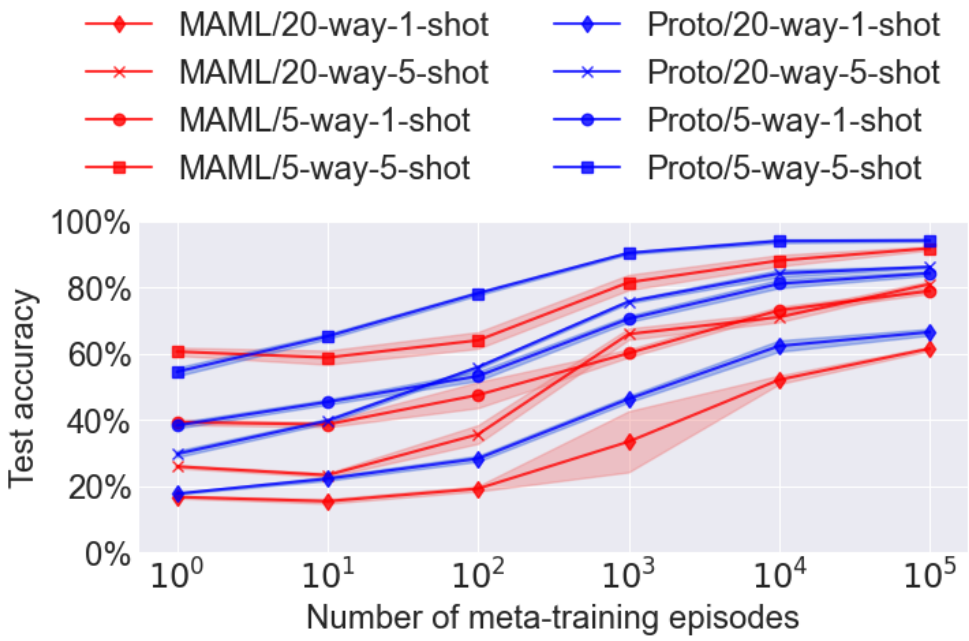


Figure 3.7 – **Influence of the number of meta-training episodes** with a larger version of OmniPrint-meta3. 95% confidence intervals are computed with 5 random seeds.

parameters.

To illustrate this idea, we generate episodes differently than in the "standard way" [99, 98, 302]. Instead of only varying the subset of classes considered from episode to episode, we also vary transformation parameters (considered nuisance parameters). This imitates the real-life situation in which data sources or recording conditions may vary between data subsets. We use OmniPrint-meta3 and OmniPrint-meta5, two datasets from OmniPrint-meta[1-5] described above, and generate episodes imposing that *rotation* and *shear* be more similar within episode than between episodes. In doing so, each episode becomes more internally cohesive, as support samples closely resemble query samples, while the episodes as a whole display greater diversity. The metadata-based episode generation algorithm is presented in Algorithm 1.

The experimental results, as depicted in Figure 3.8, reveal that with metadata-based episodes, the learning problem becomes less complex, leading few-shot learning algorithms to achieve superior classification performance. While initially surprising, these results make sense when considering that query samples become more similar to support samples in meta-test episodes, making them easier to learn. This research could be developed in various directions, including defining episodes differently at meta-training and meta-test

Algorithm 1: Metadata-based few-shot learning episode generation.

Input: Number of support images S, number of query images Q

```
// Assuming that metadata consists of real numbers.
```

for each episode **do**
  Randomly sample N classes $c_1, c_2, ..., c_N$
  for each class c_n **do**
    Find all samples $E_{c_n} = \{e_1, e_2, ...\}$ of class c_n, the metadata m_i of each sample $e_i \in E_{c_n}$ is a real-valued vector.
    Compute the bounding box B_{c_n} of the metadata vectors m_i.
    Randomly sample a centroid D within B_{c_n}.
    Select the $(S + Q)$ nearest neighbors $M = \{m_x, m_y, ..., m_{(S+Q)}\}$ from all the metadata vectors $m_1, m_2, ...$
    An sample e_i is selected to be part of the episode if and only if $m_i \in M$, all the selected samples form the set $\hat{E}_{c_n,D}$
    Randomly draw S samples from $\hat{E}_{c_n,D}$ to form the support set, the remaining samples serve as the query set.
  end
end

time *e.g.,* to study whether algorithms are capable of learning better from more diverse meta-training episodes, given fixed meta-test episodes.

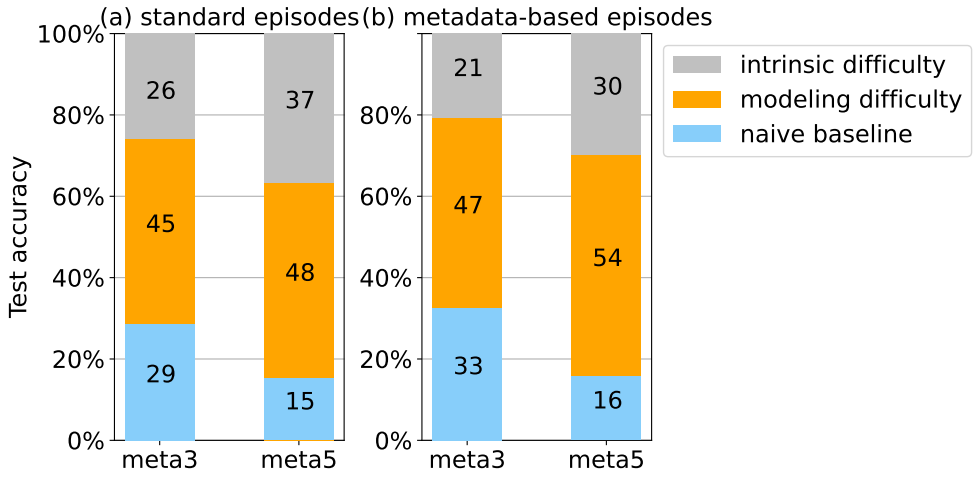


Figure 3.8 – **Comparing the difficulty of metadata-based episodes with that of standard episodes.** The height of the blue bar represents the performance of the naive baseline (low-end method). The top of the orange bar is the max of the performance of Prototypical Networks and MAML (high-end methods). (a) "Standard" episodes with uniformly sampled rotation and shear. (b) "Metadata" episodes: images within episode share similar rotation and shear; resulting tasks are easier than corresponding tasks using standard episodes.

3.3.2 . Episodic vs. batch meta-training

We used Meta-Album [3], introduced in Section 3.2, to organize a series of meta-learning competitions: MetaDL competition at NeurIPS 2021 [4] and Cross-Domain MetaDL competition at NeurIPS 2022 [5]. Both challenges aimed at assessing few-shot learning algorithms in a fair way.

Meta-learning is commonly employed to solve few-shot learning problems. It aims at leveraging similar task experiences to enable efficient adaptation to new tasks via a phase of meta-training. As the end-goal of few-shot classification is to produce a model that can effectively learn to recognize new episodes with few samples, the meta-test performance is evaluated on episodes of data, as described in Section 3.3.1. The meta-training phase, on the other hand, has the choice over different strategies.

Classical few-shot learning methods [98, 99] organize meta-training data into episodes with the intention of simulating the scenario encountered at meta-test time; each episode contains data from a subset of classes. We refer to this paradigm as *episodic* meta-training. On the other hand, a non-episodic approach [101] can be considered, this approach ignores the episodic structure of few-shot learning and merges all meta-training episodes into one "flat" dataset of labeled samples, it trains the model over all of the meta-training classes of the "flat" dataset at once. We refer to this paradigm as *batch* meta-

training. The choice between episodic meta-training and batch meta-training has been a topic of discussion among the research community [339, 340]. This debate highlights the effects of data modularity on the learning machines. Episodic meta-training is an instance of *imposed data modularity* because it groups data into episodes. Its monolithic counterpart is batch meta-training.

We investigate the choice between episodic meta-training and batch meta-training through the post-analysis of competitions we organized using Meta-Album (presented in Section 3.2). Competitions provide an equitable means to evaluate diverse approaches and stimulate contributions from the broader research community [341, 342, 343].

By analyzing the solutions of the top-ranked teams of MetaDL competition at NeurIPS 2021 [4] and Cross-Domain MetaDL competition at NeurIPS 2022 [5], we found that top-ranked participants do not use episodic meta-training, neural networks of top-ranked solutions are fine-tuned with batches of meta-training data instead of episodes.

We aim to do ablation studies to further investigate the effect of episodic meta-training using the same competition protocol as Cross-Domain MetaDL competition at NeurIPS 2022 [5]. However, the top-ranked solutions are too complex to modify to have a fair comparison. We instead chose to do the ablation studies on classical few-shot learning methods. The considered methods include Train-from-scratch, Prototypical Networks [99], Fine-tuning, and MAML [98]. Train-from-scratch does not perform any meta-training; instead, it directly learns each meta-test episode using only its support set. Fine-tuning consists of pre-training a backbone network with meta-training datasets and then only fine-tuning the last layer at meta-test time. All compared methods use a ResNet18 backbone.

The ablation studies follow the same competition protocol as Cross-Domain MetaDL competition at NeurIPS 2022 [5]. The number of classes in the meta-test episodes ranges from 2 to 20 ("ways", $N \in [2, 20]$), the support set contains 1 to 20 labeled samples per class ("shots", $K \in [1, 20]$), and the query set contains 20 samples per class. Furthermore, since this competition focuses on cross-domain meta-learning, all the episodes are carved out from Meta-Album [3] that contains datasets from 10 domains. In this competition, the data in one episode belongs to one dataset. Nevertheless, different episodes may come from different datasets, each belonging to a particular domain *e.g.,* insect classification, medical image classification, car classification. Since meta-test episodes can have a variable number of ways and shots, this competition uses the balanced accuracy (bac) normalized with respect to the number of ways N as the evaluation metric. This metric is defined as

$$\text{Normalized Accuracy} = \frac{bac - bac_{RG}}{1 - bac_{RG}}, \tag{3.4}$$

where bac is defined as

$$bac = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{correctly classified samples of class } i}{\text{total samples of class } i}, \tag{3.5}$$

and bac_{RG} is the accuracy of random guessing, *i.e.*, $\frac{1}{N}$.

For the meta-training strategy, we consider three cases: None (*i.e.*, the meta-training phase is skipped), batch meta-training (*i.e.*, the model is meta-trained with batches), or episodic meta-training (the model is meta-trained with episodes *i.e.*, N-way-K-shot episodes). At the same time, we also compare different classifiers on top of the backbone network: linear and nearest centroid classifier. Nearest centroid classifier is the strategy of Prototypical Networks where the flatten representation produced by the backbone is used to compute the centroids of each class and then the images on the query set are assigned to the closest centroid based on the Euclidean distance. On the other hand, the linear classifier consists of adding a linear layer to process the flatten representation produced by the backbone.

Inspired by the top-ranked solutions of the competition, we analyzed the impact of freezing different number of blocks of the backbone. We used Prototypical Networks for this analysis. Figure 3.9 shows that freezing 8 of the 9 blocks of ResNet18 (the backbone used by all analyzed methods) leads to the highest normalized accuracy. Therefore, we conduct all the previously mentioned ablation studies for two scenarios: (1) using an unfrozen ResNet18 and (2) using a ResNet18 with 8 of its 9 blocks frozen.

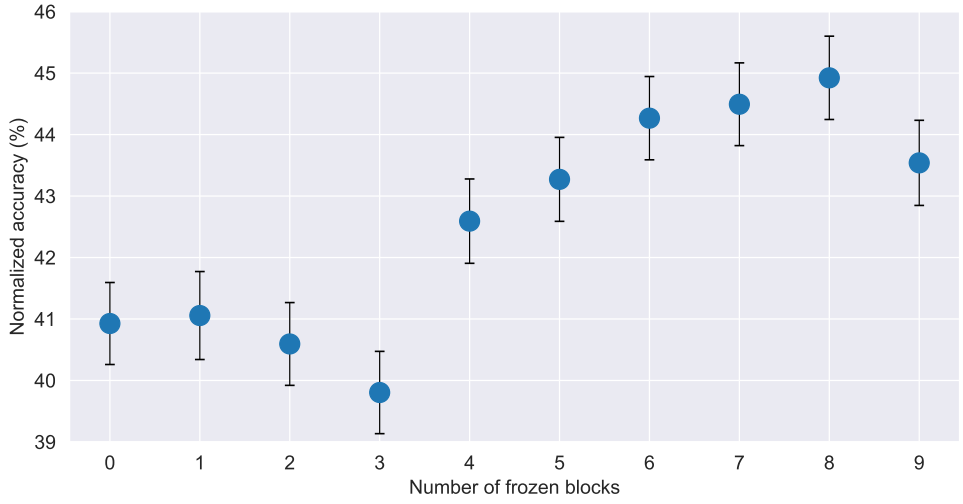


Figure 3.9 – Analysis of the impact of freezing a varying number of ResNet18 blocks (9 blocks in total) using Prototypical Networks. Each dot shows the average normalized accuracy over 6000 meta-test episodes.

Table 3.5 – **Ablation studies of the meta-training strategy.** The backbone is a pre-trained ResNet18. "Unfrozen backbone" means that all layers are updated during meta-training, "frozen backbone" means that the first 8 blocks are frozen (ResNet18 has 9 blocks in total). "NCC" refers to a nearest centroid classifier. The normalized accuracy is an average over 18000 [2-20]-way [1-20]-shot meta-test episodes (3 runs with 6000 episodes per run). The bold values indicate the best performance in each case.

Method	Meta-training Strategy			Classifier		Normalized Accuracy	
	None	Batch	Episodic	Linear	NCC	Unfrozen Backbone	Frozen Backbone
Train-from	✓			✓		24.35 ± 0.31	45.93 ± 0.39
-scratch	✓				✓	$\mathbf{43.54 \pm 0.40}$	43.54 ± 0.40
Prototypical	✓				✓	$\mathbf{43.54 \pm 0.40}$	43.54 ± 0.40
Networks		✓			✓	34.13 ± 0.37	40.87 ± 0.38
			✓		✓	40.93 ± 0.39	44.92 ± 0.39
Fine-tuning	✓			✓		38.56 ± 0.41	38.56 ± 0.41
	✓				✓	41.44 ± 0.39	41.44 ± 0.39
		✓		✓		23.24 ± 0.33	42.91 ± 0.41
		✓			✓	36.99 ± 0.38	44.77 ± 0.40
			✓	✓		3.03 ± 0.12	20.75 ± 0.34
			✓		✓	37.12 ± 0.37	45.30 ± 0.40
MAML	✓			✓		20.32 ± 0.35	19.08 ± 0.34
	✓				✓	$\mathbf{43.54 \pm 0.40}$	43.54 ± 0.40
		✓		✓		18.72 ± 0.34	16.33 ± 0.30
		✓			✓	22.14 ± 0.34	39.85 ± 0.40
			✓	✓		23.33 ± 0.34	27.03 ± 0.36
			✓		✓	43.00 ± 0.38	$\mathbf{47.01 \pm 0.40}$

Table 3.5 shows the results for all the experiments. It is worth noting that this table does not include the results of Train-from-scratch with batch or episodic meta-training because this method does not perform any meta-training by definition. Similarly, Prototypical Networks uses a nearest centroid classifier by definition.

We adopt a marginalized approach to compare the axes of variation of the ablation studies rather than analyzing them jointly. This marginalized analysis could offer insights into the design of novel algorithms.

We compare the effect of different meta-training strategies (skipping the meta-training phase, batch meta-training, and episodic meta-training) as shown in Figure 3.10. The left part of Figure 3.10 shows that when using an unfrozen backbone, episodic meta-training outperforms batch meta-training with both better mean normalized accuracy and median normalized accuracy. However, skipping the meta-training phase outperforms both batch meta-training and episodic meta-training in the case of an unfrozen backbone. The use of

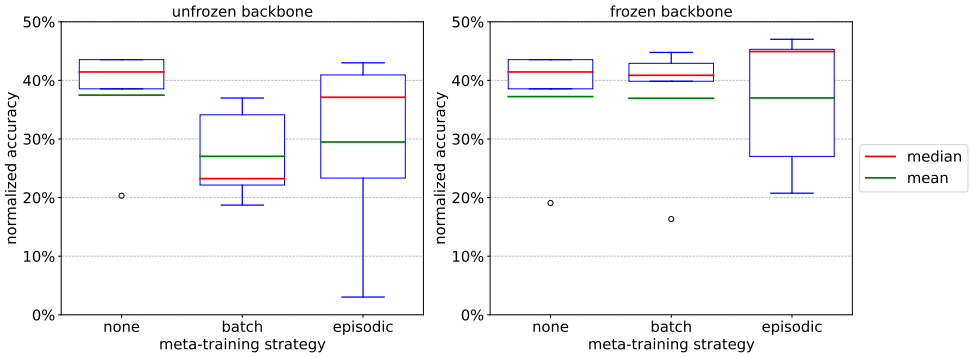


Figure 3.10 – Boxplot for the effect of different meta-training strategies in the ablation studies. The compared methods include Prototypical Networks, Fine-tuning, and MAML. Data is from Table 3.5. "frozen" means that 8 of the 9 blocks of the ResNet18 backbone are frozen during meta-training, whereas "unfrozen" means that the whole ResNet18 backbone is updated during meta-training. "none" means that the meta-training phase is skipped. "batch" means that the backbone is meta-trained with batches, "episodic" means that the backbone is meta-trained with episodes.

meta-training data harms the performance even if the meta-training data is supposed to be more similar to the meta-test data than the data used for pre-training (ImageNet). This result indicates that even if a meta-dataset is available for meta-training and contains the same domains as the meta-test data, it is tricky to benefit from it because the pre-trained backbones are already good and trained on a lot of data. When starting with pre-trained parameters, there is a risk of overfitting the meta-training data. This highlights the importance of regularization during meta-training *e.g.,* fine-tuning only the last few layers of the backbones, adding weight decay, and dropout.

We compare the effect of meta-training strategies separately for unfrozen backbone and frozen backbone. The right part of Figure 3.10 shows the results when the first 8 blocks are frozen during meta-training. There are 9 blocks in total in a ResNet18 backbone, so only the last block is updated during meta-training. With a frozen backbone, the best result is achieved by episodic meta-training (MAML + Episodic meta-training + Nearest Centroid Classifier), which outperforms the best performance with an unfrozen backbone by 3.47%. As shown in Figure 3.10, episodic meta-training shows a higher variance of performances than batch meta-training or skipping meta-training. Nevertheless, episodic meta-training shows advantages with a frozen backbone in that it achieves the best median and best-case performances; it also achieves the best worst-case performance when taking outliers into account. The mean performance is almost the same for these three meta-training strategies in the case of frozen backbone.

Furthermore, we also checked the performance change in Table 3.5 in a pairwise manner *i.e.,* comparing performances of the same ablation combination except for the choice of batch or episodic meta-training. This pairwise comparison shows that the use of episodic meta-training improves performances in almost all cases except for the fine-tuning approach with a linear classifier.

In summary, episodic meta-training should be carefully considered when designing novel few-shot learning algorithms, as it has the potential to provide good performances if applied judiciously. Although the winners of Cross-Domain MetaDL competition at NeurIPS 2022 relied on batch meta-training, which built upon the winning solution of MetaDL competition at NeurIPS 2021, our post-challenge analyses indicate that episodic meta-training could provide improved performance, in combination with *e.g.,* prevention of overfitting.

3.4 . Conclusion

In this chapter, we present our research contributions that revolve around the concept of data modularity. Central to our discussions are three contributions: OmniPrint, Meta-Album, and our case studies on episodic few-shot learning.

OmniPrint is a data synthesizer designed for images of isolated characters, showcasing modularity through its super-classes and latent generative factors. It grants complete control over the latent factors of the data-generating process, enabling the customization of the synthesized data.

Meta-Album is a meta-dataset containing 40 image classification datasets spread over 10 domains, ranging from ecology to manufacturing. Its modularity is evident through domain-based structure, and it includes datasets from OmniPrint. Meta-Album serves as an invaluable tool for benchmarking modular models in various applications, including few-shot learning, transfer learning, and meta-learning.

Few-shot learning enables Deep Learning models to train with minimal training data requirements, aligning with our objective of resource-efficient Deep Learning. We presented case studies on episodic few-shot learning. Here, data episodes in few-shot learning are regarded as data modules, where data samples are organized in a particular manner *e.g.,* by class subsets. One illustrative study uses OmniPrint to produce metadata-based episodes, adjusting latent generative factors to enhance data similarity within an episode, in line with the cohesion feature of modularity (presented in Section 1.2). Results suggested that few-shot learning algorithms can learn better with this new setting.

