

# TLIMB - A Transfer Learning Framework for Image Analysis of the Brain

Marc-André Schulz<sup>1,2,†</sup>, Jan Philipp Albrecht<sup>3,4,†</sup>, Alpay Yilmaz<sup>3</sup>, Alexander Koch<sup>1</sup>,  
Dagmar Kainmüller<sup>3,4</sup>, Ulf Leser<sup>3,†</sup> and Kerstin Ritter<sup>1,2,\*,†</sup>

<sup>1</sup>Department of Psychiatry and Neurosciences, Charité – Universitätsmedizin Berlin, Berlin, Germany

<sup>2</sup>Bernstein Center for Computational Neuroscience, Berlin, Germany

<sup>3</sup>Department of Computer Science, Humboldt-Universität zu Berlin, Berlin, Germany

<sup>4</sup>Max-Delbrueck-Center for Molecular Medicine, Berlin, Germany

## Abstract

Biomedical image analysis plays a pivotal role in advancing our understanding of the human body's functioning across different scales, usually based on deep learning-based methods. However, deep learning methods are notoriously data hungry, which poses a problem in fields where data is difficult to obtain such as in neuroscience. Transfer learning (TL) has become a popular and successful approach to cope with this issue, but is difficult to apply in practise due the many parameters it requires to set properly. Here, we present TLIMB, a novel python-based framework for easy development of optimized and scalable TL-based image analysis pipelines in the neurosciences. TLIMB allows for an intuitive configuration of source / target data sets, specific TL-approach and deep learning-architecture, and hyperparameter optimization method for a given data analysis pipeline and compiles these into a nextflow workflow for seamless execution over different infrastructures, ranging from multicore servers to large compute clusters. Our evaluation using a pipeline for analysing 10.000 MRI images of the human brain from the UK Biobank shows that TLIMB is easy to use, incurs negligible overhead and can scale across different cluster sizes.

## Keywords

framework, transfer learning, biomedical image analysis, nextflow

## Introduction

Biomedical imaging, especially in neuroscience, is crucial for understanding the complexities of the central nervous system [15]. It allows for non-invasive examination of brain structure and function, enabling clinical applications like diagnosing and monitoring neurological and psychiatric diseases [2]. Deep learning, with techniques such as convolutional neural networks (CNNs) and transformer-based architectures, show great promise in this domain [8]. Their effectiveness in tasks such as lesion segmentation and disease classification has been demonstrated [18, 20, 8]. However, the success of these advanced architectures often hinges on the availability of large and homogeneous datasets, a challenge in biomedical settings due to their scarcity.

Transfer learning (TL) has recently become popular for overcoming the constraints of small and heterogeneous datasets. In a nutshell, it allows leveraging a model trained on a given source dataset for improving model performance on a different target dataset [21]. However, applying TL in neuroimaging practise has proven difficult, as it requires the careful selection of a multitude of different yet close interacting parameters, including the base image analysis architecture (e.g. ResNet, different flavors of CNNs or transformers), the concrete TL-method (e.g. fine-tuning, multitask-learning), the concrete objective function, and the source dataset to be used. Determining these parameters manually in a framework like PyTorch is time-consuming

and error-prone, as it requires source code manipulation and extensive experimentation to find optimal configurations. These experimentations can be computationally extremely time-consuming unless adequate parallel and/or distributed infrastructures are available, which, however, makes programming the analysis pipeline even more involved.

In this work, we present TLIMB, a Transfer-Learning Framework for Image Analysis of the Brain. TLIMB is programmed in the widely-used general-purpose language Python and based on PyTorch Lightning<sup>1</sup>. With TLIMB, users specify their TL-pipeline in the form of simple and intuitive configuration files, which are then compiled into a concrete image analysis workflow in Nextflow [6], a popular and powerful workflow engine than can execute an analysis over a wide range of infrastructures, ranging from single servers to large compute clusters. With TLIMB, researchers thus are able to easily assess the effectiveness of different TL setups across diverse datasets and environments.