In another case study, we analyzed the merits of episodic meta-training in comparison to its monolithic counterpart, batch meta-training. While episodic meta-training leverages data episodes, batch meta-training merges all these episodes into one single dataset, training the model across all classes simultaneously. We investigated this problem through the post-analysis of few-shot learning competitions we organized using Meta-Album. Our findings indicate that, when episodic meta-training is judiciously employed *e.g.,* combined with overfitting prevention, it can improve the learning performance. This demonstrates the potential of using a modular approach to organize training data for improved learning performance.

In conclusion, this chapter showcases the practical implementation of data modularity and its impact on the Deep Learning models that utilize such structured data. Through the introduction of tools like OmniPrint and Meta-Album, which inherently embrace modular design, we contribute resources that can catalyze advancements in the field. Meanwhile, our case studies, utilizing these tools, provide tangible insights into the effects and benefits of data modularity.

4 - Contributions in the scope of task modularity

This chapter offers a concrete example of task modularity to which we made a contribution. As explained in Chapter 2, task modularity can manifest in various forms, involving the decomposition of a task into sub-tasks, each with a specific objective. In this specific context of task modularity, we narrow our focus to adversarial scenarios. Here, an Attacker module and an Evaluator module serve as the critic or teacher to a Defender module. Specifically, we present a modular evaluation mechanism for membership inference attacks, aiming to assess the privacy of Deep Learning models. The majority of the content in this chapter has been the subject of publication [6].

4.1 . LTU: modular evaluation against membership inference attacks

4.1.1 . Introduction

In today's data-driven landscape, organizations from large corporations to academic institutions are cautious with their information. This caution stems from concerns about privacy breaches and potential legal consequences. The current challenge is to develop protocols that protect data while still leveraging its potential value. Even though sensitive data should remain strictly confined within the source organization (Source), authorized researchers (Defender) can utilize it to develop predictive models (Defender model). These models, seen as the product of this effort, can be released provided that they maintain desired levels of utility and privacy.

Such application scenarios have prompted research into privacy challenges in Deep Learning [344, 345]. Our focus is on the setting of membership inference attack. In this context, attackers try to determine whether specific samples are part of the training dataset of the Defender model [345]. Our study considers the most complete release scenario: the predictive model, its training algorithm, and all its hyper-parameters.

At the core of our research is the design of a modularized evaluation mechanism, aimed at evaluating the robustness of Defender models against membership inference attacks. This tool assists the Source in deciding whether to release the Defender model (Figure 4.1). This evaluation mechanism consists of two modules: the Evaluator and the Attacker. The Evaluator conducts an assessment called the leave-two-unlabeled (LTU) evaluation. Through this process, it provides the Attacker with extensive information, excluding the membership label of two samples.

Task modularity emerges as a vital design choice. Drawing parallels from the dynamics in Generative Adversarial Networks and the teacher-student paradigm (discussed in Section 2.2.3), our Evaluator and Attacker function in roles resembling a teacher or critic, closely monitoring the Defender's learning process. The Evaluator, representing a specialized sub-task, ensures the integrity of the assessment. Without the Evaluator, the Attacker would have full access to the dataset, including all membership labels, which could compromise the evaluation's credibility. By separating these sub-tasks, the Evaluator acts as a safeguard, withholding certain information in its evaluation process. This approach exemplifies "independence" and "functional specialization", features of modularity as discussed in Section 1.2. In this context, task modularity ensures the evaluation's reliability and integrity.

4.1.2 . Problem statement and methodology

We consider the scenario where a data owner possesses a dataset, termed Source data and denoted as $\mathcal{D}_S$. This data owner intends to develop a predictive model using a portion of this dataset, while ensuring that privacy is preserved, especially against membership inference attacks. To achieve this, the data owner delegates the task to an agent named Defender, granting access to a random subset $\mathcal{D}_D \subset \mathcal{D}_S$ (Defender dataset). We denote by $\mathcal{M}_D$ the trained model (Defender model) and by $\mathcal{T}_D$ the algorithm used to train it (Defender training algorithm). The data owner wishes to release $\mathcal{M}_D$ and $\mathcal{T}_D$, provided that certain standards of privacy and utility of $\mathcal{M}_D$ are met.

To evaluate such utility and privacy, the data owner reserves a dataset $\mathcal{D}_R \subset \mathcal{D}_S$, disjointed from $\mathcal{D}_D$. It gives both $\mathcal{D}_D$ and $\mathcal{D}_R$ to a trustworthy agent called Evaluator. The Evaluator tags the samples with "membership labels": *Defender* or *Reserved*. Then, the Evaluator performs repeated rounds. In each round, the Evaluator randomly selects one Defender sample d and one Reserved sample r, which results in an attack dataset $\mathcal{D}_A = \mathcal{D}_D - \{\texttt{membership}(d)\} \cup \mathcal{D}_R - \{\texttt{membership}(r)\}$ for the Attacker. The attack dataset retains all information except for the membership labels of the two selected samples. The two unlabeled samples are referred to as u_1 and u_2, with each being equally likely to be from the Defender dataset. We refer to this procedure as leave-two-unlabeled (LTU) (see Figure 4.1).

The Attacker also has access to the Defender training algorithm $\mathcal{T}_D$, all its hyper-parameters, and the trained Defender model $\mathcal{M}_D$. This is the *worst-case scenario* in terms of attacker knowledge, as the only additional knowledge that could aid in a membership attack would be possession of the actual membership labels. The objective of the Attacker is correctly predicting which of the two samples u_1 and u_2 belongs to $\mathcal{D}_D$ for each LTU round.

We use the *LTU membership classification accuracy* A_{ltu} from N indepen-

dent LTU rounds (as defined above) to define the privacy score:

$$\begin{aligned}\texttt{Privacy} = &\min\{2\,(1-A_{ltu}),1\}\\ &\pm 2\sqrt{A_{ltu}(1-A_{ltu})/N}\,,\end{aligned} \tag{4.1}$$

where the error bar is an estimator of the standard error of the mean (approximating the Binomial law with the Normal law [346]). The weaker the performance of the LTU Attacker ($A_{ltu} \simeq 0.5$ for random guessing), the larger `Privacy`, and the better $\mathcal{M}_D$ should be protected from attacks.

The Evaluator also uses $\mathcal{D}_{uE} = \mathcal{D}_R$ to evaluate the utility of the Defender model $\mathcal{M}_D$. We focus on multi-class classification for c classes, and measure utility with the *classification accuracy* A_D of $\mathcal{M}_D$, defining utility as:

$$\begin{aligned}\texttt{Utility} = &\max\{(c\,A_D-1)/(c-1),0\}\\ &\pm c\sqrt{A_D(1-A_D)/|\mathcal{D}_R|}\,,\end{aligned} \tag{4.2}$$

The Attacker can use different strategies to make membership predictions. As an illustration, a selection of attacker strategies is shown below. In each LTU round, u_1 and u_2 denote the samples that were deprived of their labels.

1. The Attacker can rely on the generalization gap. It classifies u_1 as belonging to $\mathcal{D}_D$ if the loss function of $\mathcal{M}_D(u_1)$ is smaller than that of $\mathcal{M}_D(u_2)$.
2. The Attacker trains an attacker model $\mathcal{M}_A$ to predict membership. $\mathcal{M}_A$ uses any internal state or the output of $\mathcal{M}_D$ as input. Following its training, $\mathcal{M}_A$ is used to predict the labels of u_1 and u_2.
3. The Attacker trains two mock Defender models $\mathcal{M}_1$ and $\mathcal{M}_2$. The former is trained using $(\mathcal{D}_D - \{d\}) \cup \{u_1\}$ and the latter with $(\mathcal{D}_D - \{d\}) \cup \{u_2\}$. Both uses the training algorithm $\mathcal{T}_D$. If $\mathcal{T}_D$ is deterministic and independent of training sample ordering, either $\mathcal{M}_1$ or $\mathcal{M}_2$ should be identical to $\mathcal{M}_D$. Otherwise, one of them should be "closer" to $\mathcal{M}_D$. The sample corresponding to the model closest to $\mathcal{M}_D$ is classified as being a member of $\mathcal{D}_D$.
4. Starting from the trained Defender model $\mathcal{M}_D$, the Attacker executes one gradient learning step with either u_1 or u_2 using $\mathcal{T}_D$. The Attacker then contrasts the outcomes based on gradient norms and sample prediction changes. The sample associated with smaller gradient norms or fewer changes in sample predictions is classified as being a member of $\mathcal{D}_D$. It should be noted that this strategy presupposes that the Defender training algorithm $\mathcal{T}_D$ allows computing gradients.

The first two attack strategies only attack on $\mathcal{M}_D$, while the last two attack strategies attack on both $\mathcal{M}_D$ and $\mathcal{T}_D$.

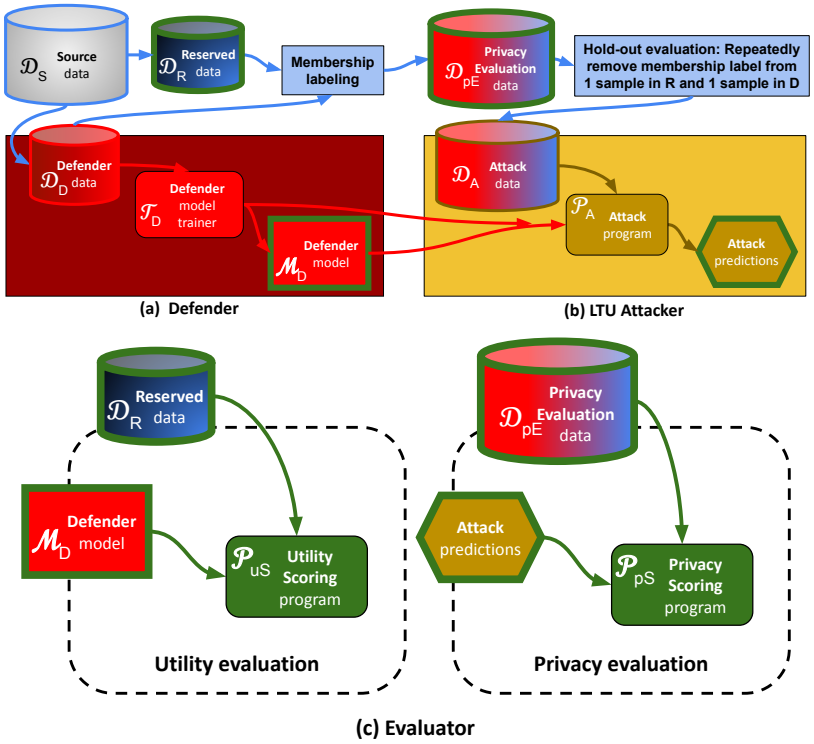


Figure 4.1 – **Illustration of our modularized evaluation mechanism for membership inference attack.** Source data are divided into Defender data, to train the predictive model under attack (Defender model) and Reserved data to evaluate such a model. The Defender model training algorithm creates a model optimizing a utility objective, while being as resilient as possible to privacy attacks. The evaluation mechanism includes an Attacker and an Evaluator: The evaluation mechanism performs an LTU evaluation by repeatedly providing the Attacker with all of the Defender and Reserved data samples, together with their *membership label*, hiding only the membership label of 2 samples. The Attacker must turn in the membership label (Defender data or Reserved data) of these 2 samples (Attack predictions). The Evaluator computes two scores: Attacker prediction error (Privacy metric), and Defender model classification performance (Utility metric).

4.1.3 . Experiments

In this section, we showcase the usage of the LTU evaluation mechanism.

We utilize neural networks as the Defender model and implement the attack strategy leveraging gradient descent. Our exploration covers two kinds of Defender models: one based on supervised learning and the other on unsupervised domain adaptation (UDA) [347]. Our experiments use the QMNIST dataset [348], an expanded version of the popular MNIST [120] dataset.

For the supervised approach, we employed ResNet50 [10] pre-trained on ImageNet [8]. We then retrained it on the Defender set of QMNIST using the cross-entropy loss. Results from this are reported in Table 4.1 (line "Supervised learning"), indicating both good utility and privacy.

Defender Model	**Utility**	**Privacy**
Supervised learning	1.00 ± 0.00	0.97 ± 0.03
Unsupervised domain adaptation	0.99 ± 0.00	0.94 ± 0.03

Table 4.1 – Utility and privacy of ResNet50 Defender models trained on QMNIST.

In an effort to still improve privacy, we explored the potential of UDA algorithms. UDA, a type of transfer learning, leverages labeled data from a source domain to train models to excel in an unlabeled target domain. This implies that, when the Defender set is used as the target domain, its labels are not used during Defender model training. We hypothesized that this technique might improve privacy against membership inference attacks. We use large-Fake-MNIST as the source domain. Large-Fake-MNIST is a synthetic dataset generated by OmniPrint (presented in Section 3.1), it is similar to MNIST [120] and consists of 500000 white-on-black images distributed evenly across digits. The target domain, meanwhile, is a subset of QMNIST.

Our UDA method of choice is DSAN [347] because of its good performance [2]. DSAN optimizes the neural network with the sum of a classification loss (cross-entropy loss) and a transfer loss (local MMD loss [347]). We experimented with three variants of attacks on this UDA model: (1) attacking only the classification loss, (2) attacking only the transfer loss, and (3) attacking both losses. Interestingly, the first variant proved most effective, treating the model as if trained with supervised learning. However, as shown in Table 4.1, UDA did not yield improved performance. We attribute this to the fact that the supervised model under attack performs well on this dataset and already has a very good level of privacy.

4.2 . Conclusion

This chapter delved into the role of task modularity in addressing privacy issues associated with Deep Learning models. In particular, we introduced a modular evaluation mechanism, termed LTU, designed to analyze the worst-case scenarios regarding the knowledge level of membership inference attackers.

Task modularity advocates for the breakdown of a task into sub-tasks. In the approach we proposed, we decomposed the evaluation process into two sub-tasks: those of the Evaluator and the Attacker. The Evaluator's sub-task involves conducting successive LTU rounds, ensuring that in each round, two examples' labels are kept hidden from the Attacker. This division allows for an authentic evaluation process since the Attacker's sub-task centers on simulating attack strategies with only the remaining data. This separation ensures that the assessment maintains its fairness and integrity, preventing the Attacker from leveraging the ground-truth membership labels.

In conclusion, this chapter showcases how task modularity can be advantageous through one example problem: addressing the privacy issues inherent in Deep Learning models. Task modularity offers benefits such as improved clarity and enhanced integrity due to the independence of modules.

5 - Contributions in the scope of model modularity

This chapter contains our research contribution regarding model modularity. As discussed in Section 2.3, the modularity of pre-trained neural networks is inherent in their architecture, which can be decomposed into a collection of modules. These modules can take various forms, such as convolutional kernels, blocks, and so on, depending on the level of granularity. This modular characteristic facilitates the application of pruning techniques, which involve the removal of redundant modules, thereby making pre-trained models more lightweight and efficient for utilization. Additionally, modules can be assembled into an ensemble to boost performance (refer to Section 2.3.4.1 for more details).

Our proposed approach for the efficient reuse of a pre-trained model is termed RRR, which stands for “Reuse, Reduce, and Recycle". This approach employs reduction and recycling techniques to adjust the pre-trained ResNet152 model. Most of the content of this chapter has been the subject of publication [7].

5.1 . RRR-Net: reusing, reducing, and recycling a modular neural network

5.1.1 . Introduction

Over the last decade, Deep Learning has set new standards in computer vision. While it has achieved state-of-the-art in various academic and industrial fields, training deep networks from scratch requires massive amounts of data and hours of GPU training, which limits its application in data-scarce and resource-scarce scenarios.

This limitation has been mainly addressed through the notion of *Transfer learning* [349, 350]. Here, knowledge is transferred from a *source domain* (typically learned from a large dataset) to one or several *target domains* (typically with less available data). A common transfer learning approach is fine-tuning [191], in which a considerable part (the *backbone*) of a pre-trained neural network is reused; the last layer (classification head) is replaced with a new classifier and the last layers are retrained to the new task at hand.

As discussed in Section 2.3, the architectural design of pre-trained neural networks inherently possesses modularity, allowing them to be broken down into separate modules. On the other hand, modern neural networks are thought of as being “the bigger, the better” as big networks keep beating large benchmarks (such as ImageNet [194]). However, they are considerably

over-parameterized when applied to smaller tasks. There is evidence that low-complexity models can, in some conditions, lead to comparably good or better performance [351].

Building on these insights, our initiative is to implement the three classical resource-saving principles "**R**euse, **R**educe, and **R**ecycle" (RRR) [352] to the maximum extent possible on pre-trained modular neural networks. The "Reduce" step means reducing the number of modules in the pre-trained modular neural network, while the "Recycle" step is realized by constructing a voting ensemble of modules from the existing pre-trained modules. We leverage the reusability of parts, rather than the entirety, of a pre-trained modular neural network. To showcase the feasibility of this idea, we selected the popular ResNet152 model [10] as our foundational modular model.

This work combines several ideas: **reusing** a pre-trained neural network, **reducing** it, and splitting the model into several branches to use ensemble techniques (**recycling**). We briefly discuss related work in these three domains.

- **Reusing pre-trained neural networks** has been a common practice in Deep Learning. Given the availability of neural networks trained on large datasets, the vanilla fine-tuning approach [191] remains the method of choice for transfer learning with neural networks. Guo *et al.* [353] propose to adaptively fine-tune pre-trained models on a per-instance basis. Wortsman *et al.* [354] and Liu *et al.* [355] propose methods to merge knowledge from multiple pre-trained models. Our method can be seen as performing neural architecture searches in the search space defined by ResNet152 [10], which allows *reusing* its pre-trained parameters.
- **Model pruning** is one way to reduce the storage/computing resource requirements of neural networks. It removes redundant parts (parameters, channels, etc.) that do not significantly contribute to the performance. Model pruning can happen at different granularities, depending on the topology constraints of the removed parts [174, 356, 357, 358, 359]. The *reduction* step of our methodology (Section 5.1.2.2) can be categorized as a coarse-grained (block-wise) pruning approach, as opposed to finer-grained (element/channel-wise) pruning approaches. We compare our *reduction* step and finer-grained pruning approaches in Section 5.1.3.2.
- **Ensemble methods** [261, 360, 230] combine multiple models to improve generalization and robustness. Neural network ensemble methods include bagging [361], Snapshot Ensemble [362], and Fast Geometric Ensemble [363]. Several works [141, 131, 262, 364] investigate tree-structured neural networks. The regularization technique dropout [365] can also be interpreted as an implicit ensemble technique. In this work, we use ensembling to *recycle* parts of the pre-trained network, as described in

Phases	output size	Blocks		Num. parameters		FLOPs (10^6)	
		Baseline	Ours (conv4_x, conv5_x: each branch)	Baseline	Ours	Baseline	Ours
conv1	32×32	7×7, 64, conv (stride 2) 3×3, max-pool (stride 2)	7×7, 64, conv (stride 2) 3×3, max-pool (stride 2)	9.5k	9.5k	40	40
conv2_x	32×32	$\begin{bmatrix} 1\times1,\ 64 \\ 3\times3,\ 64 \\ 1\times1,\ 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1,\ 64 \\ 3\times3,\ 64 \\ 1\times1,\ 256 \end{bmatrix}\times1$	70k (75k)	70k (75k)	73 (78)	73 (78)
conv3_x	16×16	$\begin{bmatrix} 1\times1,\ 128 \\ 3\times3,\ 128 \\ 1\times1,\ 512 \end{bmatrix}\times8$	$\begin{bmatrix} 1\times1,\ 128 \\ 3\times3,\ 128 \\ 1\times1,\ 512 \end{bmatrix}\times1$	280k (379k)	280k (379k)	72 (123)	72 (123)
conv4_x	8×8	$\begin{bmatrix} 1\times1,\ 256 \\ 3\times3,\ 256 \\ 1\times1,\ 1024 \end{bmatrix}\times36$	$\begin{bmatrix} 1\times1,\ 256/\sqrt{b}-a \\ 3\times3,\ 256/\sqrt{b}-a \\ 1\times1,\ (256/\sqrt{b}-a)\times4 \end{bmatrix}\times1$	1.12M (1.51M)	136k (333k) × 8	72 (122)	9 (30) × 8
conv5_x	4×4	$\begin{bmatrix} 1\times1,\ 512 \\ 3\times3,\ 512 \\ 1\times1,\ 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1,\ 512/\sqrt{b}-a \\ 3\times3,\ 512/\sqrt{b}-a \\ 1\times1,\ (512/\sqrt{b}-a)\times4 \end{bmatrix}\times1$	4.46M (6.04M)	553k (745k) × 8	72 (122)	9 (15) × 8
Total				58.23M	9.09 M	3799	601

Table 5.1 – **The structure and approximated statistics of ResNet152 and RRR-Net.** We call “Baseline" the original ResNet152 [10]. We call “Ours" the reduced and recycled architecture. Reduction means removing blocks from each phase (until one block in each phase). Recycling means splitting blocks in the conv4_x and conv5_x phases into multiple branches while preserving the total number of parameters and FLOPs. This table assumes the input size is 128× 128. Building blocks are shown in brackets, with the numbers of blocks stacked. For conv4_x and conv5_x in the “Ours" column, blocks are split into branches, where b denotes the number of branches, and a is a scalar to adjust the model size. The “Num. parameters" and “FLOPs" columns assume $b = 8$. The values in parentheses indicate the statistics of the first block of each phase, which is responsible for downsampling. k $= 10^3$, M $= 10^6$. FLOPs mean floating-point operations or multiply-adds; they are measured with respect to a single $128 \times 128 \times 3$ image using the open source software Pytorch-OpCounter [12].

Section 5.1.2.3.

This study aims at boosting the compactness and efficacy of a pre-trained modular neural network. Beyond just introducing a model compression technique, this research emphasizes how efficiently one could reuse existing knowledge in a pre-trained modular model.

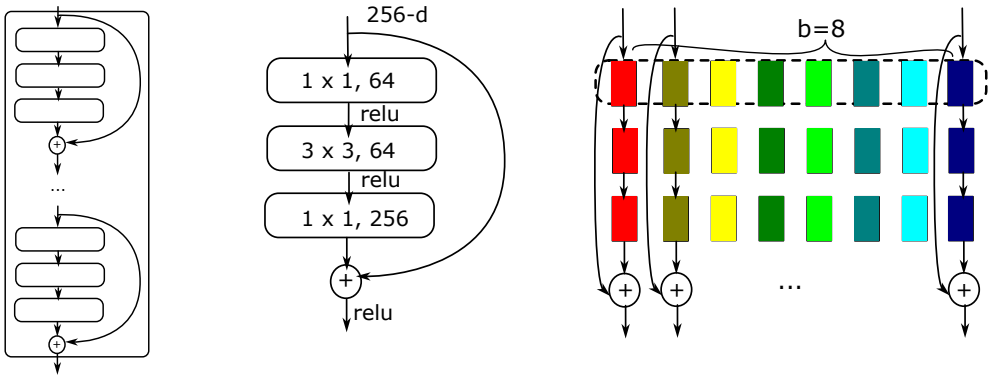
5.1.2 . The RRR principle for image classification

When creating an artifact based on the RRR principle [352], one selects an object of interest for *reuse*, then *reduces* its complexity, and finally *recycles* parts of it for some innovative modification. Adopting the RRR principle, our suggestion to arrive at a high-quality neural network for image classification is as follows:

- **Reuse:** Due to its great performance and simplicity, we choose a ResNet152 pre-trained on ImageNet [194] as the basis (backbone).
- **Reduction:** We reduce the backbone to its basic version by eliminating potentially dispensable blocks (modules).
- **Recycling:** We keep the first few blocks (stump) as the feature extractor and split the remaining blocks into multiple branches to create a voting ensemble.

We next describe each of these points in more detail.

5.1.2.1 Reuse: description of the pre-trained backbone



(a) Phase (b) Block (c) Branches

Figure 5.1 – Illustration of phase, block, and branches. (a) A phase represents a group of blocks. (b) A block, also called bottleneck block, stacks 3 convolutional layers with a skip-connection. (c) A block can be split into b branches.

We reuse a ResNet152 [10] model pre-trained on ImageNet [194]. This model consists of five groups of blocks, which we call *phases* (Table 5.1): conv1, conv2_x, conv3_x, conv4_x, conv5_x. The conv1 phase contains a convolutional layer and a max-pooling layer; this can be seen as a special block. All other phases include multiple *bottleneck blocks* (Figure 5.1 (a)). A bottleneck block (Figure 5.1 (b)) consists of 3 convolutional layers and a skip-connection path (directly connecting input and output). All bottleneck blocks within a phase are identical, with the exception of the first one, which includes an additional convolutional layer in its skip-connection path. Feature map downsampling is performed by conv1, conv3_1, conv4_1, and conv5_1. As seen in Table 5.1, ResNet152 repeats the same blocks many times in each phase.

While our study focuses on ResNet152, the reduction step described in Section 5.1.2.2 can, in principle, also be applied to other ResNet architectures. The important properties are the skip-connections and that we have phases consisting of identical blocks. In this case, we can configure the phases by

deciding upon the number of blocks they contain; the initial ResNet152 serves as a template.

5.1.2.2 Reducing: model block pruning

The ResNet architecture has the benefit of a modular structure consisting of identical blocks with skip-connections. It allows us to perform a neural architecture search that aims to find the optimal number of blocks in each phase. Since the goal here is to *reduce* the network, the maximum number of blocks in each phase is determined by the original ResNet152 architecture. To ease the notation, we will name the architectures as ResNet_x_1_x_2_x_3_x_4, where x_i is the number of blocks used in phase i, the conv1 phase is phase 0. Following this notation, the original ResNet152 is written as ResNet_3_8_36_3; the smallest possible network in this regime is ResNet_1_1_1_1, which contains only one block per phase.

For ResNet152, the search space consists of 2592 possible architectures. Extensively evaluating all of them once on one dataset could take several weeks using GPU, which is computationally expensive. Thus, we resort to a greedy forward selection approach to narrow down the search space. This approach, termed forward block selection v1, starts with the simplest architecture ResNet_1_1_1_1 and successively adds one block at a time until no improvement can be observed. Within this logic, we fill one phase before moving to the next phase until we get ResNet152 (ResNet_3_8_36_3). The pseudocode of the procedure is given in Algorithm 2. This method reuses parameters pre-trained on ImageNet as initialization.

An alternative approach would be to add blocks in a cyclic way *i.e.,* always add the block to the phase with the fewest blocks and where blocks may still be added. However, we found in preliminary experiments that these two approaches lead to the same results, so we do not present results for different approaches.

5.1.2.3 Recycling: ensembling network branches

One way to recycle a given pre-trained backbone is to split a part of it into an *ensemble of branches*. It means, starting from a certain layer, partitioning convolutional filters and adjusting layer connections so that all subsequent layers are influenced by exactly one of the filter sets in the partition. It creates one sub-network (branch) for each filter set in the partition (Figure 5.1 (c)). Each branch ends with a branch-specific output layer with softmax activation. The branches are merged by averaging the probability distributions predicted by different branches. In this sense, the branches can be seen as ensemble members whose votes are aggregated with an average.

Algorithm 2: The forward block selection v1 algorithm for reducing ResNet.

Input: ResNet152, Task={TrainSet, TestSet}
Output: MiniNet
Initialize
MiniNet = ResNet_1_1_1_1 + new_classification_layer
MiniNet.train_all_layers(TrainSet, epochNum)
Accuracy* = MiniNet.evaluate(TestSet)
OldAccuracy = Accuracy*
# Traverse phases
for $P = 2 : 5$ **do**
  # Traverse blocks
  # $maxBlock(2) = 3, maxBlock(3) = 8$
  # $maxBlock(4) = 36, maxBlock(5) = 3$
  for $i = 2 : maxBlock(P)$ **do**
    # Add block i in phase P.
    MiniNet = MiniNet $+convP\_i$
    # Continue training, no re-initialization.
    MiniNet.train_all_layers(TrainSet, epochNum)
    Accuracy = MiniNet.evaluate(TestSet)
    if (Accuracy - OldAccuracy) / Accuracy* $\geq \epsilon$ **then**
      OldAccuracy = Accuracy
      continue
    else
      return MiniNet $-convP\_i$
    end if
  end for
end for

We seek to do the branching in such a way that the overall model size and FLOPs (multiply-adds) are preserved. Each branch is constructed by adjusting the number of output channels of convolutional layers. For each branch, the number of output channels C_b is computed as follows:

$$C_b = \left\lfloor \frac{C_o}{\sqrt{b}} \right\rfloor - a \tag{5.1}$$

where C_o is the number of output channels in the original pre-trained convolutional layer, $\lfloor \cdot \rfloor$ is the floor function, and b is the number of branches to create. The variable a in Equation 5.1 is useful to ensure that the total number of parameters in the branches is reduced or equal to the original model's parameter count. The value of a relies on: the number of branches b, the

starting point from which to split the model, the number of classes of the downstream task, and the model structure (*e.g.,* ResNet_1_1_1_1 or ResNet152). a can be empirically computed by finding the smallest value that satisfies the total parameter count constraint. This computation results in each branch having C_b output channels as shown in Table 5.1 ("Ours" columns).

Each branch is initialized with a part of pre-trained kernels. In Algorithm 3, we describe the process of splitting pre-trained layers into a set of branches. The input to Algorithm 3 is the parameter tensors P of the pre-trained model and the number of desired branches N. The output of this algorithm is the parameter tensors T of the branches. Here, tensor indices are assumed to start at 1. At layer k, the shape of the 4-dimensional parameter tensor in a convolutional layer is $(C^{k,out}, C^{k,in}, s^h, s^w)$. $C^{k,out}$ means the number of output channels, $C^{k,in}$ means the number of input channels, s^h and s^w are height and width of convolutional kernels. For the pre-trained model, we add the subscript o; for the branch, we add the subscript b. This is also illustrated in Figure 5.2.

Algorithm 3: The algorithm describing how to split pre-trained kernels into branches.

Require: P the parameter tensors of the pre-trained model
Require: N denotes the number of branches
# Traverse pre-trained layers to split
for $k = 1 : \text{size}(P)$ **do**
  # Traverse input channels at layer k
  for $x = 1 : C_b^{k,in}$ **do**
    # Compute output channel indices I
    if $C_o^{k,out} \geqslant (C_b^{k,out} \times N)$ **then**
      $I = 1 : (C_b^{k,out} \times N)$
    else
      $I = 1 : C_o^{k,out}$
      while $\text{size}(I) < (C_b^{k,out} \times N)$ **do**
        $I = \text{concatenate}(I, \text{shuffled}(1 : C_o^{k,out}))$
      end while
      $I = I[: (C_b^{k,out} \times N)]$
    end if
    $start = 1, end = C_b^{k,out}$
    for $i = 1 : N$ **do**
      $T_k[i][:, x, :, :] = P_k[I[start : end], x, :, :]$
      $start = start + C_b^{k,out}, end = end + C_b^{k,out}$
    end for
  end for
end for

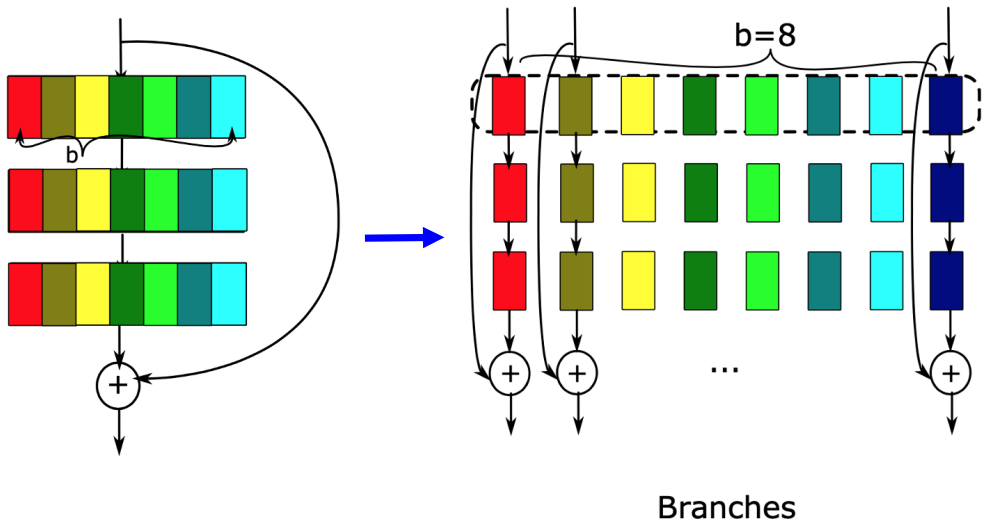


Figure 5.2 – **Illustration of the way to split pre-trained kernels into branches in RRR-Net.**

In our design, we keep the conv1, conv2_x, and conv3_x phases as the shared stump (preprocessor), and we create the branches starting from conv4_1 *i.e.,* the first block of the conv4_x phase. This way, conv4_x and conv5_x effectively constitute an ensemble of independent sub-networks (branches). The reason for branching at the last two phases is that these two phases include the most number of parameters and output channels, allowing for more branches to be made. By maintaining the total parameter count, the more branches are created, the fewer parameters each branch will have.

There can be many methods to train this ensemble of branches. One method is to train branches as if they were individual models without using specific ensemble training techniques. We call this training approach **naive ensemble**. Other choices include bagging [361], Snapshot Ensemble [362] and Fast Geometric Ensemble [363]. All ensemble training methods can be accompanied by random data augmentation to create diverse ensemble members. The stump (conv1, conv2_x, conv3_x) shared by branches is frozen; it keeps the pre-trained parameters.

5.1.3 . Experiments

We carry out comprehensive experiments in this section. We evaluate our methodology's reduction step compared to finer-grained pruning approaches. We perform model selection and evaluation for our methodology's reduction and recycling steps. Finally, we demonstrate the performance of the proposed model on a large benchmark. We also report results from a multi-criteria comparison, including accuracy, inference time, and the number of

parameters.

5.1.3.1 Experimental settings

We conduct experiments on Meta-Album micro [3], which is presented in Section 3.2. It consists of 40 datasets and represents a diverse and challenging set of data for our models to learn. Each dataset contains 128×128 images, and there are 20 classes per dataset (except for two datasets which only have 19 classes). Each class in each dataset has 20 images for the training and 20 images for the testing (*i.e.,* both the training set and the test set contain 400 images each). The choice of using this benchmark for our experiments is due to its diversity of problems and because it demonstrates low-resource scenarios (*i.e.,* few training examples available). These scenarios are common in real-world applications and highlight challenges faced when developing learning models with limited data.

Figure 5.3 – **Sample images from ICDAR-micro.**

We formatted an extra held-out dataset to perform model selection and hyper-parameter validation: we carved out a dataset from an OCR dataset of images of alphanumeric characters in the wild [366]. This dataset is dimensioned similarly to the Meta-Album benchmark. It has 20 classes, 15 images per class for training, 5 images per class for validation, and 20 images per class for the test. We call this dataset ICDAR-micro (named after its original source). Some sample images are shown in Figure 5.3.

In the spirit of Automated Machine Learning, we optimize all the hyper-parameters of our model with the ICDAR-micro dataset.[1] This allows us to use a simple train/test split when we evaluate our method on the Meta-Album benchmark since we do not need validation splits to select any setting.

Experimenting on the ICDAR-micro dataset allowed us to determine the following settings: The AdamW optimizer [367] has been chosen, the learning rate is set to 10^{-3}, the weight decay equals 0.01, the batch size is set to 32, the cross-entropy has been chosen as the loss function, the model parameters

1. Admittedly, we could have used more datasets, but, as it turns out, we already obtained quite good results with this strategy.

are initialized with parameters pre-trained on ImageNet [194] (as opposed to training from scratch). Data augmentation transformations include rotation, translation, scaling, shear, random brightness/contrast/color change, sharpening, image inverting, gaussian noise, motion blur, Jpeg compression, posterization, histogram equalization, and solarization. Two transformations are drawn uniformly at random with replacement and then sequentially applied to each training example.

The number of training epochs for each model is 300. Except for the pruning experiments, we perform model exponential moving average (EMA) during the last 60 epochs with a decay rate $d = 0.9833$ for all compared models (if applicable). Model exponential moving average is one way to stabilize the model training and smooth out noises. It keeps a running average θ^{EMA} of the model parameters θ during training. θ^{EMA} is updated after each training epoch as follows:

$$\theta_{t+1}^{\text{EMA}} = d\theta_t^{\text{EMA}} + (1-d)\theta_{t+1} \tag{5.2}$$

where $\theta_{t+1}^{\text{EMA}}$ is the model parameter running average at epoch $t+1$, θ_t^{EMA} is the model parameter running average at epoch t, θ_{t+1} is the current value of the model parameter at epoch $t+1$, d is the decay rate. The running average θ^{EMA} is used to report the performance (error rate) on the test split.

We also used the ICDAR-micro dataset to tune the reduction and recycling procedure for our low-resource regime: (1) how much pruning of the original backbone we could do (to reduce inference time and storage); (2) into how many branches we should split the pre-trained backbone (to gain performance by ensembling). All experimental results are averaged over repeated runs. 95% confidence intervals are computed over repeated runs using t-distribution.

5.1.3.2 Results from the reduction procedure

In this section, we summarize two results observed in the reduction procedure. First, we compare the inference speed of our methodology's *reduction* step with finer-grained pruning approaches that could be applied to reduce the network. Second, we study the extent to which we could reduce the network without significantly degrading performance on the validation dataset ICDAR-micro.

Comparing pruning granularities

This section investigates the potential gain in hardware inference speed of different pruning granularities; here, we ignore the classification performance. We compare 3 levels of pruning granularity: (1) element-wise prun-

ing, which treats individual parameters as removable modules; (2) channel-wise pruning, in which channels within a convolutional layer act as removable modules; (3) block-wise pruning, where a single ResNet block (as seen in Figure 5.1, one block contains three layers) is considered as a module that can be removed (one at a time). Among these, our RRR approach adopts the latter, the block-wise pruning. Element-wise pruning implementation is the official pruning package of PyTorch [175], where pruned individual parameters are set to zero. Channel-wise pruning is the Torch-Pruning implementation [368], where parameter tensors are effectively slimmed to remove pruned channels. Block-wise pruning skips pruned blocks.

We compare all pruning granularities on the same backbone network: ResNet152 [10]. To measure the potential gain in hardware inference speed, we report inference time in seconds, both on CPU (Intel Xeon Gold 6126) and GPU (GeForce RTX 2080 Ti), across different sparsity levels (Figure 5.4). Sparsity is the fraction of parameters that are removed compared to the full ResNet152. In our experiments, individual parameters or convolutional channels are removed randomly for element-wise pruning and channel-wise pruning, and the sparsity is approximately uniformly distributed among layers; block-wise pruning removes blocks following the block order (Algorithm 2).

The results show that element-wise pruning (yellow curves in Figure 5.4) provides limited speedup, which is expected because the number of parameters does not effectively change, tensor sparsity is not exploited; curve fluctuation corresponds to noise in evaluation. Channel-wise pruning (blue curves in Figure 5.4) provides limited speedup on GPU but effectively accelerates the inference on CPU. Our block-wise pruning strategy (red curves in Figure 5.4) results in the most significant computational gain on both CPU and GPU, particularly when sparsity increases. The advantage of block-wise pruning over channel-wise pruning and the poor result of channel-wise pruning on GPU can be explained by the use of parallel computing in modern CPUs and GPUs. The computation within each layer is parallelized because it involves matrix multiplications [369, 370]. The effect of channel-wise pruning boils down to reducing the parameter tensor size within each layer. Its benefit diminishes when there is more parallelism in the hardware processor, which is the case of GPUs [371, 372]. On the other hand, ResNet152, like many other modern neural networks, consists of sequentially concatenated layers. Computation between layers is blocking; the execution of one layer needs to wait for the result of the previous layer. Removing blocks (hence layers) reduces the number of sequentially blocking computations.