We specifically designed TLIMB as a framework and not as a proper domain specific language (i.e., a programming language tailored to a particular problem; DSL) because of the advantages of this approach in terms of flexibility, ease of creation, extensibility, and seamless integration with existing tools [13, 1]. A Python-based framework, in particular, provides a familiar environment for data scientists, capitalizing on the language's popularity and compatibility with established machine and deep learning frameworks (like PyTorch).

Through a series of experiments following the "brain-age" paradigm [4], a widely-used method for assessing brain health through neuroimaging data, we validated the platform's capability to create a diverse landscape of TL-based pipelines and to execute them seamlessly over any infrastructure supported by Nextflow.

Published in the Proceedings of the Workshops of the EDBT/ICDT 2024 Joint Conference (March 25-28, 2024), Paestum, Italy

\* Corresponding author.

<sup>†</sup> These authors contributed equally.

✉ marc-andre.schulz@charite.de (M. Schulz);

jan-philipp.albrecht@mdc-berlin.de (J. P. Albrecht);


alpay.yilmaz@student.hu-berlin.de (A. Yilmaz);

alexander.koch@charite.de (A. Koch);

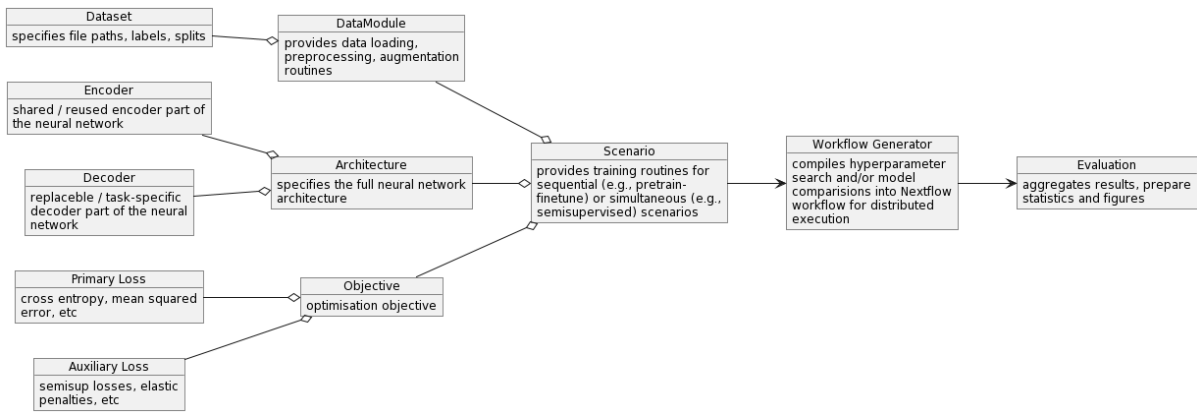
Dagmar.Kainmueller@mdc-berlin.de (D. Kainmüller);

leser@informatik.hu-berlin (U. Leser); kerstin.ritter@charite.de

(K. Ritter)

 © 2024 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<sup>1</sup><https://pypi.org/project/pytorch-lightning/>



**Figure 1:** Simplified diagram of TLIMB’s structure, showcasing all necessary components.

## Related Work

There has been a number of efforts to develop DSLs as well as frameworks for machine learning-based (image) data analysis [9]. OptiML, a DSL tailored for machine learning tasks, seeks to provide an implicitly parallel, expressive, and high-performance alternative to MATLAB and C++ [17]. However, it does not address TL and is agnostic to the data types analysed and thus requires some effort for using it in image analysis. Extending it with TL abilities would be non-trivial due to its design as a DSL. P-Hydra employs Transfer Learning and Multitask learning for image analysis in cancer detection, aiming to validate its algorithmic effectiveness and establish a baseline for other methods [11]. In contrast to TLIMB, the method is implemented in a single pipeline and not designed as configurable framework. Furthermore, our approach supports multiple heads per model, enabling a broader spectrum of TL-methods. Ilastik, designed as an interactive tool for machine-learning-based (bio)image analysis, addresses challenges associated with manual image analysis by providing pre-defined workflows for segmentation, object classification, counting, and tracking, with a user-friendly interface emphasizing accessibility for non-programmers [3]. In contrast to ilastik, our framework concentrates on training neural networks for TL-based analysis. Finally, SimpleITK, is a software package designed for image analysis that provides a simplified interface for flexible and reproducible computational workflows [24], aligning closely with the goals of our framework. While SimpleITK streamlines image analysis through Jupyter Notebooks and introduces various abstractions, our framework adopts a different approach, allowing users to initiate analysis by starting with our framework components and building upon them as needed.