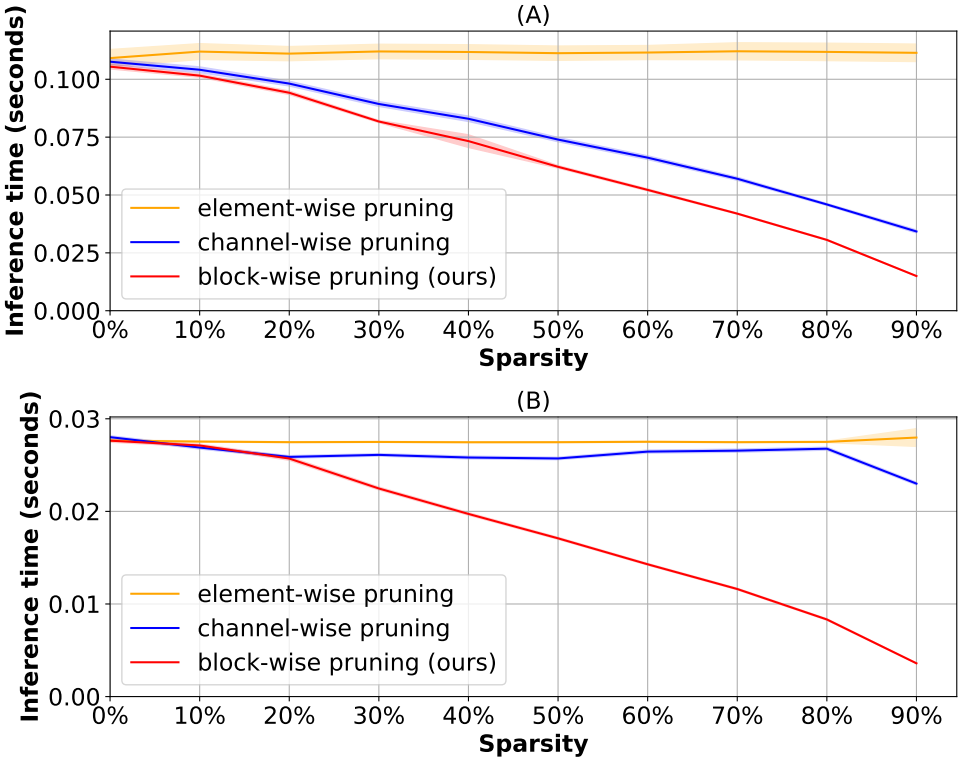


Figure 5.4 – **Inference time of the pruned model.** The upper figure (A) shows the evaluation on CPU; the lower figure (B) shows the evaluation on GPU. Vertical axes show the number of seconds required to run one forward pass for one image; the lower, the better. Horizontal axes denote sparsity. 0% means no pruning, 90% means 90% of parameters are removed. Curves and 95% confidence interval are computed with at least 200 independent runs.

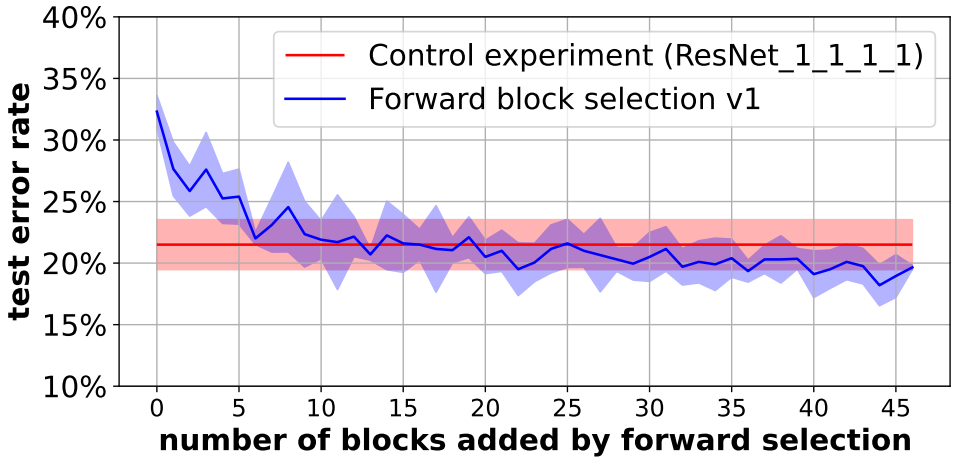


Figure 5.5 – **Optimizing the number of blocks.** Experiments on ICDAR-micro data, applying the forward selection algorithm presented in Algorithm 2 (blue curve). The red horizontal line indicates the final performance of the control experiment. The vertical axis denotes the error rate (1 - accuracy) on the test set; the lower, the better. Curves and 95% confidence intervals are computed with 5 independent runs (5 random seeds).

Reducing the number of blocks

We ran the forward selection algorithm, presented in Algorithm 2, on ICDAR-micro to determine the smallest number of blocks we could keep in our low-resource regime without significantly degrading performance. The results are shown in Figure 5.5. The blue curve is the learning curve of the forward selection algorithm, where each point in the curve is obtained by training the candidate architecture for 300 epochs. In this plot, the stopping criterion is deactivated, so the forward selection algorithm added 47 blocks in the end, which corresponds to 14100 epochs in total. As a control experiment, we compare the performance of the forward selection algorithm with training ResNet_1_1_1_1 for the same overall budget of epochs; the red horizontal line shows the final performance of training ResNet_1_1_1_1 for 14100 epochs. We observe that the final point of the blue curve does not significantly improve upon the red horizontal line.

Hence, these results showed that the model ResNet_1_1_1_1 is sufficient for the given type of tasks and budget, and adding more blocks does not result in significant improvement. Therefore, the recycling/ensembling technique will be applied to ResNet_1_1_1_1 in subsequent experiments.

5.1.3.3 Results from the recycling procedure (varying the number of branches)

Since our approach preserves the total parameter count in the Recycling step, the number of branches provides a trade-off between the number of ensemble members and the capacity of each member. We used ICDAR-micro again to determine an optimal number of branches for our low-resource regime.

In Figure 5.6, we vary the number of branches recycled from our backbone (trained by the naive ensemble approach). The results show that "8 branches" is optimal (performances are quite insensitive to the exact number of branches around that point). In what follows, we use ResNet_1_1_1_1 with 8 branches (splitting starts from conv4_1), denoted as ResNet_1_1_1_1-8_branch or RRR-Net.

We also tried alternative ensembling methods, including bagging [361], Snapshot Ensemble [362], and Fast Geometric Ensemble (FGE) [363], to train branches on the ICDAR-micro data. However, the experiments showed that the naive ensemble approach leads to better performances than the alternative ensembling methods. For this reason, we adopt the naive ensemble approach to train branches.

5.1.3.4 Results on the Meta-Album benchmark

The results on the Meta-Album benchmark [3] are shown in Figure 5.7. We compare 5 models:

- [A] blue Training the last layer of the pre-trained ResNet152
- [B] red Retraining all layers of the pre-trained ResNet152
- [C] green ResNet_1_1_1_1 without splitting
- [D] purple ResNet_1_1_1_1 without splitting but we apply dropout [365] on the last two blocks
- [E] yellow ResNet_1_1_1_1-8_branch (RRR-Net) trained by the naive ensemble approach.

We use models [A] and [B] as baselines. Depending on the distribution shift between the pre-training dataset (ImageNet) and the target dataset, models [A] and [B] can have different relative rankings. Model [E] is our proposed model using ensembling with 8 branches. Models [C] and [D] are controls for [E]. Model [C] is the reduced network without splitting and ensembling; it only has one branch. Model [D] replaces the recycling step of model [E] with dropout [365]: in the last two blocks, we inserted dropout after each activation function (except for the softmax activation). Of several tried dropout rates (0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6), we report the most favorable result for model [D] (dropout rate $= 0.05$).

The Meta-Album benchmark [3] is challenging since it covers a wide diversity of domains and scales. As shown in Figure 5.8, the ranking of different

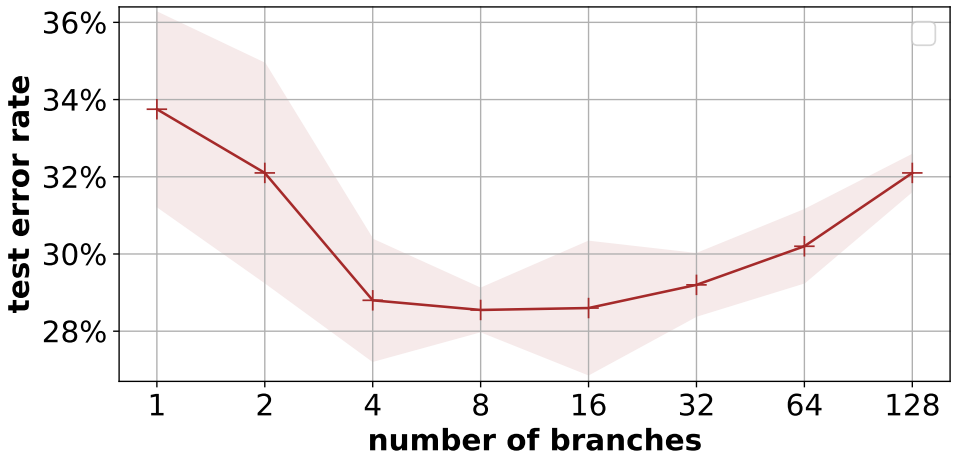


Figure 5.6 – **Optimizing the number of branches.** Experiments on IDCAR-micro data, varying the number of branches. Branches are trained by the naive ensemble approach. In this plot, the number of branches is doubled each time, starting from 1 (ResNet_1_1_1_1 without splitting) up to 128. The optimum (lowest error rate) is at 8 branches. Curves and 95% confidence intervals are computed with 5 independent runs (5 random seeds).

models varies with respect to each dataset. None of the models consistently outperforms all others in all datasets. For example, the overall best baseline model [A] failed on datasets such as ARC and ASL_ALP. The degree of similarity between the target dataset and the pre-training dataset (ImageNet) affects the models' relative performance. This pattern is particularly visible for datasets of macroscopic animals (DOG, AWA), for which last-layer fine-tuning ([A]) clearly outperforms the other models. Pre-trained features are readily available for these datasets, except for the last layer. On the other hand, the character recognition datasets (MD_6, MD_5_T, MD_MIX, MD_5_BIS) are very dissimilar to ImageNet; retraining all layers ([B]) clearly outperforms the other models in these character recognition datasets. In contrast, the proposed model [E] outperforms the other models on a wide variety of datasets, including datasets that are quite different from ImageNet *e.g.,* BCT, PNU, and POT_TUB from the microscopy domain; BRK, TEX, and TEX_ALOT from the texture domain; UCMLU and RSICB from the remote sensing domain.

The diversity of the Meta-Album benchmark [3] made us consider the overall performance of the models under examination. We calculated the average error rates of these 5 models by averaging over the 40 datasets in Meta-Album. The average error rates for these 5 models are 35.8%, 36.0%, 40.3%, 39.9%, 35.5%. Similarly, the median error rates of these 5 models are 32.0%, 34.1%, 38.2%, 36.0%, 29.2%. Our proposed model [E] is found to have the best average and median error rates, as shown in Figure 5.9. To fur-

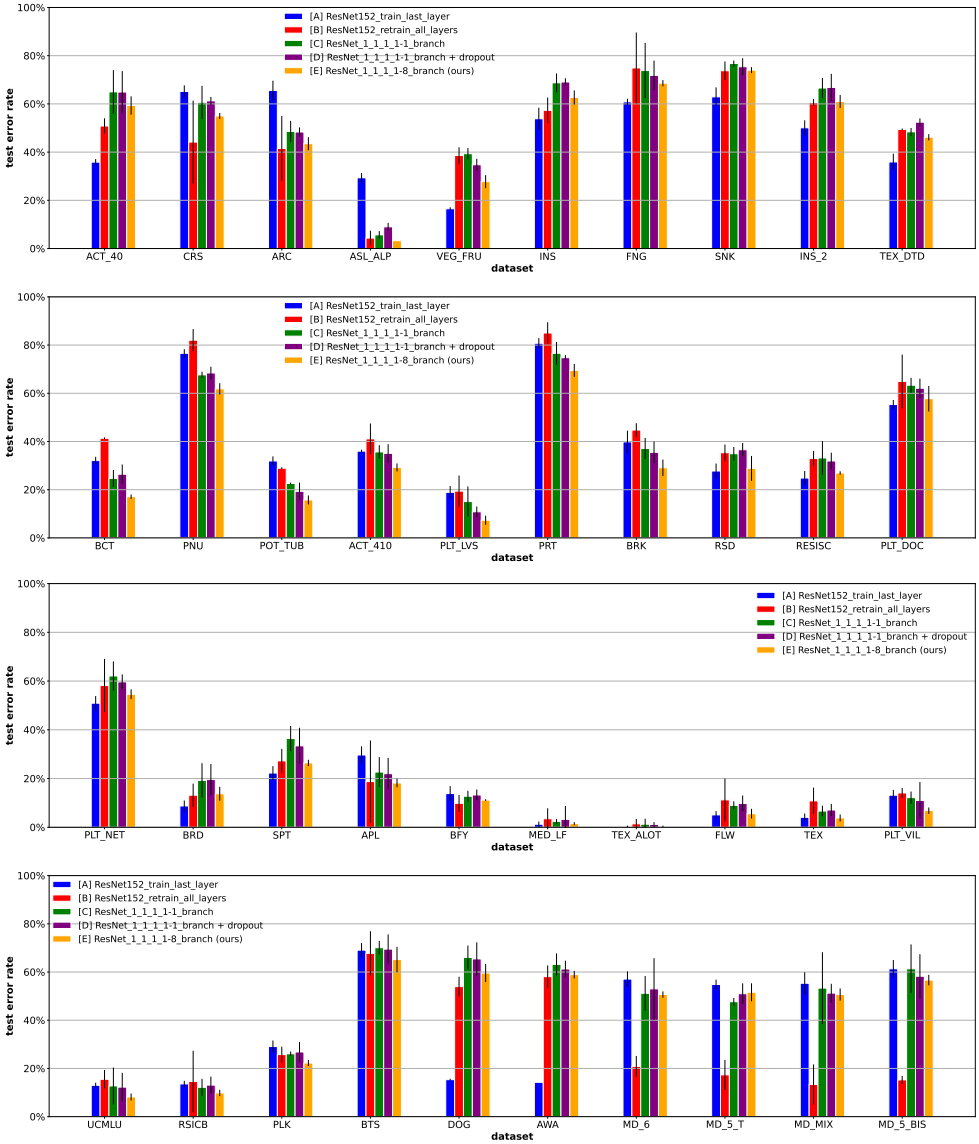


Figure 5.7 – **RRR-Net results on the Meta-Album benchmark.** The vertical axis is the error rate on the test set; the lower, the better. Five models are shown for each dataset: [A] training the last layer (classification head) of the pre-trained ResNet152; [B] retraining all layers of the pre-trained ResNet152; [C] ResNet_1_1_1_1 without splitting; [D] ResNet_1_1_1_1 without splitting but we apply dropout on the last two blocks; [E] ResNet_1_1_1_1-8_branch (RRR-Net) trained by the naive ensemble approach. The 95% confidence intervals are computed with 3 independent runs (3 random seeds).

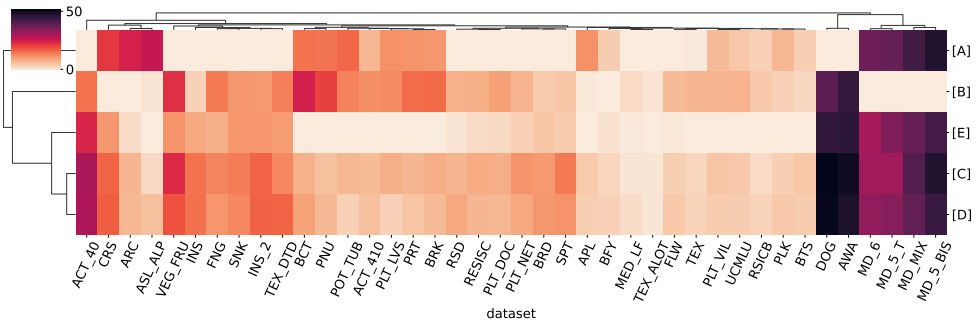


Figure 5.8 – **Hierarchically-clustered heatmap of the model-dataset matrix.** Each column is one dataset. Each row is one model, [E] is our proposed model. The heatmap values are the error rate difference with respect to the best-performing model on the same dataset; the lower, the better.

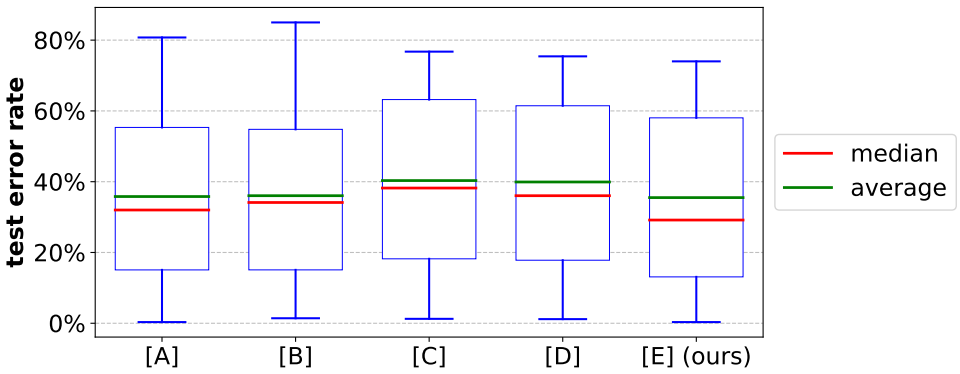


Figure 5.9 – **Box plot of test error rates for each model.** The vertical axis shows the error rate; the lower, the better. Our model [E] has the best average, median, 25% quantile, and worst-case (upper whiskers) error rates. Baselines [A] and [B] have better 75% quantiles.

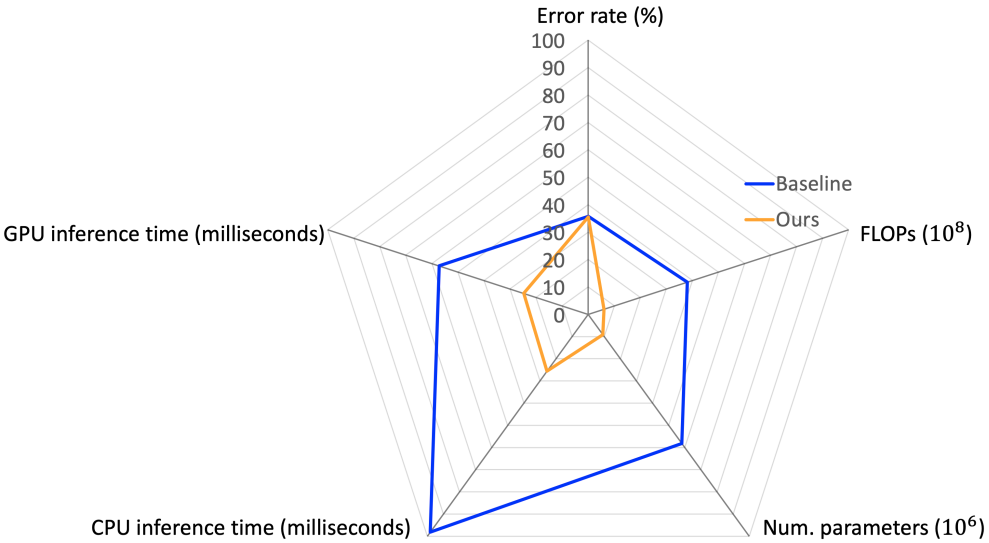


Figure 5.10 – **Multi-criteria comparison.** Our model [E] performs similarly to the overall best baseline [A] while being significantly smaller and faster. Hardware inference time (for one image) is measured with Intel Xeon Gold 6126 and GeForce RTX 2080 Ti.

ther validate these findings, we conducted Wilcoxon signed-rank tests (one-tailed) [373]. These statistical tests indicated that model [E] significantly outperforms model [D] with a p-value of 3×10^{-12} and significantly outperforms model [C] with a p-value of 8×10^{-11}. Model [E] is also found to be significantly better than model [B] with a p-value of 0.036. On the other hand, model [E] performs comparably well to model [A] with a p-value of 0.16. These results indicate that the proposed model [E] performs well on this benchmark with 40 datasets.

Furthermore, we compared the overall best baseline model [A] and our proposed model [E] with respect to five evaluation criteria. These criteria include (1) the average error rate across all 40 datasets in Meta-Album; (2) the number of parameters; (3) the FLOPs (multiply-adds), which indicate the theoretical inference speed; (4) the inference time on CPU; and (5) the inference time on GPU. We represent the comparison as a spider graph depicted in Figure 5.10. Here the baseline error rate (35.8%) corresponds to the average of the blue bars ([A]) in Figure 5.7, which is the strongest baseline on this benchmark. "ours" refers to model [E] ResNet_1_1_1_1-8_branch, whose error rate (35.5%) corresponds to the average of the yellow bars ([E]) in Figure 5.7. We observe that model [E] uses significantly fewer parameters and FLOPs (multiply-adds) and achieves faster inference speed on both CPU and GPU. In summary, Figure 5.10 supports the advantage of our model; it indicates that our model ([E]) performs similarly well to the overall best baseline while being significantly more frugal.