## Methods

### Architecture of TLIMB

The core of the framework is constructed around three primary components: the `DataModule`, `Architecture`, and `ObjectiveFunction`. These are orchestrated within a `Scenario` to create a comprehensive TL pipeline. Users can execute different configurations of these Scenarios, such as for hyperparameter tuning or model comparisons, by automatically generating a Nextflow workflow from their

definitions. An overview of TLIMB’s architecture is shown in Figure 1.

The **Scenario** component is responsible for the training logic: it orchestrates training, validation, and testing by sourcing data from the `DataModule`, processing it through the specified deep learning `Architecture`, calculating losses using the `ObjectiveFunction`, and executing the optimization step. This abstraction level facilitates not only the conventional sequential pretrain-finetune TL workflows, but also enables the implementation of workflows that require simultaneous processing of both pretraining and fine-tuning data, such as semi-supervised learning algorithms [19]. Scenarios are designed to be data-operation agnostic, i.e., independent of the specific deep learning architecture and objective function, thereby enhancing the modularity of the design.

The **Architecture** component relates to the adaptable configuration of network layers and nodes, providing users with the versatility to select from predefined architectures or to incorporate their own custom designs by referencing them in the configuration file. To facilitate efficient TL, architectures are decomposed into two main elements: the *encoder*, which is often repurposed from the source task, and the *head*, which is specific to and replaceable for the target task. This modular structure supports a variety of TL strategies, ensuring adaptability to methodologies such as the core train-fine-tune paradigm and multitask learning.

**Objective Functions** embody the core logic of TL, composed of a primary objective (such as cross-entropy for classification) and an auxiliary objective (such as an elastic penalty on weights during fine-tuning or reconstruction losses in semi-supervised training). During the training phase, this class considers batches and the architecture from the scenario class to compute the loss and performance metrics. In pursuit of greater modularity, Objective Functions have been architected to remain decoupled from other framework components. For instance, employing an Objective Function designed for multitask learning does not require predefined knowledge of the number of heads within the configuration. This design choice facilitates transitions of the objective function, enhancing the user experience and adaptability within the TL workflow.

Our **DataModule** defines the handling of diverse data types, ranging from 3D brain MRI data to 1D fMRI time series. It encompasses data-specific loading, preprocessing, and data augmentation routines. It ensures that batch

preparation conforms to a defined structure and assembles DataLoaders. In contrast to the standard PyTorch Lightning (see below), our method imposes constraints on DataLoader instantiation, mandates a uniform Dataset structure, and centralizes all data-related augmentations and transformations within the DataModule itself. Such a separation-of-concerns supports simple substitutability of Datasets and DataModules. This module also inherits several features from the PyTorch Lightning DataModule, such as the `on_after_batch_transfer` and `on_before_batch_transfer` hooks. These hooks grant users the capability to refine batch post-retrieval but prior to their delivery to the Scenario, enabling, for instance, the offloading of resource-intensive data augmentation strategies to a GPU. This design promotes user-driven adaptability in our framework, ensuring the flexibility to customize components while preserving the integrity of essential operations.