5.1.4 . Discussion

In this section, we studied the benefits of adjusting pre-trained modular neural network architectures for image classification, emphasizing their compactness and efficiency. Our Reduction step can be categorized as a pruning technique performed at the coarse granularity, while our Recycling step partitions pre-trained parameters into multiple branches.

Beyond merely suggesting a model compression technique, this study highlights the strategic reuse of knowledge from pre-trained models, following the "**R**euse, **R**educe, and **R**ecycle" paradigm (Section 5.1.2). Empirically, we showed that by removing a large number of blocks (one form of modularity) from a large pre-trained ResNet model, significant resource savings (in computation and storage) could be achieved without considerable loss in classification accuracy. This highlights the feasibility of repurposing parts of a pre-trained modular neural network instead of using it wholesale. On the other hand, our findings indicate that branching a pre-trained neural network into an ensemble (another form of modularity) could boost classification performance while preserving the same amount of computation and storage.

Overall, the resulting model matches the performance of the overall best baseline (if not better) while being smaller and faster, demonstrating the feasibility of efficiently reusing a pre-trained modular neural network.

It is worth noting that these results are obtained by performing all model selection processes (the Reduction step in Section 5.1.3.2, the Recycling step in Section 5.1.3.3) on a validation dataset (as outlined in Section 5.1.3.1). We have refrained from adjusting the Reduction and Recycling steps for each of the 40 datasets present in the final evaluation benchmark (Meta-Album). This approach strengthens the robustness and relevance of our results.

Building upon the insights presented in this section, we are expanding our exploration to other modular neural network architectures, notably the increasingly popular vision transformer models [119, 374]. In our ongoing research, we are exploring a differentiable method anchored by performance proxies. This pursuit is currently a work in progress.

5.2 . Conclusion

In this chapter, we focused on the modularity in neural network architectures. Pre-trained neural networks are valuable assets in Deep Learning because many computational resources and human efforts have been put into building them. Reusing such assets aligns with our objective: achieving resource-efficient Deep Learning.

The modularity of pre-trained neural networks is evident in their architecture, segmented into modules. These can be convolutional kernels, blocks, and others, depending on granularity. This modular design allows for pruning techniques, aiming to eliminate unnecessary modules, leading to improved utilization efficiency, such as faster inference speed and reduced storage needs.

In this context, we introduced RRR. RRR uses a forward search process to discard unnecessary modules. Additionally, it uses an innovative recycling technique, breaking a pre-trained neural network into an ensemble of modules with the aim of enhancing performance. Results indicated that we can significantly enhance utilization efficiency of a modular pre-trained model while preserving performance, demonstrating the viability of using only parts of a modular model.

In conclusion, this chapter highlights the benefits of leveraging modularity in the architecture of neural networks to enhance utilization efficiency.

6 - Conclusion and future work

In this final chapter, we summarize the main contributions of this thesis, discuss the implications and limitations, and suggest directions for future research.

6.1 . Summary of main contributions

The objective of this thesis is to improve the efficiency of Deep Learning in low-resource scenarios. The principle of modularity, which suggests that complex systems can be decomposed into smaller components or modules, serves as the guiding principle. As it offers various advantages *e.g.,* ease of conceptualization, scalability, module reusability, this concept is prevalent in different fields and disciplines *e.g.,* biology, system design, computer science, mathematics.

Though modularity is intuitively simple to understand as a concept, defining it is not evident. We began by comparing different definitions of modularity and found the foundational definition, which serves as the framework of this thesis. Using this definition, we conducted a literature survey of the modularity principle in Deep Learning. We reviewed its presence in Deep Learning, categorizing it along three axes: data, task, and model, which characterize the life cycle of Deep Learning. Data modularity refers to the observation or creation of data groups for various purposes, task modularity refers to the decomposition of a task into sub-tasks, and model modularity means that the architecture of a neural network system can be decomposed into modules.

In this thesis, we have made contributions in the scope of data modularity and model modularity, which focus on improving the training and utilization efficiency of Deep Learning models, respectively. We have also contributed to the field of task modularity when considering the privacy evaluation process for Deep Learning models.

Our contributions in the field of data modularity include the proposition of OmniPrint, a data synthesizer, and Meta-Album, a meta-dataset, as well as an investigation into the effects of data modularity. This investigation, aided by the proposed OmniPrint and Meta-Album, delves into the relationship between training efficiency and the way to organize training data samples in the context of few-shot learning.

Our contribution in the field of task modularity centers on the conception of LTU, a modular evaluation mechanism tailored to safeguard the privacy of Deep Learning models. By decomposing the evaluation process into independent modules, we ensured the reliability of the assessment.

Our contribution in the field of model modularity focuses on the efficient reuse of a pre-trained modular model to improve utilization efficiency. Specifically, we introduced the "Reuse, Reduce, and Recycle" approach, which applies reduction and recycling techniques to a pre-trained ResNet model. Results show that our approach significantly improves utilization efficiency.

The work presented in this thesis has several implications.

- This thesis highlights the significance of the modularity principle in Deep Learning and demonstrates its benefits in selected subjects of Deep Learning.
- OmniPrint and Meta-Album provide valuable assets for the research community to investigate the effects of data modularity and benchmark few-shot learning, transfer learning, and meta-learning algorithms.
- Our results of the investigation of data modularity reveals that episodic meta-training, an instance of data modularity, could improve the performance of few-shot learning methods if applied judiciously *e.g.,* in combination with monitoring the depth of fine-tuning. This also highlights the importance of careful reuse of pre-trained models.
- The LTU evaluation mechanism echoes the interplay observed in Generative Adversarial Networks and the teacher-student paradigm, highlighting the significance of adversarial learning.
- The "Reuse, Reduce, and Recycle" strategy showcases the potential of reusing segments of pre-trained Deep Learning models rather than their entirety. Unlike the majority of existing pruning methods, our approach operates at a coarse-grained level, pruning entire blocks within the sequential chain of a neural network. This coarse-grained pruning enables a substantial gain in inference speed on modern hardware.

6.2 . Limitations and directions for future work

Data modularity, task modularity and model modularity

As elaborated in Chapter 2, we interpret the modularity principle in Deep Learning through three different yet related axes: data modularity, task modularity, and model modularity, which collectively characterize the life cycle of Deep Learning. In this thesis, we have made contributions related to all three forms of modularity. A potential avenue for future work could involve examining the interrelationships between these three forms of modularity *i.e.,* their mutual influence and how to leverage this interplay to enhance the efficiency of Deep Learning models. For instance, task modularity often correlates with model modularity as sub-tasks are typically managed by separate neural network modules *e.g.,* in Mixture-of-Experts models [45, 124, 48]. It would be worthwhile to explore whether certain forms of data modularity could have a synergy with model modularity. Additionally, it would be beneficial to develop a quantitative measure for the degree of modularity in the context of Deep Learning. This would enable the optimization towards an optimal degree of modularity for some purposes.

Chapter 2 also provides a survey on modularity in Deep Learning. Its findings unveil potential directions for future research. For instance, McNeely-White *et al.* [248] and Bouchacourt *et al.* [249] demonstrated that, given the same training data, learned features exhibit similar properties across models with markedly different architectural inductive biases. This raises a pivotal question: is it still worthwhile to enhance neural network architectures if data predominantly influence learning outcomes [375]? Future studies could corroborate the findings of McNeely-White *et al.* [248] and Bouchacourt *et al.* [249] by extending their research to a broader range of models and datasets. If these results persist, it would be crucial to theoretically ground these results. Another open question for future research involves determining whether neural networks can learn and behave compositionally [67, 376], which necessitates a domain-agnostic approach to test the compositionality of neural networks.

OmniPrint and Meta-Album

In Section 3.1, we introduced OmniPrint, a data synthesizer for printed isolated characters. While it should provide a useful tool to conduct Deep Learning research as is, it can also be customized to become an effective OCR research tool [158]. In some respects, OmniPrint goes beyond state-of-the-art software to generate realistic characters. In particular it has the unique capability of incorporating pre-rasterization transformations, allowing users to distort characters by moving anchor points in the original font vector representation. Still, many synthetic data generators meant to be used for OCR research put emphasis on other aspects, such as more realistic backgrounds,

shadows, sensor aberrations, etc., which have not been our priority. Our modular program interface should facilitate such extensions. Another limitation of OmniPrint is that, so far, emphasis has been put on generating isolated characters, although words or sentences can also be generated. Typeset text is not the focus of this work. Future work could involve enhancing OmniPrint by adding more transformations and utilizing it in a number of other applications, including recognizing printed text in the wild and generating captchas.

In Section 3.2, we presented Meta-Album, a meta-dataset consisting of 40 image classification datasets. While preparing the datasets, we identified several biases, including correlations between class labels and nuisance variables *e.g.,* background, luminosity, contrast, color spectrum, position, and orientation of objects. In this thesis, we avoided correcting such biases, to avoid introducing yet more bias. Future work could involve examining the problem of bias [377, 378] in these datasets and potentially organizing competitions where training and test data exhibit intentional distribution shifts.

Episodic meta-training in few-shot learning

In Section 3.3.1, we observed that, in the context of few-shot learning [98], having internally cohesive metadata-based episodes, where data samples are more similar within episodes than between episodes for both meta-training and meta-test, simplifies the learning problem for few-learning algorithms. This line of research could be extended to explore the influence of cohesion levels in meta-training episodes on the performance and efficiency of few-shot learning algorithms, while keeping the meta-test episodes fixed.

Section 3.3.2 demonstrates that episodic meta-training, an instance of data modularity, can yield superior performances compared to batch meta-training. This analysis was conducted using classical few-shot learning algorithms, such as MAML [98] and Prototypical Networks [99]. We did not extend this study to the top-ranked solutions of the competition as those solutions are too complex to modify for a fair comparison [5]. In future competitions, we could encourage participants to propose solutions using a more unified API. This would facilitate the ablation studies in Section 3.3.2, allowing further investigation into the effects of episodic meta-training.

In Section 3.3.2, we empirically investigated the impacts of episodic meta-training. It would also be interesting to theoretically examine the benefits of episodic meta-training, which could guide the design of novel few-shot learning algorithms in a more principled manner.

LTU for assessing privacy concerns

In Chapter 4, we proposed the LTU evaluation mechanism for assessing the robustness of Deep Learning models against membership inference attacks. In this thesis, we illustrated its usage using a ResNet model defended

by either supervised learning or unsupervised domain adaptation, with the attack strategy rooted in gradient descent. Potential future avenues include the exploration of diverse defender and attacker strategies, as well as the possibility of organizing membership inference attack competitions leveraging the LTU evaluation mechanism.

RRR for pre-trained models

In Chapter 5, we proposed the RRR approach to improve the utilization efficiency of a pre-trained model. In this thesis, we demonstrated its utilities on ResNet [10]. A promising future direction involves exploring its potential with other neural network architectures and proposing necessary adaptations.

On the other hand, the RRR approach has an ensemble nature as its recycling step splits pre-trained parameters into parallel branches. A future direction could involve improving the ensemble training method with the objective of improving classification performance [230]. Moreover, the ensemble nature of the RRR approach lends itself to the predictive uncertainty computation [379, 380], and this combination could be explored in future work.

6.3 . Conclusion

The emphasis on modularity throughout this thesis highlights its significance in Deep Learning. By breaking down various aspects of Deep Learning into separate modules, we have illustrated how complex challenges can be addressed, whether by optimizing data organization, refining tasks, or improving model architectures.

The advantages of this modular perspective are clear. It promotes adaptability, reduces interference, and enhances the reusability of existing assets. Embracing modularity could pave the way for more efficient Deep Learning.

Bibliography

[1] Haozhe Sun and Isabelle Guyon. Modularity in deep learning: A survey. In Kohei Arai, editor, *Intelligent Computing*, pages 561–595, Cham, 2023. Springer Nature Switzerland.

[2] Haozhe Sun, Wei-Wei Tu, and Isabelle Guyon. OmniPrint: A Configurable Printed Character Synthesizer. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[3] Ihsan Ullah, Dustin Carrion, Sergio Escalera, Isabelle M Guyon, Mike Huisman, Felix Mohr, Jan N. van Rijn, Haozhe Sun, Joaquin Vanschoren, and Phan Anh Vu. Meta-album: Multi-domain meta-dataset for few-shot image classification. In *Thirty-Sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[4] Adrian El Baz, Ihsan Ullah, Edesio Alcobaça, André C. P. L. F. Carvalho, Hong Chen, Fabio Ferreira, Henry Gouk, Chaoyu Guan, Isabelle Guyon, Timothy Hospedales, Shell Hu, Mike Huisman, Frank Hutter, Zhengying Liu, Felix Mohr, Ekrem Öztürk, Jan N. van Rijn, Haozhe Sun, Xin Wang, and Wenwu Zhu. Lessons learned from the NeurIPS 2021 MetaDL challenge: Backbone fine-tuning without episodic meta-learning dominates for few-shot learning image classification. In Douwe Kiela, Marco Ciccone, and Barbara Caputo, editors, *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*, volume 176 of *Proceedings of Machine Learning Research*, pages 80–96. PMLR, December 2022.

[5] Dustin Carrión-Ojeda, Mahbubul Alam, Sergio Escalera, Ahmed Farahat, Dipanjan Ghosh, Teresa Gonzalez Diaz, Chetan Gupta, Isabelle Guyon, Joël Roman Ky, Xian Yeow Lee, Xin Liu, Felix Mohr, Manh Hung Nguyen, Emmanuel Pintelas, Stefan Roth, Simone Schaub-Meyer, Haozhe Sun, Ihsan Ullah, Joaquin Vanschoren, Lasitha Vidyaratne, Jiamin Wu, and Xiaotian Yin. NeurIPS'22 cross-domain MetaDL challenge: Results and lessons learned. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 50–72. PMLR, November 2022.

[6] Joseph Pedersen, Rafael Muñoz-Gómez, Jiangnan Huang, Haozhe Sun, Wei-Wei Tu, and Isabelle Guyon. LTU attacker for membership inference. *Algorithms*, 15(7), 2022.

[7] Haozhe Sun, Isabelle Guyon, Felix Mohr, and Hedi Tabia. RRR-Net: Reusing, reducing, and recycling a deep backbone network. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9, 2023.

[8] 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, 2009.

[9] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[10] 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.

[11] Guillaume Chevalier. Long short-term memory (LSTM cell). *Wikipedia*, September 2022.

[12] Lyken17/pytorch-OpCounter: Count the MACs / FLOPs of your PyTorch model. https://github.com/Lyken17/pytorch-OpCounter.

[13] Herbert A. Simon. The Architecture of Complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, 1962.

[14] Herbert A. Simon and Albert Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29(2):111–138, 1961.

[15] Peter Gentile. Theory of Modularity, a Hypothesis. *Procedia Computer Science*, 20, December 2013.

[16] Yue Shao and Victor M. Zavala. Modularity measures: Concepts, computation, and applications to manufacturing systems. *AIChE Journal*, 66(6):e16965, 2020.

[17] Vladimir Modrak and Zuzana Soltysova. Development of the Modularity Measure for Assembly Process Structures. *Mathematical Problems in Engineering*, 2021:e4900748, December 2021.

[18] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, et al. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 75:284–298, 2017.

[19] Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity*, volume 1. Cambridge, MA: MIT Press, first edition, 1999.

[20] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[21] Jeremy Avigad. Modularity in mathematics. *The Review of Symbolic Logic*, 13(1):47–79, March 2020.

[22] Nicholas Bourbaki. The Architecture of Mathematics. *The American Mathematical Monthly*, 57(4):221–232, April 1950.

[23] Scott C. Burgess, Will H. Ryan, Neil W. Blackstone, Peter J. Edmunds, Mia O. Hoogenboom, Don R. Levitan, and Janie L. Wulff. Metabolic scaling in modular animals. *Invertebrate Biology*, 136(4):456–472, 2017.

[24] Michel A Hofman. Evolution of the human brain: When bigger is better. *Frontiers in neuroanatomy*, 8:15, 2014.

[25] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences*, 280(1755):20122863, 2013.

[26] Jerry A Fodor. *The modularity of mind*. MIT press, 1983.

[27] Philip Robbins. Modularity of Mind. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2017 edition, 2017.

[28] H. Clark Barrett and Robert Kurzban. Modularity in cognition: Framing the debate. *Psychological Review*, 113(3):628–647, July 2006.

[29] Willem E. Frankenhuis and Annemie Ploeger. Evolutionary Psychology Versus Fodor: Arguments For and Against the Massive Modularity Hypothesis. *Philosophical Psychology*, 20(6):687–710, December 2007.

[30] Leda Cosmides and John Tooby. Origins of domain specificity: The evolution of functional organization. In Lawrence A. Hirschfeld and Susan A. Gelman, editors, *Mapping the Mind: Domain Specificity in Cognition and Culture*, pages 85–116. Cambridge University Press, Cambridge, 1994.

[31] Zenon Pylyshyn. Is vision continuous with cognition?: The case for cognitive impenetrability of visual perception. *Behavioral and Brain Sciences*, 22(3):341–365, June 1999.

[32] Martin Ford. *Architects of Intelligence: The Truth about AI from the People Building It*. Packt Publishing, Birmingham, UK, first published: november 2018 edition, 2018.

[33] Ray Kurzweil. *How to Create a Mind: The Secret of Human Thought Revealed*. Penguin Books, USA, 2013.

[34] Leda Cosmides and John Tooby. Cognitive Adaptations for Social Exchange. *undefined*, pages 163–228, 1992.

[35] Stewart Shipp. Structure and function of the cerebral cortex. *Current Biology*, 17(12):R443–R449, 2007.

[36] Manuel F Casanova and Emily L Casanova. The modular organization of the cerebral cortex: Evolutionary significance and possible links to neurodevelopmental conditions. *Journal of Comparative Neurology*, 527(10):1720–1730, 2019.

[37] American Association for Research into Nervous, Mental Diseases, S Zeki, and A Bartels. The autonomy of the visual systems and the modularity of conscious vision. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 353(1377):1911–1914, 1998.

[38] Jose B Pereira-Leal, Emmanuel D Levy, and Sarah A Teichmann. The origins and evolution of functional modules: Lessons from protein complexes. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 361(1467):507–517, 2006.

[39] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[40] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. Local modularity measure for network clusterizations. *Physical Review E*, 72(5):056107, November 2005.

[41] Timothée Poisot. An a posteriori measure of network modularity. *F1000Research*, 2:130, December 2013.

[42] Daniel Gómez, J. Tinguaro Rodríguez, Javier Yáñez, and Javier Montero. A new modularity measure for Fuzzy Community detection problems based on overlap and grouping functions. *International Journal of Approximate Reasoning*, 74:88–107, July 2016.

[43] G. Auda and M. Kamel. Modular neural networks a survey. *International journal of neural systems*, 9 2:129–51, 1999.

[44] Farooq Azam. *Biologically Inspired Modular Neural Networks*. PhD thesis, Virginia Tech, May 2000.

[45] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, March 1991.

[46] J. Reisinger, K. Stanley, and R. Miikkulainen. Evolving Reusable Neural Modules. In *GECCO*, 2004.

[47] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural Module Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48, Las Vegas, NV, USA, June 2016. IEEE.

[48] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[49] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. PathNet: Evolution Channels Gradient Descent in Super Neural Networks. *arXiv:1701.08734 [cs]*, January 2017.

[50] Louis Kirsch, Julius Kunze, and David Barber. Modular Networks: Learning to Decompose Neural Computation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[51] Ferran Alet, Tomás Lozano-Pérez, and Leslie P. Kaelbling. Modular meta-learning. *arXiv:1806.10166 [cs, stat]*, May 2019.

[52] Mohammed Amer and Tomas Maul. A Review of Modularization Techniques in Artificial Neural Networks. *Artificial Intelligence Review*, 52, June 2019.

[53] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks. In *International Conference on Learning Representations*, 2021.

[54] Daniel Filan, Stephen Casper, Shlomi Hod, Cody Wild, Andrew Critch, and Stuart Russell. Clusterability in Neural Networks. *arXiv:2103.03386 [cs]*, March 2021.

[55] Michael Chang, Sid Kaushik, Sergey Levine, and Tom Griffiths. Modularity in Reinforcement Learning via Algorithmic Independence in Credit Assignment. In *International Conference on Machine Learning*, pages 1452–1462. PMLR, July 2021.

[56] Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient Continual Learning with Modular Networks and Task-Driven Priors. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[57] Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular Deep Learning, February 2023.

[58] Badih Ghazi, Rina Panigrahy, and Joshua Wang. Recursive Sketches for Modular Deep Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2211–2220. PMLR, May 2019.

[59] J. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1872–1877 vol.2, May 2002.

[60] Gunter P. Wagner and Lee Altenberg. Perspective: Complex Adaptations and the Evolution of Evolvability. *Evolution*, 50(3):967–976, 1996.

[61] Jerry A. Fodor. *The Mind Doesn't Work That Way: The Scope and Limits of Computational Psychology*. MIT Press, 2000.

[62] Miriam L Zelditch and Anjali Goswami. What does modularity mean? *Evolution & Development*, 23(5):377–403, 2021.

[63] Gabriel Béna and Dan F. M. Goodman. Extreme sparsity gives rise to functional specialization. *arXiv:2106.02626 [cs, q-bio]*, June 2021.

[64] Shane Legg, Marcus Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

[65] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and machines*, 17:391–444, 2007.

[66] Sahil Verma and Julia Rubin. Fairness definitions explained. In *Proceedings of the International Workshop on Software Fairness*, pages 1–7, Gothenburg Sweden, May 2018. ACM.

[67] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

[68] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, October 2018.

[69] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. *arXiv:2004.05439 [cs, stat]*, November 2020.

[70] ALBRECHT Schmidt and ZUHAIR Bandar. Modularity-a concept for new neural network architectures. In *Proc. IASTED International Conf. Computer Systems and Applications*, pages 26–29. Citeseer, 1998.

[71] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[72] Dominik Janzing and Bernhard Schölkopf. Causal Inference Using the Algorithmic Markov Condition. *IEEE Trans. Inf. Theor.*, 56(10):5168–5194, October 2010.

[73] Melissa Schilling. Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity. *Academy of Management Review*, 25, April 2000.

[74] Joost Huizinga, Jeff Clune, and Jean-Baptiste Mouret. Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 697–704, 2014.

[75] Anirudh Goyal ALIAS PARTH GOYAL, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael C Mozer,

and Yoshua Bengio. Neural production systems. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25673–25687. Curran Associates, Inc., 2021.

[76] Tomer Galanti and Lior Wolf. On the Modularity of Hypernetworks. *arXiv:2002.10006 [cs, stat]*, November 2020.

[77] Jianan Wang, Eren Sezener, David Budden, Marcus Hutter, and Joel Veness. A Combinatorial Perspective on Transfer Learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 918–929. Curran Associates, Inc., 2020.

[78] Julian Zaidi, Jonathan Boilard, Ghyslain Gagnon, and Marc-André Carbonneau. Measuring Disentanglement: A Review of Metrics. *arXiv:2012.09276 [cs]*, January 2021.

[79] Anirudh Goyal, Aniket Rajiv Didolkar, Alex Lamb, Kartikeya Badola, Nan Rosemary Ke, Nasim Rahaman, Jonathan Binas, Charles Blundell, Michael Curtis Mozer, and Yoshua Bengio. Coordination among neural modules through a shared global workspace. In *International Conference on Learning Representations*, 2022.

[80] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. Hierarchical agglomerative clustering. In Werner Dubitzky, Olaf Wolkenhauer, Kwang-Hyun Cho, and Hiroki Yokota, editors, *Encyclopedia of Systems Biology*, pages 886–887. Springer New York, New York, NY, 2013.

[81] Palash B Pal. *An Introductory Course of Particle Physics*. Taylor & Francis, 2014.

[82] Rangeet Pan and Hridesh Rajan. On Decomposing a Deep Neural Network into Modules. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 889–900, New York, NY, USA, 2020. Association for Computing Machinery.

[83] Rangeet Pan and Hridesh Rajan. Decomposing Convolutional Neural Networks into Reusable and Replaceable Modules. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022)*, December 2021.

[84] Edoardo M. Ponti, Alessandro Sordoni, Yoshua Bengio, and Siva Reddy. Combining Modular Skills in Multitask Learning, March 2022.

[85] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, December 2015.

[86] Yutian Chen, Abram L Friesen, Feryal Behbahani, Arnaud Doucet, David Budden, Matthew Hoffman, and Nando de Freitas. Modular meta-learning with shrinkage. *Advances in Neural Information Processing Systems*, 33:2858–2869, 2020.

[87] Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *International Conference on Machine Learning*, pages 4036–4044. PMLR, 2018.

[88] Ignasi Clavera, David Held, and Pieter Abbeel. Policy transfer via modularity and reward guiding. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1537–1544. IEEE, 2017.

[89] Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic inference with neural interpreters. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10985–10998. Curran Associates, Inc., 2021.

[90] Trang Nguyen, Amin Mansouri, Kanika Madan, Khuong Nguyen, Kartik Ahuja, Dianbo Liu, and Yoshua Bengio. Reusable Slotwise Mechanisms, February 2023.

[91] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021.

[92] Licheng Yu, Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu, Mohit Bansal, and Tamara L Berg. MAttNet: Modular attention network for referring expression comprehension. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1307–1315, 2018.

[93] Felipe Almeida and Geraldo Xexéo. Word Embeddings: A Survey, January 2019.

[94] Eunjeong Koh and Shlomo Dubnov. Comparison and Analysis of Deep Audio Embeddings for Music Emotion Recognition, April 2021.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[96] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[97] Guy Hacohen and Daphna Weinshall. On The Power of Curriculum Learning in Training Deep Networks. In Kamalika Chaudhuri and Ruslan

Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2535–2544. PMLR, June 2019.

[98] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, July 2017.

[99] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical Networks for Few-shot Learning. *arXiv:1703.05175 [cs, stat]*, June 2017.

[100] Jun-Fei Qiao, Xi Meng, Wen-Jing Li, and Bogdan M. Wilamowski. A novel modular RBF neural network based on a brain-like partition method. *Neural Computing and Applications*, 32(3):899–911, February 2020.

[101] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations*, 2020.

[102] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv:1411.1792 [cs]*, November 2014.

[103] Y. LeCun, Fu Jie Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–104 Vol.2, June 2004.

[104] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, second edition, 2009.

[105] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4212–4221. PMLR, June 2019.

[106] Likang Wu, Zhi Li, Hongke Zhao, Qi Liu, Jun Wang, Mengdi Zhang, and Enhong Chen. Learning the implicit semantic representation on graph-structured data. In *International Conference on Database Systems for Advanced Applications*, pages 3–19. Springer, 2021.

[107] Ulrike von Luxburg, Robert C. Williamson, and Isabelle Guyon. Clustering: Science or Art? In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 65–79. JMLR Workshop and Conference Proceedings, June 2012.

[108] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence*, 41(2):423–443, 2018.

[109] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6904–6913, 2017.

[110] Madina Abdrakhmanova, Askat Kuzdeuov, Sheikh Jarju, Yerbolat Khassanov, Michael Lewis, and Huseyin Atakan Varol. Speakingfaces: A large-scale multimodal dataset of voice commands with visual and thermal video streams. *Sensors*, 21(10):3465, 2021.

[111] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[112] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[113] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *ICLR*, 2017.

[114] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40, 2021.

[115] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, August 2003.

[116] John S Rose. *A Course on Group Theory*. Courier Corporation, 1994.

[117] Shuxiao Chen, Edgar Dobriban, and Jane Lee. A group-theoretic framework for data augmentation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21321–21333. Curran Associates, Inc., 2020.

[118] Tian Jin and Seokin Hong. Split-CNN: Splitting Window-Based Operations in Convolutional Neural Networks for Memory System Optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 835–847, New York, NY, USA, 2019. Association for Computing Machinery.

[119] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani,

Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

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

[121] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. Efficient-capsnet: Capsule network with self-attention routing. *Scientific reports*, 11(1):1–13, 2021.

[122] David Krueger, Tegan Maharaj, Janos Kramar, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Christopher Pal. Zoneout: Regularizing RNNs by randomly preserving hidden activations. In *International Conference on Learning Representations*, 2017.

[123] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[124] Robert A Jacobs, Michael I Jordan, and Andrew G Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive science*, 15(2):219–250, 1991.

[125] Oleksiy Ostapenko, Pau Rodriguez, Alexandre Lacoste, and Laurent Charlin. Attention for compositional modularity. In *NeurIPS '22 Workshop on All Things Attention: Bridging Different Perspectives on Attention*, 2022.

[126] Elliot Meyerson and Risto Miikkulainen. Modular universal reparameterization: Deep multi-task learning across diverse domains. *Advances in Neural Information Processing Systems*, 32, 2019.

[127] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2169–2176. IEEE, 2017.

[128] Christopher Simpkins and Charles Isbell. Composable modular reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4975–4982, 2019.

[129] Senthil Purushwalkam, Maximilian Nickel, Abhinav Gupta, and Marc'Aurelio Ranzato. Task-driven modular networks for zero-shot compositional learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3593–3602, 2019.

[130] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: Intrinsically motivated modular multi-goal reinforcement learning. In *International Conference on Machine Learning*, pages 1331–1340. PMLR, 2019.

[131] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. SplitNet: Learning to Semantically Split Deep Networks for Parameter Reduction and Model Parallelization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1866–1874. PMLR, July 2017.

[132] Hiroaki Kingetsu, Kenichi Kobayashi, and Taiji Suzuki. Neural Network Module Decomposition and Recomposition, December 2021.

[133] Asmaa Abbas, Mohammed Abdelsamea, and Mohamed Gaber. DeTraC: Transfer Learning of Class Decomposed Medical Images in Convolutional Neural Networks. *IEEE Access*, PP:1–1, April 2020.

[134] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. Editable Neural Networks. In *International Conference on Learning Representations*, 2019.

[135] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast Model Editing at Scale. *arXiv:2110.11309 [cs]*, October 2021.

[136] Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. BeliefBank: Adding Memory to a Pre-Trained Language Model for a Systematic Notion of Belief. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8849–8861, 2021.

[137] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Memory-Based Model Editing at Scale. In *International Conference on Machine Learning*, 2022.

[138] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

[139] Birhanu Belay, Tewodros Habtegebrial, Marcus Liwicki, Gebeyehu Belay, and Didier Stricker. Factored Convolutional Neural Network for Amharic Character Image Recognition. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2906–2910, 2019.

[140] Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. DeFormer: Decomposing Pre-trained Transformers for Faster Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4497, Online, July 2020. Association for Computational Linguistics.

[141] Changxing Ding and Dacheng Tao. Trunk-branch ensemble convolutional neural networks for video-based face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):1002–1014, 2017.

[142] Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-Learning Symmetries by Reparameterization. *arXiv:2007.02933 [cs, stat]*, October 2020.

[143] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, Las Vegas, NV, USA, June 2016. IEEE.

[144] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014.

[145] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Computer Vision (ICCV), 2017 IEEE International Conference On*, 2017.

[146] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[147] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, November 2014.

[148] Zhenxing Zhang and Lambert Schomaker. DTGAN: Dual attention generative adversarial networks for text-to-image generation. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[149] Zhenxing Zhang and Lambert Schomaker. DiverGAN: An efficient and effective single-stage framework for diverse text-to-image generation. *Neurocomputing*, 473:182–198, 2022.

[150] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *arXiv:1703.01780 [cs, stat]*, April 2018.

[151] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[152] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[153] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu.

Asynchronous Methods for Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1602.01783, February 2016.

[154] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *arXiv e-prints*, page arXiv:1708.05144, August 2017.

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

[156] G Ranganathan et al. A study to find facts behind preprocessing on deep learning algorithms. *Journal of Innovative Image Processing (JIIP)*, 3(01):66–74, 2021.

[157] Noman Islam, Zeeshan Islam, and Nazia Noor. A Survey on Optical Character Recognition System. *arXiv:1710.05703 [cs]*, October 2017.

[158] Xiaoxue Chen, Lianwen Jin, Yuanzhi Zhu, Canjie Luo, and Tianwei Wang. Text Recognition in the Wild: A Survey. *arXiv:2005.03492 [cs]*, December 2020.

[159] Isabelle Guyon, Lambert Schomaker, Rejean Plamondon, Mark Liberman, and Stan Janet. UNIPEN project of on-line data exchange and recognizer benchmarks. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 29–33 vol.2, 1994.

[160] Louis Vuurpijl and Lambert Schomaker. Two-stage character classification: A combined approach of clustering and support vector classifiers. In *Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, September 11-13 2000, Amsterdam*. Nijmegen: International Unipen Foundation, 2000.

[161] Dan Jurafsky and James H Martin. Speech and language processing (3rd draft ed.), 2019.

[162] Baoguang Shi, Xiang Bai, and Cong Yao. Script identification in the wild via discriminative convolutional neural network. *Pattern Recognition*, 52:448–458, April 2016.

[163] Jing Huang, Guan Pang, Rama Kovvuri, Mandy Toh, Kevin J Liang, Praveen Krishnan, Xi Yin, and Tal Hassner. A multiplexed network for end-to-end, multilingual OCR. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4547–4557, 2021.

[164] Richard G Casey and Eric Lecolinet. A survey of methods and strategies in character segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 18(7):690–706, 1996.

[165] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[166] Amit Choudhary, Rahul Rishi, and Savita Ahlawat. A new character segmentation approach for off-line cursive handwritten words. *Procedia Computer Science*, 17:88–95, 2013.

[167] Amandeep Kaur, Seema Baghla, and Sunil Kumar. Study of various character segmentation techniques for handwritten off-line cursive words: A review. *International Journal of Advances in Science Engineering and Technology*, 3(3):154–158, 2015.

[168] Guolei Sun, Thomas Probst, Danda Pani Paudel, Nikola Popovic, Menelaos Kanakis, Jagruti Patel, Dengxin Dai, and Luc Van Gool. Task Switching Network for Multi-task Learning. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8271–8280, Montreal, QC, Canada, October 2021. IEEE.

[169] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[170] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[171] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. *arXiv:1707.07012 [cs, stat]*, April 2018.

[172] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search. In *Automatic Machine Learning: Methods, Systems, Challenges*, pages 69–86, 2019.

[173] Ming Li and Paul M.B. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, third edition, 2008.

[174] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[175] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit

Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

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

[177] Accelerate Fast Math with Intel® oneAPI Math Kernel Library. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html.

[178] Pralhad Gavali and J. Saira Banu. Chapter 6 - deep convolutional neural network for image classification on CUDA platform. In Arun Kumar Sangaiah, editor, *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, pages 99–122. Academic Press, 2019.

[179] Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3(2):2, 2017.

[180] William Fedus, Jeff Dean, and Barret Zoph. A Review of Sparse Expert Models in Deep Learning, September 2022.

[181] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 120–134, 2022.

[182] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[183] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay,

Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311 [cs]*, April 2022.

[184] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Dan Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, Brennan Saeta, Parker Schuh, Ryan Sepassi, Laurent El Shafey, Chandramohan A. Thekkath, and Yonghui Wu. Pathways: Asynchronous Distributed Dataflow for ML. *arXiv:2203.12533 [cs]*, March 2022.

[185] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[186] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[187] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model, February 2022.

[188] Yamuna Krishnamurthy and Chris Watkins. Interpretability in gated modular neural networks. In *eXplainable AI Approaches for Debugging*

and Diagnosis., 2021.

[189] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432 [cs]*, August 2013.

[190] Alexander Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Reuse of neural modules for general video game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[191] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[192] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *European Conference on Computer Vision*, pages 435–442. Springer, 2016.

[193] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pretraining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[194] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision*, volume 115, pages 211–252. Springer, 2015.

[195] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 843–852, 2017.

[196] I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *CoRR*, abs/1905.00546, 2019.

[197] Vanessa D'Amario, Tomotake Sasaki, and Xavier Boix. How Modular should Neural Module Networks Be for Systematic Generalization? In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[198] Riashat Islam, Hongyu Zang, Anirudh Goyal, Alex Lamb, Kenji Kawaguchi, Xin Li, Romain Laroche, Yoshua Bengio, and Remi Tachet Des Combes. Discrete Factorial Representations as an Abstraction for Goal Conditioned Reinforcement Learning, October 2022.

[199] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.

[200] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. CTL++: Evaluating Generalization on Never-Seen Compositional Patterns of Known Functions, and Compatibility of Neural Representations. In *Proc. Conf.*

on Empirical Methods in Natural Language Processing (EMNLP), December 2022.

[201] Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[202] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR, 2018.

[203] Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.

[204] Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. Automatically composing representation transformations as a means for generalization. In *International Conference on Learning Representations*, 2019.

[205] Jürgen Schmidhuber. Towards compositional learning in dynamic networks. *Technical University of Munich (Technical Report FKI-129-90)*, 1990.

[206] Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a Modular Architecture Enough?, 2022.

[207] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

[208] Edoardo Ponti. *Inductive Bias and Modular Design for Sample-Efficient Neural Language Learning*. PhD thesis, University of Cambridge, 2021.

[209] Dieuwke Hupkes, Mario Giulianelli, Verna Dankers, Mikel Artetxe, Yanai Elazar, Tiago Pimentel, Christos Christodoulopoulos, Karim Lasri, Naomi Saphra, Arabella Sinclair, Dennis Ulmer, Florian Schottmann, Khuyagbaatar Batsuren, Kaiser Sun, Koustuv Sinha, Leila Khalatbari, Maria Ryskina, Rita Frieske, Ryan Cotterell, and Zhijing Jin. State-of-the-art generalisation research in NLP: A taxonomy and review, October 2022.

[210] Ivan I Vankov and Jeffrey S Bowers. Training neural networks to encode symbols enables combinatorial generalization. *Philosophical Transactions of the Royal Society B*, 375(1791):20190309, 2020.

[211] João Loula, Marco Baroni, and Brenden M. Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *BlackboxNLP@EMNLP*, pages 108–114, 2018.

[212] Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing Networks and the Challenges of Modular and Compositional Computation, April 2019.

[213] Brenden M. Lake. Compositional generalization through meta sequence-to-sequence learning. *arXiv:1906.05381 [cs]*, October 2019.

[214] Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D. Manning. Characterizing Intrinsic Compositionality in Transformers with Tree Projections, November 2022.

[215] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2019.

[216] Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32, 2019.

[217] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-Task Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5824–5836. Curran Associates, Inc., 2020.

[218] Menelaos Kanakis, David Bruggemann, Suman Saha, Stamatios Georgoulis, Anton Obukhov, and Luc Van Gool. Reparameterizing convolutions for incremental multi-task learning without task interference. In *European Conference on Computer Vision*, pages 689–707. Springer, 2020.

[219] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive Single-Tasking of Multiple Tasks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1851–1860, Long Beach, CA, USA, June 2019. IEEE.

[220] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv:1606.04671 [cs]*, September 2016.

[221] Alexander Terekhov, Guglielmo Montone, and J. O'Regan. Knowledge Transfer in Deep Block-Modular Neural Networks. In *Biomimetic and Biohybrid Systems*, pages 268–279. Springer, July 2015.

[222] Ark Anderson, Kyle Shaffer, Artem Yankov, Court D. Corley, and Nathan O. Hodas. Beyond Fine Tuning: A Modular Approach to Learning on Small Data, November 2016.

[223] Khurram Javed and Martha White. Meta-Learning Representations for Continual Learning. *arXiv:1905.12588 [cs, stat]*, October 2019.

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

[225] Wickliffe C. Abraham and Anthony Robins. Memory retention – the synaptic stability versus plasticity dilemma. *Trends in Neurosciences*, 28(2):73–78, February 2005.

[226] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. Neural Modularity Helps Organisms Evolve to Learn New Skills without Forgetting Old Skills. *PLOS Computational Biology*, 11(4):e1004128, April 2015.

[227] Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34:22443–22456, 2021.

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

[229] Randall Balestriero and Yann LeCun. POLICE: Provably Optimal Linear Constraint Enforcement for Deep Neural Networks, November 2022.

[230] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC press, 2012.

[231] 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.

[232] Gamaleldin F. Elsayed, Prajit Ramachandran, Jonathon Shlens, and Simon Kornblith. Revisiting Spatial Invariance with Low-Rank Local Connectivity. *arXiv:2002.02959 [cs, stat]*, August 2020.