**Datasets**, pivotal elements within the DataModule, are tasked with providing the necessary data and associated labels. The DataModule delineates the procedures for processing a certain category of data, whereas the Dataset is explicit about the specific input files to utilize, their locations within the file system, and the particular labels to retrieve (for instance, selecting the participant's sex for a pretraining task, and later using the same dataset to return the participant's age, thus facilitating straightforward label specification). A DataModule can include multiple Datasets, accommodating various TL strategies that incorporate data from diverse sources. Each Dataset implements a custom `get_item` method to ensure the standardized conveyance of images and labels to the DataModule. This `get_item` method invariably produces a tuple, which includes an image paired with a list of labels, thereby adapting to the diverse labeling demands posed by different Objective Functions. Varied learning paradigms such as Multitask, Pre-train Fine-tune, and Unsupervised Domain Adaptation require unique label arrangements.

The **Configuration** component serves as an important tool for managing configuration within our framework, offering users the ability to customize every aspect of their workflow. Unlike PyTorch Lightning, which primarily focuses on non-structural hyperparameters, our Configuration empowers users to tailor Scenarios, Architectures, Objective-functions, Datasets, DataModules, trainers, and optimizer parameters, ensuring high configurability and modularity. This user-centric approach minimizes coding efforts, allowing users to predominantly interact with the Configuration instead. The framework seamlessly integrates with PyTorch Lightning components, enabling the utilization of features like early stopping and automatic optimizers, effortlessly configurable through the provided configuration file. To ensure reproducibility, each workflow is associated with a defined configuration, facilitating run reproduction. The configuration provides essential details such as splits, which define the distribution of images across training, validation, and testing sets. It also includes adjustable seeds to guarantee consistent runs, except when randomness is introduced by the user. Users are not confined to predefined components; instead, our framework provides interfaces for Objective-functions, Architectures, DataModules, Datasets, and Scenarios, making it easy to implement specialized versions of these components, such as a new Objective-function.

## Integrated models and implementation

Our framework, implemented in Python, provides a seamless integration of PyTorch and incorporates PyTorch Lightning components. This integration offers multiple benefits, such as support for distributed training, compatibility with multi-GPU setups, and optimized performance for various machine learning tasks. The framework's alignment with Python and PyTorch's popularity in the research community simplifies the learning curve, making it a user-friendly and accessible option for TL projects. However, it also offers significant additional functionalities compared to PyTorch Lightning. For instance, our TL Command-Line Interface (CLI) distinguishes itself from the PyTorch Lightning CLI by facilitating the passage of parameters from the DataModule to the Scenario during initialization. This enables users to customize various aspects, such as output size and input size.

The framework is used mainly via configuration files. Users specify key components such as a particular 'DataModule' for input data specifications, a 'Dataset' for data file and label paths, 'Architecture' for neural network structure, 'Objective' for the transfer learning strategy, and 'Scenario' for training details. The framework supports class path parsing, allowing users to define parameters via class references, which can be particularly useful for complex configurations. To facilitate hyperparameter tuning, multiple variants of these parameters can be provided. The framework's workflow generator leverages this information to create Nextflow workflows, which orchestrate the execution of tasks across the computational infrastructure. This streamlined approach enables systematic exploration and efficient optimization of model parameters.

TLIMB comes with a number of readily available models and configurations for its different components. Regarding **architectures**, it currently offers 3d-ResNets of different depths [22], the 3d Simple-Fully-Convolutional-Network [14], as well as vision and swin transformers, three highly popular imaging architectures. ResNet utilizes shortcut connections to enhance training performance, while SFCN is a lightweight 3D convolutional neural network specifically tailored for 3D neuroimaging data. Transformers are fully connected deep encoder-decoder stacks with self-attention. Several customization options, such as filter and kernel sizes and addition of dropout layers are available for each. The framework also integrates pre-processing, neuroimaging domain specific data augmentation, and data transformation techniques.

Regarding **TL algorithms**, our framework encompasses five methods: Pre-train-fine-tune, multitask learning, self-supervised semi-supervised learning, elastic penalty, and unsupervised domain adaptation. Pre-train-fine-tune involves using a pre-trained network for a target task, while elastic penalty introduces an  $L^2$  penalty to preserve learned features during fine-tuning. Multitask learning optimizes models by sharing representations between related tasks. Self-supervised semi-supervised learning leverages both labeled and unlabeled data. Unsupervised domain adaptation allows training deep models using labeled data from a source domain and unlabeled data from a target domain. An overview of these techniques can be found in [10].