[233] Taco S. Cohen and Max Welling. Group Equivariant Convolutional Networks. *arXiv:1602.07576 [cs, stat]*, June 2016.

[234] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting Cyclic Symmetry in Convolutional Neural Networks. *arXiv:1602.02660 [cs]*, May 2016.

[235] Taco S. Cohen and Max Welling. Steerable CNNs. *arXiv:1612.08498 [cs, stat]*, December 2016.

[236] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic Networks: Deep Translation and Rotation Equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

[237] Hongyang Gao and Shuiwang Ji. Efficient and Invariant Convolutional Neural Networks for Dense Prediction. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 871–876, 2017.

[238] Erik J. Bekkers, Maxime W. Lafarge, Mitko Veta, Koen AJ Eppenhof, Josien PW Pluim, and Remco Duits. Roto-Translation Covariant Convolutional Networks for Medical Image Analysis. *arXiv:1804.03393 [cs, math]*, June 2018.

[239] Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning Steerable Filters for Rotation Equivariant CNNs. *arXiv:1711.07289 [cs]*, March 2018.

[240] Maurice Weiler and Gabriele Cesa. General $E(2)$-Equivariant Steerable CNNs. *arXiv:1911.08251 [cs, eess]*, April 2021.

[241] Sifre Laurent. Rigid-motion scattering for image classification. *Ph. D. thesis section*, 6(2), 2014.

[242] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[243] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[244] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[245] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[246] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[247] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[248] David McNeely-White, J. Ross Beveridge, and Bruce A. Draper. Inception and ResNet features are (almost) equivalent. *Cognitive Systems Research*, 59:312–318, January 2020.

[249] Diane Bouchacourt, Mark Ibrahim, and Ari Morcos. Grounding inductive biases in natural images: Invariance stems from variations in data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 19566–19579. Curran Associates, Inc., 2021.

[250] 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.

[251] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–Decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

[252] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[253] Shudong Yang, Xueying Yu, and Ying Zhou. LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example. In *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pages 98–101, 2020.

[254] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[255] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[256] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6706–6713, 2019.

[257] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[258] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT Pre-Training of Image Transformers. In *International Conference on Learning Representations*, 2022.

[259] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015.

[260] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[261] Cheng Ju, Aurélien Bibaut, and Mark van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *Journal of Applied Statistics*, 45(15):2800–2818, 2018.

[262] Michael Opitz, Horst Possegger, and Horst Bischof. Efficient model averaging for deep neural networks. In *Asian Conference on Computer Vision*, pages 205–220. Springer, 2016.

[263] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[264] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[265] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017.

[266] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

[267] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring Randomly Wired Neural Networks for Image Recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[268] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature Verification using a "Siamese" Time Delay Neural Network. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1994.

[269] Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15745–15753, Nashville, TN, USA, June 2021. IEEE.

[270] Li Jing, Jiachen Zhu, and Yann LeCun. Masked Siamese ConvNets, June 2022.

[271] Nicolas Y. Masse, Gregory D. Grant, and David J. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018.

[272] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional Computation in Neural Networks for faster models. *arXiv:1511.06297 [cs]*, January 2016.

[273] Tianyi Zhou, Shengjie Wang, and Jeff A Bilmes. Diverse ensemble evolution: Curriculum data-model marriage. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[274] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2):181–214, 1994.

[275] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In *ICLR Workshop*, 2014.

[276] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing Networks: Adaptive Selection of Non-Linear Functions for Multi-Task Learning. In *International Conference on Learning Representations*, 2018.

[277] Prajit Ramachandran and Quoc V. Le. Diversity and depth in per-example routing models. In *International Conference on Learning Representations*, 2019.

[278] Bangpeng Yao, Dirk Walther, Diane Beck, and Li Fei-fei. Hierarchical mixture of classification experts uncovers interactions between brain regions. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[279] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to Reason: End-to-End Module Networks for Visual Question Answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[280] Quanshi Zhang, Yu Yang, Qian Yu, and Ying Nian Wu. Network Transplanting. *arXiv:1804.10272 [cs, stat]*, December 2018.

[281] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[282] Ninnart Fuengfusin and Hakaru Tamukoh. Network with Sub-networks: Layer-wise Detachable Neural Network. *Journal of Robotics, Networking and Artificial Life*, 7(4):240–244, 2020.

[283] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable Neural Networks. In *International Conference on Learning Representations*, 2019.

[284] Chao Xiong, Xiaowei Zhao, Danhang Tang, Karlekar Jayashree, Shuicheng Yan, and Tae-Kyun Kim. Conditional convolutional neural

network for modality-aware face recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3667–3675, 2015.

[285] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE transactions on knowledge and data engineering*, 20(2):172–188, 2007.

[286] Guosheng Hu, Yuxin Hu, Kai Yang, Zehao Yu, Flood Sung, Zhihong Zhang, Fei Xie, Jianguo Liu, Neil Robertson, Timothy Hospedales, and Qiangwei Miemie. Deep Stock Representation Learning: From Candlestick Charts to Investment Decisions. *arXiv:1709.03803 [q-fin]*, February 2018.

[287] Guillaume Salha-Galvan, Johannes F. Lutzeyer, George Dasoulas, Romain Hennequin, and Michalis Vazirgiannis. Modularity-Aware Graph Autoencoders for Joint Community Detection and Link Prediction, June 2022.

[288] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast algorithm for modularity-based graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 1170–1176, 2013.

[289] Chihiro Watanabe, Kaoru Hiramatsu, and Kunio Kashino. Modular representation of layered neural networks. *Neural Networks*, 97:62–73, 2018.

[290] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. *arXiv:1609.09106 [cs]*, December 2016.

[291] Karl Ridgeway and Michael C Mozer. Learning Deep Disentangled Embeddings With the F-Statistic Loss. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[292] Cian Eastwood and Christopher K. I. Williams. A Framework for the Quantitative Evaluation of Disentangled Representations. In *Sixth International Conference on Learning Representations (ICLR 2018)*, May 2018.

[293] Pablo Moreno-Muñoz, Antonio Artés, and Mauricio A. Álvarez. Modular Gaussian Processes for Transfer Learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[294] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Networks. *arXiv:1506.02025 [cs]*, February 2016.

[295] Yusuke Iwasawa and Yutaka Matsuo. Test-Time Classifier Adjustment Module for Model-Agnostic Domain Generalization. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[296] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. A review of feature selection methods on synthetic data. *Knowledge and information systems*, 34:483–519, 2013.

[297] David Turner, Robert Wilhelm, Werner Lemberg, Alexei Podtelezhnikov, Toshiya Suzuki, Oran Agra, Graham Asher, David Bevan, Bradley Grainger, Infinality, Tom Kacvinsky, Pavel Kaňkovský, Antoine Leca, Just van Rossum, and Chia-I Wu. The freetype project. https://www.freetype.org/index.html. (Accessed on 11/25/2020).

[298] Alex Clark. Pillow (pil fork) documentation. https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf, 2015. (Accessed on 12/22/2020).

[299] The Unicode Consortium. Unicode – the world standard for text and emoji. https://home.unicode.org/. (Accessed on 12/21/2020).

[300] Ken Whistler. Unicode standard annex #15 unicode normalization forms. https://unicode.org/reports/tr15/. (Accessed on 12/21/2020).

[301] Patrice Y. Simard, Yann A. LeCun, John S. Denker, and Bernard Victorri. Transformation Invariance in Pattern Recognition — Tangent Distance and Tangent Propagation. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, pages 239–274. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[302] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. *arXiv:1606.04080 [cs, stat]*, December 2017.

[303] 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.

[304] Vincent Dumoulin, Neil Houlsby, Utku Evci, Xiaohua Zhai, Ross Goroshin, Sylvain Gelly, and Hugo Larochelle. Comparing Transfer and Meta Learning Approaches on a Unified Few-Shot Classification Benchmark, April 2021.

[305] Gerald Piosenka. BIRDS 450 - SPECIES IMAGE CLASSIFICATION.

[306] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, 2011.

[307] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot Learning—A comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2019.

[308] Heidi M. Sosik, Emily E. Peacock, and Emily F. Brownlee. Annotated plankton images - data set for developing and evaluating classification methods., 2015.

[309] Xiaoping Wu, Chi Zhan, Yu-Kun Lai, Ming-Ming Cheng, and Jufeng Yang. IP102: A large-scale benchmark dataset for insect pest recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[310] Hortense Serret, Nicolas Deguines, Yikweon Jang, Grégoire Lois, and Romain Julliard. Data quality and participant engagement in citizen science: Comparing two approaches for monitoring pollinators in France and South Korea. *Citizen Science: Theory and Practice*, 4(1), 2019.

[311] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE Computer Society, 2008.

[312] Camille Garcin, alexis joly, Pierre Bonnet, Antoine Affouard, Jean-Christophe Lombardo, Mathias Chouet, Maximilien Servajean, Titouan Lorieul, and Joseph Salmon. Pl@ntNet-300K: A plant image dataset with high label ambiguity and a long-tailed distribution. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[313] Lukáš Picek, Milan Šulc, Jiří Matas, Thomas S Jeppesen, Jacob Heilmann-Clausen, Thomas Læssøe, and Tobias Frøslev. Danish fungi 2020-not just another image recognition dataset. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1525–1535, 2022.

[314] David P. Hughes and Marcel Salathé. An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing, 2015.

[315] J. Arun Pandian and G. Geetharamani. Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network. *Mendeley Data, V1*, 2019.

[316] S Roopashree and J. Anitha. Medicinal leaf dataset, October 2020.

[317] Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat, and Nipun Batra. PlantDoc: A dataset for visual plant disease detection. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pages 249–253. Association for Computing Machinery, 2020.

[318] Bartosz Zieliński, Anna Plichta, Krzysztof Misztal, Przemysław Spurek, Monika Brzychczy-Włoch, and Dorota Ochońska. Deep learning approach to bacterial colony classification. *PLOS ONE*, 12(9):1–14, September 2017.

[319] Jevgenij Gamper, Navid Alemi Koohbanani, Ksenija Benet, Ali Khuram, and Nasir Rajpoot. PanNuke: An open pan-cancer histology dataset for

nuclei instance segmentation and classification. In *European Congress on Digital Pathology*, pages 11–19. Springer, 2019.

[320] Jevgenij Gamper, Navid Alemi Koohbanani, Simon Graham, Mostafa Jahanifar, Syed Ali Khurram, Ayesha Azam, Katherine Hewitt, and Nasir M. Rajpoot. PanNuke dataset extension, insights and baselines, 2020.

[321] Peter J Thul, Lovisa Åkesson, Mikaela Wiking, Diana Mahdessian, Aikaterini Geladaki, Hammou Ait Blal, Tove Alm, Anna Asplund, Lars Björk, Lisa M Breckels, et al. A subcellular map of the human proteome. *Science (New York, N.Y.)*, 356(6340), 2017.

[322] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

[323] Haifeng Li, Xin Dou, Chao Tao, Zhixiang Wu, Jie Chen, Jian Peng, Min Deng, and Ling Zhao. RSI-CB: A large-scale remote sensing image classification benchmark using crowdsourced data. *Sensors*, 20(6), 2020.

[324] Zhifeng Xiao, Yang Long, Deren Li, Chunshan Wei, Gefu Tang, and Junyi Liu. High-resolution remote sensing image retrieval based on CNNs from a dimensional perspective. *Remote Sensing*, 9(7), 2017.

[325] Yang Long, Yiping Gong, Zhifeng Xiao, and Qing Liu. Accurate object localization in remote sensing images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(5):2486–2498, 2017.

[326] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

[327] Zhize Wu. Multi-type aircraft of remote sensing images: MTARSI, May 2019.

[328] Erhan Gundogdu, Berkan Solmaz, Veysel Yücesoy, and Aykut Koç. MARVEL: A large-scale image dataset for maritime vessels. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, volume 10115 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2017.

[329] Mario Fritz, E. Hayman, B. Caputo, and J. Eklundh. THE KTH-TIPS database, 2004.

[330] P. Mallikarjuna, Alireza Tavakoli Targhi, Mario Fritz, E. Hayman, B. Caputo, and J. Eklundh. THE KTH-TIPS 2 database, 2006.

[331] Gustaf Kylberg. The kylberg texture dataset v. 1.0. External Report (Blue Series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden, September 2011.

[332] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1265–1278, 2005.

[333] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613. IEEE Computer Society, 2014.

[334] Gertjan J. Burghouts and Jan-Mark Geusebroek. Material-specific adaptation of color invariant features. *Pattern Recognition Letters*, 30(3):306–313, 2009.

[335] Gerald Piosenka. 100 sports image classification.

[336] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1331–1338, 2011.

[337] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2D human pose estimation: New benchmark and state of the art analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3686–3693, 2014.

[338] Christophe Salperwyck and Vincent Lemaire. Learning with few examples: An empirical study on leading classifiers. In *The 2011 International Joint Conference on Neural Networks*, pages 1010–1019. IEEE, 2011.

[339] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: A good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[340] Steinar Laenen and Luca Bertinetto. On episodes, prototypical networks, and few-shot learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24581–24592. Curran Associates, Inc., 2021.

[341] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macià, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, and Evelyne Viegas. Design of the 2015 ChaLearn AutoML challenge. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[342] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, et al. Analysis of the AutoML challenge series. *Automated Machine Learning*, 177, 2019.

[343] Isabelle Guyon, Gavin C Cawley, Gideon Dror, and Vincent Lemaire. Results of the active learning challenge. In *Active Learning and Experimental Design Workshop in Conjunction with AISTATS 2010*, pages 19–45. JMLR Workshop and Conference Proceedings, 2011.

[344] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753, May 2019.

[345] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[346] I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik. What size test set gives good error rate estimates? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):52–64, 1998.

[347] Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep Subdomain Adaptation Network for Image Classification. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1713–1722, April 2021.

[348] Chhavi Yadav and Léon Bottou. Cold Case: The Lost MNIST Digits. *arXiv:1905.10498 [cs, stat]*, November 2019.

[349] S.J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[350] Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and David W Aha. Unsupervised and transfer learning challenge. In *The 2011 International Joint Conference on Neural Networks*, pages 793–800. IEEE, 2011.

[351] Lorenzo Brigato and Luca Iocchi. A close look at deep learning with small data. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2490–2497. IEEE, 2021.

[352] Bernard Koch, Emily Denton, Alex Hanna, and Jacob Gates Foster. Reduced, reused and recycled: The life of a dataset in machine learning research. In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[353] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: Transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

[354] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong,

Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *arXiv:2203.05482 [cs]*, March 2022.

[355] Iou-Jen Liu, Jian Peng, and Alexander Schwing. Knowledge flow: Improve upon your teachers. In *International Conference on Learning Representations*, 2019.

[356] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[357] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[358] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV*, 2017.

[359] Hao Li, Hanan Samet, Asim Kadav, Igor Durdanovic, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[360] Merijn van Erp, Louis Vuurpijl, and Lambert Schomaker. An overview and comparison of voting methods for pattern recognition. In *Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition*, pages 195–200, 2002.

[361] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[362] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In *ICLR*, 2017.

[363] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[364] Emad MalekHosseini, Mohsen Hajabdollahi, Nader Karimi, Shadrokh Samavi, and Shahram Shirani. Splitting Convolutional Neural Network Structures for Efficient Inference, February 2020.

[365] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[366] S.M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. ICDAR 2003 robust reading competitions. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 682–687, 2003.

[367] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

[368] Gongfan Fang. Torch-Pruning, July 2022.

[369] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[370] Yangqing Jia. *Learning Semantic Image Representations at a Large Scale*. PhD thesis, University of California, Berkeley, 2014.

[371] Cristóbal A. Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

[372] David Kirk et al. NVIDIA CUDA software and GPU parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[373] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[374] Sachin Mehta and Mohammad Rastegari. MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022.

[375] Laura Morán-Fernández, Verónica Bólon-Canedo, and Amparo Alonso-Betanzos. How important is data quality? Best classifiers vs best features. *Neurocomputing*, 470:365–375, 2022.

[376] Jacob Andreas. Measuring compositionality in representation learning. In *International Conference on Learning Representations*, 2019.

[377] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528, 2011.

[378] Borja Seijo-Pardo, Amparo Alonso-Betanzos, Kristin P. Bennett, Verónica Bolón-Canedo, Julie Josse, Mehreen Saeed, and Isabelle Guyon. Biases in feature selection with missing data. *Neurocomputing*, 342:97–112, 2019.

[379] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[380] Romain Egele, Romit Maulik, Krishnan Raghavan, Bethany Lusch, Isabelle Guyon, and Prasanna Balaprakash. AutoDEUQ: Automated deep ensemble with uncertainty quantification. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1908–1914, 2022.

[381] Nicolas P. Rougier. Rougier/freetype-py: Python binding for the freetype library.

[382] G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

[383] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. Scikit-image: Image processing in Python. *PeerJ*, 2:e453, June 2014.

[384] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. Imgaug, 2020.

[385] Kevin Wu. Kevinwuhoo/randomcolor-py: A port of David Merfield's randomColor to python.

[386] Gregory Taylor. Gtaylor/python-colormath: A python module that abstracts common color math operations. For example, converting from CIE L*a*b to XYZ, or from RGB to CMYK.

[387] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

[388] Ankush Gupta, Andrea Vedaldi, and Andrew Zisserman. Synthetic Data for Text Localisation in Natural Images. *arXiv:1604.06646 [cs]*, April 2016.

[389] Table of general standard chinese characters.

[390] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105. PMLR, 2015.

[391] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep Domain Confusion: Maximizing for Domain Invariance, December 2014.

[392] Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. *arXiv:1409.7495 [cs, stat]*, February 2015.

[393] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, the Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 443–450. Springer, 2016.

[394] Chaohui Yu, Jindong Wang, Yiqiang Chen, and Meiyu Huang. Transfer learning with dynamic adversarial adaptation network. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 778–786. IEEE, 2019.

[395] Gabriela Csurka. Domain Adaptation for Visual Applications: A Comprehensive Survey. *arXiv:1702.05374 [cs]*, March 2017.

[396] David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting Visual Category Models to New Domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, volume 6314, pages 213–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[397] Jindong Wang and Wenxin Hou. DeepDA: Deep domain adaptation toolkit.

[398] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A. Efros, and Trevor Darrell. CyCADA: Cycle-Consistent Adversarial Domain Adaptation. *arXiv:1711.03213 [cs]*, December 2017.

[399] Geoffrey French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. *arXiv:1706.05208 [cs]*, September 2018.

[400] Jian Liang, Dapeng Hu, and Jiashi Feng. Do We Really Need to Access the Source Data? Source Hypothesis Transfer for Unsupervised Domain Adaptation. *arXiv:2002.08546 [cs]*, October 2020.

A - List of definitions of modularity from the literature

Modularity can be defined as subdivision of a complex object into simpler objects. The subdivision is determined either by the structure or function of the object and its subparts.

(Schmidt *et al.* [70])

Modularity is the property of a system whereby it can be broken down into a number of relatively independent, replicable, and composable subsystems (or modules).

(Amer *et al.* [52])

Modularity as a system design principle is apprehended here as the extent to which processes can be decomposed into modules to be executed in parallel and/or in series.

(Modrak *et al.* [17])

In a wide range of fields, the word “modular” is used to describe complex systems that can be decomposed into smaller systems with limited interactions between them. [...] Roughly speaking, a complex system is said to be modular when it can be decomposed into smaller systems, or components, with limited or controlled interactions between them.

(Avigad [21])

Modularity is an attribute of a system that can be decomposed into a set of cohesive entities that are loosely coupled.

(Pereira-Leal *et al.* [38])

The modularity is a general systems concept: it is a continuum describing the degree to which a system’s components can be separated and recombined.

(Schilling [73])

A module is an entity which owns a function (or service) at a defined performance and can share that function through the employment of an integral interface. A modular architecture is a entity composed of multiple modules, requiring (1) Behavior and boundary conditions to be compatible and consistent for module

types to join; (2) Contains least 2 modular functions and 2 interface; (3) Of any size and complexity; (4) Will merge or join through interoperable interfaces; (5) Can transfer and/or share functional content; (6) The propensity to form modular entities exists prior to the formation of modules.

(Gentile [15])

From a manufacturing perspective, a system is deemed modular if: (1) the equipment units that comprise it form clusters (modules) of dense connectivity (i.e., difficult module assembly tasks are performed off-site), (2) connectivity between modules is sparse (i.e., easy assembly tasks are performed on-site), (3) the number of modules is small, and (4) the module dimensions facilitate transportation.