TLIMB's **objective functions** mirror PyTorch Lightning training/validation/testing steps. **Hyperparameter optimization** is facilitated through grid search and random search. TLIMB comes with three directly usable **DataMod-**

**Table 1**

Reduction in Lines of Source Code for simple pretrain-finetune scenario when moving from native pytorchlightning to our framework.

Description	Manual Implementation	Framework Implementation
Total Source Lines	286	34
Data Module Definition	81	-
Dataset Definition	34	34
Lightning Module	51	-
Rest (Losses, Architecture, Import)	125	-

**Table 2**

Comparison of Execution Times (on AMD 3970X 32-Core / Nvidia GeForce 3090). We report average values of three runs, together with their standard deviation (in brackets). Reduction in execution time in our framework is mostly due to parallelisation of training and testing steps.

Execution Workflow	Manual Implementation (s)	Framework Implementation (s)
CPU only	109.66 (+- 0.47)	79.33 (+- 1.88)
Single GPU	72.66 (+- 0.47)	46.00 (+- 0.00)

ules, namely BaseDataModule, CropCenterDataModule, and BioImageDataModule. Additionally, several pre-defined **Datasets** from the Human Connectome Project are readily available, but researchers can effortlessly incorporate any image analysis dataset of their choice by utilizing the provided interface.

### Nextflow as workflow manager

Nextflow is a mature and popular scientific workflow engine [7]. Workflows in Nextflow are written in a proper workflow language based on Groovy and are executed by a workflow engine which controls data dependencies, maximises parallelism in task executions, and supports reproducibility by a sophisticated logging mechanism. Workflows can either be executed locally (non distributed) by the system itself, or passed on to popular resource managers, such as Slurm or Kubernetes [25], for scheduling on arbitrarily large clusters. In TLIMB, we utilize Nextflow to assemble TL workflows from user-provided configurations into a workflow script. This script can then be executed in parallel and distributed across all supported infrastructures, significantly accelerating the processing speed.

## Experiments

For the evaluation of the TLIMB framework, we used T1-weighted brain images from the UK Biobank [12]. To streamline the evaluation process, we processed images by applying linear registration and extracting the central axial slices. This reduced the dimensionality of the data, allowing us to expedite the training process. We created three subsets of randomly sampled images: 10,000 for pre-training, 500 for fine-tuning, and 1,000 for the test set. Models were pre-trained on age regression, and fine-tuned on sex classification.

To assess usability improvements, we specified a simplified search space comprising two different neural network architectures (ResNet-18 and a Vision Transformer), three different learning rates ( $10^{-4}$ ,  $10^{-3}$ ,  $10^{-2}$ ), and an optional elastic penalty loss [23] as an advanced fine-tuning technique. Both models were pre-trained for 10 epochs and fine-tuned for one epoch. Such limited training time would be insufficient for real world applications, but our aim here is

to investigate the framework’s usability and computational overhead rather than achieving state-of-the-art accuracy. Each variant was implemented in two ways: manually using PyTorch with PyTorch Lightning, and through our TLIMB framework compiled into a Nextflow workflow. These implementations were then run in three different scenarios: manually without the framework, with the framework sequentially, and with the framework in parallel. The primary metrics for evaluation were the execution times and the lines of code required for each scenario. Execution times are documented in Table 1, illustrating the comparison between running the processes with and without the framework, both sequentially and in parallel.

Additionally, we conducted a minimal set of experiments illustrating how TLIMB may be used in practice. On the same data set and using the ResNet-18 architecture, we compared fine-tuning effectiveness for different numbers of frozen layers in the pre-trained model. Freezing lower layers of a pre-trained model reduces the number of trainable parameters and thus reduces the risk of overfitting during the fine-tuning process. Metrics for pre-training and fine-tuning performance are shown in Table 3 and 4 respectively. TLIMB achieved expected levels of accuracy, in line with other studies [5, 16].

**Execution Time:** The execution times indicated minimal to no computational overhead when using the TLIMB framework. The parallel execution with nextflow significantly reduced the time compared to the sequential runs, showcasing the framework’s scalability (see Table 1).

**Lines of Code:** A notable reduction in lines of code was observed when using TLIMB, emphasizing the ease of use and time savings in coding. The framework abstracted many of the repetitive tasks, such as setting up data loaders, model configurations, and hyperparameter tuning, which contributed to a more streamlined development process.

**Prediction Performance:** Although no exact replication of literature results was attempted at the time of writing, our preliminary results are compatible with literature expectations.

**Table 3**

Accuracy of a ResNet18 predicting brain age from 2D images of human brain from the UK Biobank. Models were trained from scratch. Reported results are the average and standard deviation over three training runs.

Learn Rate	MSE	MAE
0.01	31.3 (+- 2.63)	4.45 (+- 0.2)
0.001	<b>26.4</b> (+- 0.17)	<b>4.1</b> (+- 0.03)
0.0001	32.99 (+- 1.14)	4.6 (+- 0.08)
0.00001	59.99 (+- 1.63)	6.32 (+- 0.09)

**Table 4**

Accuracy of a ResNet18 predicting sex from 2D images of the human brain from the UK Biobank. Models were pre-trained on age prediction (see Table 3). Learning rate was fixed at 0.001. We show 4 variations in which increasing numbers of layers are kept frozen during fine-tuning. "None" refers to no frozen parameters.

Freeze up to Layer	CE	Accuracy	Trainable Parameters
None	1.29 (+- 0.3)	0.76 (+- 0.04)	11.2 M
layer3.1.conv1	0.71 (+- 0.1)	0.78 (+- 0.01)	9.6 M
layer4.0.conv1	<b>0.67</b> (+- 0.3)	<b>0.79</b> (+- 0.01)	<b>8.4 M</b>
layer4.1.conv1	0.56 (+- 0.0)	0.72 (+- 0.0)	4.7 M

## Conclusion and outlook

In this study, we introduce our innovative solution – a tailored implementation and evaluation platform for TL techniques in biomedical imaging applications. Guided by specific requirements, we opted for a comprehensive framework over a DSL. Our framework comprises two key components: firstly, a Python framework built upon PyTorch Lightning, facilitating diverse user-defined TL tasks. Secondly, a workflow generator and executor ensuring scalability. We provide in-depth descriptions of both components, highlighting their functionalities and capabilities. To ascertain the effectiveness and utility of our framework, we applied it to the "brain-age" paradigm. In this context, the assessment of brain-age deviations from chronological age serves as a metric for evaluating brain health. Our framework demonstrates minimal or no computational overhead, while significantly reducing the number of lines of code required. In the pursuit of refining our framework, we propose several avenues for future development. Firstly, we recommend the establishment of a standardized template to streamline the evaluation of TL methods. This template would simplify result and methodology comparisons among researchers, fostering a more cohesive and efficient research environment. Moreover, to enhance the efficiency of model tuning, we advocate for the implementation of additional hyperparameter optimization methods within our framework. Specifically, techniques like Bayesian Optimization can be incorporated to further optimize model performance. Furthermore, to minimize manual intervention and improve user experience, we suggest enhancing the workflow manager. This enhancement includes the addition of automatic ranking capabilities, which will facilitate a more efficient comparison and selection of the best-performing models, guided by predefined evaluation metrics.