(Shao *et al.* [16])

The modularity is, up to a multiplicative constant, the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random.

(Newman [39])

A system can be considered 'modular' if its functions are made of multiple dimensions or units to some degree.

(Fodor [26])

Modularity can be intuitively defined as "the integration of functionally related structures and the dissociation of unrelated structures".

(Bongard [59])

(Modularity is) a particular design structure, in which parameters and tasks are dependent within units (modules) and independent across them.

(Baldwin *et al.* [19])

[...] the widespread modularity of biological networks—their organization as functional, sparsely connected subunits.

(Clune *et al.* [25])

Modularity is the localization of function within an encapsulated unit.

(Huizinga *et al.* [74])

Modular— (modules) can be formulated independently of each other.

(Goyal *et al.* [75])

Modularity, or the capacity for the mechanisms in a system to be independently modified, [...].

(Chang *et al.* [55])

In this paper, we define the property of modularity as the ability to effectively learn a different function for each input instance

(Galanti *et al.* [76])

the networks are modular in the sense that we could locally (per node/neuron) ensemble the two networks, [...]

(Wang *et al.* [77])

Factor independence means that variation in one factor does not affect other factors, i.e. there is no causal effect between them. In a disentangled representation factors are also independent in the representation space. In other words, a factor affects only a subset of the representation space, and only this factor affects this subspace. [...] In this paper, we [...] refer to this property as modularity.

(Zaidi *et al.* [78])

[...] factored as a set of specialists (incorporating modularity). [...] there is only a single specialist i.e., without any modularity.

(Goyal *et al.* [79])

B - Pre-rasterization transformations in OmniPrint

The rendering process of modern digital fonts (TrueType/OpenType) is divided into two phases by the rasterization. Digital fonts are originally stored as anchor points expressed in font units within the EM square. Before being able to be rendered into bitmaps, the anchor points are scaled to be aligned with the device pixel grid. The grid-fitting (also called hinting) and rasterization are performed by the FreeType engine (Figure B.1).

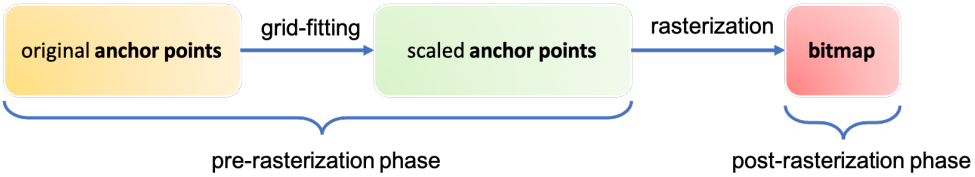


Figure B.1 – **Conversion process from TrueType/OpenType fonts to digital images.** In OmniPrint, pre-rasterization elastic transformation is performed on the original anchor points (yellow), linear transformations of anchor points are performed on the scaled anchor points (green).

Pre-rasterization transformations refer to direct manipulation of the anchor points of the digital font files. Modern fonts (*e.g.,* TrueType or OpenType) are made of straight line segments and quadratic Bézier curves, connecting anchor points. OmniPrint uses the low-level Freetype font rasterization engine [297] (Python binding [381] which is under BSD license), which makes direct manipulation of anchor points possible. With pre-rasterization transformations, one can deform the characters without incurring aberrations due to aliasing and generate some local deformations that would be difficult to achieve with post-rasterization transformations (digital image processing) *i.e.,* natural elastic transformation, variation of character proportion, structured deformation of specific characters, etc.

The implemented pre-rasterization transformations are listed as follows:

- **Elastic transformation (pre-rasterization)** corresponds to random vibration of independent anchor points. The pseudocode is shown in Algorithm 4. Of note is that elastic transformations are implemented in both pre-rasterization phase and post-rasterization phase, which can also be used together. All the elastic transformations mentioned in the main paper refer to pre-rasterization elastic transformation.
- **Stroke width variation** Variation of the stroke width *e.g.,* thinning or thickening of the strokes. Only variable fonts support stroke width vari-

Algorithm 4: Pre-rasterization elastic transformation

Input: A sequence of characters S, a digital font F, a probability distribution D

Output: Rendered text image I

```
// C denotes characters, P denotes anchor points, the function
 load loads the initial anchor points of a digital font for a
 certain character.  The function enumerate returns the index as
 well as the value of an array.
// First pass to compute bounding box of the sequence
```

xmin, xmax, ymin, ymax = 0, 0, 0, 0
Initialize cache `// In order to save random vibration`
for *C in S* **do**
  for *P in load(C, F)* **do**
    xdelta $\sim D$
    ydelta $\sim D$
    P.x $\leftarrow$ P.x + xdelta
    P.y $\leftarrow$ P.y + ydelta
    cache.append((xdelta, ydelta))
    xmin, xmax, ymin, ymax $\leftarrow$ update(xmin, xmax, ymin, ymax, P)
  end
end
$I \leftarrow$ build_image(xmin, xmax, ymin, ymax)
`// Second pass to render text`
for *i, C in enumerate(S)* **do**
  for *j, P in enumerate(load(C, F))* **do**
    P.x $\leftarrow$ P.x + cache[i][j][0]
    P.y $\leftarrow$ P.y + cache[i][j][1]
  end
  $I \leftarrow$ fill_image(I, C)
end

ation, each variable font has its own continuous range of permissible stroke width.

- **Variation of character proportion** *e.g.,* variation of length of ascenders and descenders by some font units.
- **Linear transformations** Rotation, shear, scaling, stretch are assembled into a 2×2 matrix, see Equation B.1. θ denotes the angle (in degree) of counter clockwise rotation, λ_1, λ_2 denote the shear parameters along horizontal axis and vertical axis respectively, s_1, s_2 denote the scaling (stretch) parameters along horizontal axis and vertical axis respectively. If $s_1 = s_2$, this corresponds to a scaling operation, otherwise this corresponds to a stretch operation along horizontal or vertical axes. The stretch along main diagonal axis and anti-diagonal axis by setting $\beta = \gamma \in \mathbb{R}$ or $\lambda_1 = \lambda_2 \in \mathbb{R}$ [301]. The four parameters $\alpha, \beta, \gamma, \delta$ allow inserting an arbitrary linear transform into the default linear transformation pipeline. Users are also allowed to directly set the values of a, b, d, e *i.e.,* the composed linear transformation matrix L.

$$
\begin{aligned}
L &= \begin{pmatrix} a & b \\ d & e \end{pmatrix} \\
&= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 \\ \lambda_2 & 1 \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix} \\
&= \begin{pmatrix} s_1((\alpha+\gamma\lambda_1)\cos\theta - (\alpha\lambda_2+\gamma)\sin\theta) & s_2((\beta+\delta\lambda_1)\cos\theta - (\beta\lambda_2+\delta)\sin\theta) \\ s_1((\alpha+\gamma\lambda_1)\sin\theta + (\alpha\lambda_2+\gamma)\cos\theta) & s_2((\beta+\delta\lambda_1)\sin\theta + (\beta\lambda_2+\delta)\cos\theta) \end{pmatrix}
\end{aligned}
\tag{B.1}
$$

C - Post-rasterization transformations in OmniPrint

- **Translation** is performed, if any, when the foreground text is blended into the background.
- **Perspective transformations** can be used to imitate the effect of different camera viewpoints. A perspective transformation is generally parameterized by a 3×3 matrix in homogeneous coordinates. The homogeneous matrix coefficients are computed from 4 pairs of 2D points in the two projection planes by solving a linear system.
- **Morphological image processing** is a set of operations on the shape of the character and they operate on binary images (foreground vs background). In total, 7 morphological transformations are available via OpenCV [382]: morphological erosion, morphological dilation, morphological opening, morphological closing, morphological gradient, Top Hat, Black Hat.
  - **Morphological erosion** can be used to thin the stroke width in the post-rasterization phase. It erodes away the boundaries of foreground text and it can detach some previously connected strokes. The principle is to apply a 2D convolution, a pixel in the foreground text layer will be kept only if all the neighbor pixels are within the foreground area, otherwise it is eroded. The neighborhood is defined by a convolution kernel whose shape can be selected among rectangle, ellipse or cross-shaped.
  - **Morphological dilation** can be used to thicken the stroke width in the post-rasterization phase and join detached strokes, which is the opposite of morphological erosion. A pixel will be put into the foreground if at least one neighbor pixel is within the foreground area.
  - **Morphological opening** is the morphological erosion followed by morphological dilation. It can remove small pixel noises in the background, if any.
  - **Morphological closing** is the morphological dilation followed by the morphological erosion, which is the opposite of morphological opening. It can close small holes inside the foreground text, if any.
  - **Morphological gradient** is the difference between morphological dilation and morphological erosion of the input image. It can render hollow text in the post-rasterization phase.
  - **Top Hat** is the difference between the input image and the morphological opening of the input image.
  - **Black Hat** is the difference between the morphological closing of the input image and the input image.

- **Gaussian blur** is implemented using scikit-image [383]. In the synthesis pipeline, Gaussian blur is usually applied before downsampling to avoid aliasing.
- **Variation of contrast, brightness, color enhancement, sharpness** is implemented using jungImgaug2020 [384].
- **Elastic transformation (post-rasterization)** [115, 384] moves pixels locally around using displacement field. Depending on parameters, this transform can produce pixelated images or smooth deformation.
- **Foreground filling** Foreground text can be filled either by uniform color or by natural image/texture. The sampling distribution (Figure C.1) of random color is from [385] (MIT License). When using random color for both foreground text and background, OmniPrint automatically ensures that foreground and background colors are visually distinguishable by thresholding the Delta E value (CIE2000). The computation of the Delta E value (CIE2000) is enabled by [386] (BSD-3-Clause License).
- **Text outline** can be generated and filled either by uniform color or by natural image/texture.
- **Background blending** can be done in two ways: (1) naively paste the foreground text onto the background while considering the mask; (2) Poisson Image Editing [387] which ensures seamless blending, this is particularly useful in case of natural background. The implementation is from [388], which is under Apache License 2.0. Background can be filled by uniform color, natural image/texture or uniform color augmented with a random regular polygon.

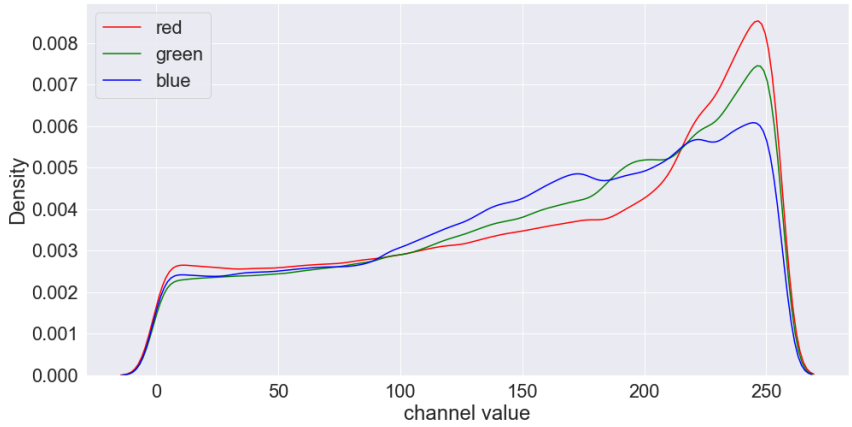


Figure C.1 – **Kernel density estimation of the marginal color distribution.** Each curve is the estimated distribution of one color channel.

D - Alphabets in OmniPrint

Here we present the character selection criteria for OmniPrint:

- For Latin script, we included basic uppercase and lowercase letters, all the variants in different European languages as well as the International Phonetic Alphabet. They are classified into basic Latin uppercase, basic Latin lowercase, Latin-1 Supplement, Latin Extended-A, Latin Extended-B, IPA letters and IPA for disordered speech and sinology, as defined in Unicode standard.
- Chinese characters, also known as CJK Unified Ideographs, are numerous and their usage in real life are extremely imbalanced. In consequence, we only included Chinese characters from Table of General Standard Chinese Characters [389]. These Chinese characters are divided into three levels containing 3500, 3000 and 1605 characters respectively. Characters in group 1 and 2 (the first 6500) are designated as common. Different from other writing systems, the distinction between simplified Chinese characters, traditional Chinese characters, Japanese Kanji and Korean Hanja is only handled by fonts in principle, because many of them share the same code points. The only way to distinguish them is the fonts' rendering. Generally, the fonts that were designed for simplified Chinese characters should never be used when rendering traditional Chinese text or Japanese text, and vice versa. Otherwise, it can be unintelligible or be unacceptable for native speakers. To avoid this overhead, we only aim to render simplified Chinese characters.
- For Japanese, all of Hiragana and Katakana are included. Note that each letter of these two scripts appears twice in the Unicode standard, one corresponds to the normal-sized version, the other is the smaller version. We only included the normal-sized versions.
- For Korean, there are up to 11172 unique syllabic blocks, we only included 2350 syllabic blocks which are assumed to be commonly used.
- All letters of Cyrillic script are not included. Only modern Russian alphabet is included, which consists of 66 upper case and lower case letters.
- Writing systems like Abjad (Arabic, Hebrew, etc.) and Abugida (Thai, Lao, Tibetan, Devanagari, Bengali, etc.) are only partly included. Typically, we only included consonants, independent vowels and digits of these languages. For these scripts (Khmer, Balinese, Bengali, Devanagari, Gujarati, Myanmar, Oriya, Sinhala, Tamil, Telugu, Tibetan, Thai and Lao.), dependent vowel signs were excluded, independent vowels were included if there are any.
- Even though the Mongolian script has been adapted to write languages such as Oirat and Manchu, we only included basic Mongolian letters and

Mongolian digits.

- For the Arabic script, we only included the 29 Arabic letters. For the Hebrew script, we only included the 27 Hebrew letters.
- All of the Ethiopic syllables available in the Unicode standard are included.
- Common punctuations and symbols, ASCII digits, some musical symbols and some mathematical operators are also included. However, neither of the collected fonts fully support these musical symbols.

E - OmniPrint for benchmarking domain adaptation algorithms

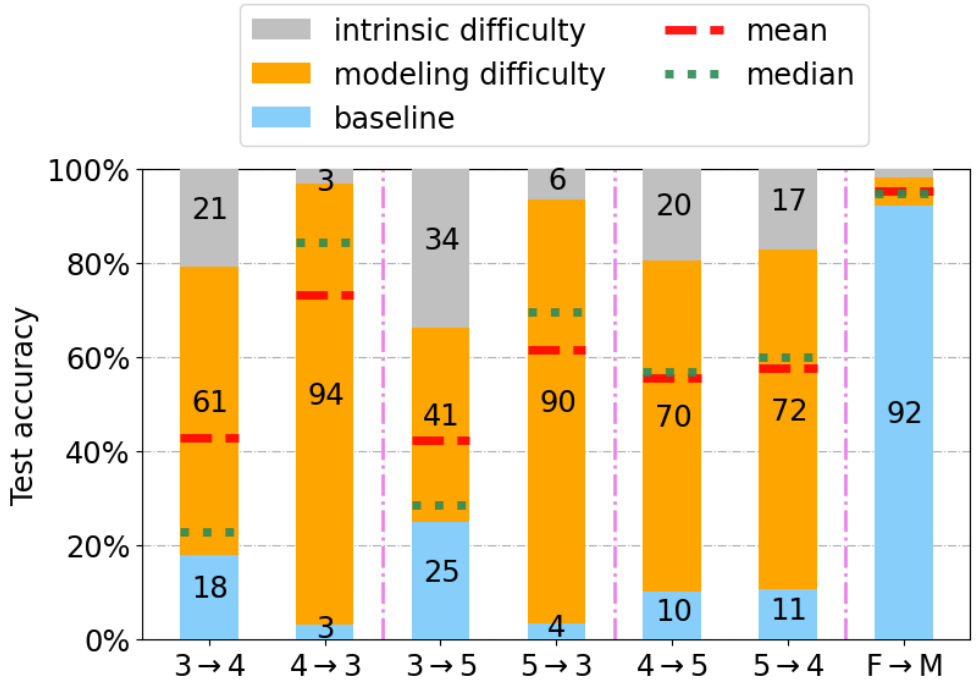


Figure E.1 – **Domain adaptation**. $A \rightarrow B$ means OmniPrint-metaA is source domain and OmniPrint-metaB is target domain, $A, B \in 3, 4, 5$. F $\rightarrow$ M means Fake-MNIST is source domain and MNIST is target domain. Mean and median are computed over 5 methods tried.

This section showcases the utility of OmniPrint in benchmarking domain adaptation [395], one form of transfer learning [349]. Since OmniPrint-meta[1-5] datasets, introduced in Section 3.3.1, share the same label space and only differ in styles and transforms, they lend themselves to benchmarking domain adaptation. We created a domain adaptation benchmark, called OmniPrint-metaX-31 based on OmniPrint-meta[3-5] (last 3 datasets). Inspired by Office-31 [396], a popular domain adaptation benchmark, we only used 31 randomly sampled characters (out of 1409), and limited ourselves to 3 domains, and 20 examples per class. This yields 6 possible domain adaptation tasks, for each combinations of domains.

We tested each one with the 5 DeepDA unsupervised domain adaptation methods [397]: DAN [390, 391], DANN [392], DeepCoral [393], DAAN [394] and DSAN [347]. The experimental results are summarized in Figure E.1 and Ta-

Table E.1 – **Unsupervised domain adaptation results** on OmniPrint-metaX-31. meta$A$ $\rightarrow$ meta$B$ means the source domain is OmniPrint-meta$A$, the target domain is OmniPrint-meta$B$, where $A, B \in 3, 4, 5$. The 95% confidence intervals are computed with 8 random seeds.

	meta3→meta4	meta4→meta3	meta3→meta5	meta5→meta3	meta4→meta5	meta5→meta4
DAN [390, 391]	18.0 ± 2.4	3.2 ± 0.0	25.8 ± 1.7	3.5 ± 0.3	10.1 ± 16.0	10.7 ± 16.5
DANN [392]	72.2 ± 2.8	96.8 ± 0.5	65.6 ± 2.9	82.2 ± 2.7	79.8 ± 1.2	81.5 ± 2.1
DeepCoral [393]	22.9 ± 2.5	84.6 ± 1.5	28.6 ± 1.7	69.6 ± 2.5	57.0 ± 1.3	60.2 ± 1.0
DAAN [394]	22.3 ± 1.8	84.5 ± 2.1	25.1 ± 1.7	59.9 ± 5.9	50.9 ± 1.5	53.3 ± 2.3
DSAN [347]	**79.3 ± 2.3**	**96.9 ± 0.3**	**66.4 ± 2.5**	**93.5 ± 0.8**	**80.5 ± 1.0**	**82.8 ± 1.9**
Average	42.9	73.2	42.3	61.7	55.7	57.7
Median	22.9	84.6	28.6	69.6	57.0	60.2

Figure E.2 – **Example images from Fake-MNIST.** Random pre-rasterization elastic transformation, horizontal shear, rotation and translation were used.

ble E.1. We observe that transfers $A \rightarrow B$ when A is more complex than B works better than the other way around, which is consistent with the domain adaptation literature [398, 399, 400]. The adaptation tasks $4 \rightarrow 5$ and $5 \rightarrow 4$ are similarly difficult, consistent with Section 3.3.1. We also observed that when transferring from the more difficult domain to the easier domain, the weakest baseline method (DAN [390, 391]) performs only at chance level, while other methods thrive.

We also performed unsupervised domain adaptation from a dataset generated with OmniPrint, which we call Fake-MNIST, to MNIST [120], as shown in Figure E.1. Fake-MNIST contains 3000 white-on-black character images for each of the 10 digits. Random pre-rasterization elastic transformation, horizontal shear, rotation and translation were used to synthesize Fake-MNIST. Figure E.2 shows some example images from Fake-MNIST. The performances

Table E.2 – **Unsupervised domain adaptation from Fake-MNIST to MNIST**. 95% confidence intervals are computed with 27 random seeds.

	DAN	DANN	DeepCoral	DAAN	DSAN	Average	Median
Fake-MNIST → MNIST	94.8 ± 0.1	98.0 ± 0.1	92.4 ± 0.2	93.3 ± 0.2	**98.2 ± 0.1**	95.34	94.8

of the 5 DeepDA unsupervised domain adaptation methods (Table E.2) range from 92% to 98% accuracy, which is very honorable (current supervised learning results on MNIST are over 99%).