## References

- [1] Alexander Alexandrov et al. "Implicit parallelism through deep language embedding". In: *SIGMOD*. 2015, pp. 47–61.
- [2] Rohit Bakshi et al. "MRI in multiple sclerosis: current status and future prospects". In: *The Lancet Neurology* 7.7 (2008), pp. 615–625.
- [3] Stuart Berg et al. "Ilastik: interactive machine learning for (bio) image analysis". In: *Nature methods* 16.12 (2019), pp. 1226–1232.
- [4] James H Cole et al. "Predicting brain age with deep learning from raw imaging data results in a reliable and heritable biomarker". In: *NeuroImage* 163 (2017), pp. 115–124.
- [5] James H. Cole. "Multimodality neuroimaging brain-age in UK biobank: relationship to biomedical, lifestyle, and cognitive factors". In: *Neurobiology of Aging* 92 (Aug. 2020), pp. 34–42. ISSN: 0197-4580. DOI: 10.1016/j.neurobiolaging.2020.03.014. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7280786/> (visited on 03/04/2024).
- [6] P. Di Tommaso et al. "Nextflow enables reproducible computational workflows". In: *Nat Biotechnol* 35.4 (2017), pp. 316–319.
- [7] Paolo Di Tommaso et al. "Nextflow enables reproducible computational workflows". In: *Nature biotechnology* 35.4 (2017), pp. 316–319.
- [8] Fabian Eitel et al. "Promises and pitfalls of deep neural networks in neuroimaging-based psychiatric research". In: *Experimental Neurology* 339 (2021), p. 113608.
- [9] Joan Giner-Miguel, Abel Gómez, and Jordi Cabot. "A domain-specific language for describing machine learning datasets". In: *Journal of Computer Languages* 76 (2023), p. 101209.
- [10] Padmavathi Kora et al. "Transfer learning techniques for medical image analysis: A review". In: *Biocybernetics and Biomedical Engineering* 42.1 (2022), pp. 79–107.
- [11] Jiyoung Lee. "P-Hydra: Bridging Transfer Learning And Multitask Learning". In: *Master Thesis, University of Friburg* (2020).

- [12] Thomas J Littlejohns et al. “The UK Biobank imaging enhancement of 100,000 participants: rationale, data collection, management and future directions”. In: *Nature communications* 11.1 (2020), p. 2624.
- [13] Marjan Mernik, Jan Heering, and Anthony M. Sloane. “When and How to Develop Domain-Specific Languages”. In: *ACM Comput. Surv.* 37.4 (Dec. 2005), pp. 316–344.
- [14] Han Peng et al. “Accurate brain age prediction with lightweight deep neural networks”. In: *Medical image analysis* 68 (2021), p. 101871.
- [15] Rangaraj M Rangayyan. *Biomedical image analysis*. CRC press, 2004.
- [16] Marc-Andre Schulz et al. “Performance reserves in brain-imaging-based phenotype prediction”. In: *Cell Reports* 43.1 (2024).
- [17] Arvind Sujeeth et al. “OptiML: an implicitly parallel domain-specific language for machine learning”. In: *ICML*. 2011, pp. 609–616.
- [18] Sergi Valverde et al. “Improving automated multiple sclerosis lesion segmentation with a cascaded 3D convolutional neural network approach”. In: *NeuroImage* 155 (2017).
- [19] Jesper E Van Engelen and Holger H Hoos. “A survey on semi-supervised learning”. In: *Machine learning* 109.2 (2020), pp. 373–440.
- [20] Sandra Vieira, Walter HL Pinaya, and Andrea Mechelli. “Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications”. In: *Neuroscience & Biobehavioral Reviews* 74 (2017), pp. 58–75.
- [21] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big data* 3.1 (2016), pp. 1–40.
- [22] Wannu Xu, You-Lei Fu, and Dongmei Zhu. “ResNet and Its Application to Medical Image Processing: Research Progress and Challenges”. In: *Computer Methods and Programs in Biomedicine* (2023), p. 107660.
- [23] LI Xuhong, Yves Grandvalet, and Franck Davoine. “Explicit inductive bias for transfer learning with convolutional networks”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2825–2834.
- [24] Ziv Yaniv et al. “SimpleITK image-analysis notebooks: a collaborative environment for education and reproducible research”. In: *Journal of digital imaging* 31.3 (2018), pp. 290–303.
- [25] Naweiluo Zhou, Huan Zhou, and Dennis Hoppe. “Containerization for High Performance Computing Systems: Survey and Prospects”. In: *IEEE Transactions on Software Engineering* 49.4 (2022), pp. 2722–2740.

## Acknowledgements and Funding

This work was funded by FONDA (DFG; SFB 1404; Project ID: 414984028).