ClinicaDL: an open-source deep learning software for reproducible neuroimaging processing

Elina Thibeau-Sutre, Mauricio Diaz, Ravi Hassanaly, Alexandre M Routier, Didier Dormont, Olivier Colliot, Ninon Burgos

ClinicaDL: an open-source deep learning software for reproducible neuroimaging processing

Elina Thibeau-Sutre[*a], Mauricio Díaz[*a], Ravi Hassanaly[a], Alexandre Routier[a], Didier Dormont[b], Olivier Colliot[a], Ninon Burgos[a]

[a]*Sorbonne Université, Institut du Cerveau - Paris Brain Institute - ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, F-75013, Paris, France*
[b]*Sorbonne Université, Institut du Cerveau - Paris Brain Institute - ICM, CNRS, Inria, Inserm, AP-HP, Hôpital de la Pitié Salpêtrière, DMU DIAMENT, F-75013, Paris, France*

Abstract

Background and Objective As deep learning faces a reproducibility crisis and studies on deep learning applied to neuroimaging are contaminated by methodological flaws, there is an urgent need to provide a safe environment for deep learning users to help them avoid common pitfalls that will bias and discredit their results. Several tools have been proposed to help deep learning users design their framework for neuroimaging data sets.

Software overview We present here ClinicaDL, one of these software tools. ClinicaDL interacts with BIDS, a standard format in the neuroimaging field, and its derivatives, so it can be used with a large variety of data sets. Moreover, it checks the absence of data leakage when inferring the results of new data with trained networks, and saves all necessary information to guarantee the reproducibility of results. The combination of ClinicaDL and its companion project Clinica allows performing an end-to-end neuroimaging analysis, from the download of raw data sets to the interpretation of trained networks, including neuroimaging preprocessing, quality check, label definition, architecture search, and network training and evaluation.

Conclusions We implemented ClinicaDL to bring answers to three common issues encountered by deep learning users who are not always familiar with neuroimaging data: (1) the format and preprocessing of neuroimaging data sets, (2) the contamination of the evaluation procedure by data leakage and (3) a lack of reproducibility. We hope that its use by researchers will allow producing more reliable and thus valuable scientific studies in our field.

Keywords: Deep learning, Reproducibility, Neuroimaging, Data leakage, Open-source

[*]The two authors contributed equally to this paper.

Article accepted for publication at Computer Methods and Programs in Biomedicine

1. Introduction

In recent years, deep learning has become one of the most used data analysis technique. This statement also applies to computer-aided diagnosis systems in which convolutional neural networks (CNNs) are widely used to provide a diagnosis or predict the future state of patients from medical imaging data. Unfortunately, this recent massive use of deep learning has also been associated with methodological flaws in many studies [1, 2, 3, 4, 5] as new deep learning users may not be familiar with the validation methods that were adopted by the machine learning community. Such studies overestimate the performance of their network in performing classification because their test set (when it exists) is contaminated by data leakage. This is a major issue in the field that may lead to troublesome consequences:

- If they are biased, these frameworks will not be translated to real-life applications, leading to a loss of resources and time by the scientific community.
- Authors producing honest results with a sound method may not succeed in publishing their results because the biased state-of-the-art performance is too high.
- Deep learning methods are inappropriately used where it is not needed.

Moreover, [6] points out that the whole deep learning community faces a reproducibility crisis that discredits the results obtained with this method. Hence there is an urgent need in publishing open-source software, data sets and scripts that allow reproducing the methodologies described in deep learning studies. Finally, another main difficulty encountered by deep learning users who are not neuroimaging specialists is the access to properly formatted and preprocessed data sets. In our field, a standard was developed to better organize and share neuroimaging data sets: the Brain Imaging Data Structure (BIDS) [7]. Then projects such as Clinica [8] or BIDS Apps [9] help preprocess these BIDS data sets. But unfortunately, many data sets are still not distributed in this format, and deep learning users are not always familiar with image preprocessing tools.

Several open-source tools have been made available in the last years to ease deep learning application to medical images. Lakhani et al. [10] proposed a tutorial-like paper explaining how to use Keras to train a classifier on two specific data sets hosted on a dedicated repository[1]. Others preferred to implement dedicated open-source libraries to propose an easier use of deep learning for medical image analysis. The most important one is Monai[2], a large library merging three other libraries that are not maintained anymore: NiftyNet[3] [11], DeepNeuro[4] [12] and DLTK[5] [13]. This library goes beyond the context of neuroimaging and allows processing

[1] `https://github.com/ImagingInformatics/machine-learning`
[2] `https://monai.io`
[3] `https://github.com/NifTK/NiftyNet`
[4] `https://github.com/QTIM-Lab/DeepNeuro`
[5] `https://github.com/DLTK/DLTK`

medical images from different body parts. It is meant to work on MedMNIST[6], a series of ten data sets of preprocessed medical images of different modalities (cancer histology, chest X-ray, dermatoscopy, optical coherence tomography, fundus photography, breast ultrasound and abdominal computed tomography) and the ten data sets of the medical segmentation decathlon challenge [14, 15][7] that aims at segmenting organs or tumours. It is also possible to use Monai on other data sets but this requires additional work. Monai provides low-level functions and classes that must be combined in a Python script to learn a classification or a segmentation task, or to train a generative adversarial network (GAN). Attribution methods are also available: class activation mapping (CAM), gradient-weighted CAM and occlusion sensitivity, which allow interpreting the trained network results. A large diversity of transforms, loss functions, metrics and optimizers are provided. Finally, with the support of Nvidia, Monai provides multi-GPU parallel processing. The other main Python library for deep learning in medical research is TorchIO [16]. This library does not implement deep neural network training, but proposes a large variety of transforms for 3D image preprocessing and/or augmentation to prepare data for deep learning use. As with Monai, two public data sets are integrated with the library for ease of use: IXI[8] and EPISURG [17]. An interface to manage custom data sets is also provided. Finally, Nobrainer[9], focuses on learning brain segmentations and is documented by Jupyter notebook tutorials. Other initiatives were launched but seemingly abandoned, such as for example NiftyTorch[10], MildInt[11] [18] or pymia[12] [19].

Though there was an effort from previous works to integrate several data sets that can be easily downloaded through their API, other cohorts might be quite difficult to process. Moreover, they do not easily handle longitudinal data sets, in which several images correspond to the same participants and then should not be distributed between the training and test sets. Finally, the reproducibility of the experiments conducted with these frameworks still heavily relies on the user. Indeed, the hyperparameter values are defined in the scripts, hence their previous values may be lost as new experiments are launched.

To help deep learning users to (1) format and preprocess neuroimaging data sets, (2) prevent data leakage from biasing their results and (3) reproduce their experiments, we implemented ClinicaDL: a command line software written in Python meant to train deep learning networks to reconstruct input images, or to predict a categorical (classification) or a continuous (regression) variable based on neuroimaging data. As the name suggests, it is meant to work on the outputs of Clinica [8], an open-source software platform for reproducible clinical neuroimaging studies. The core of Clinica is a set of automatic pipelines for multimodal

[6] `https://medmnist.github.io`
[7] `http://medicaldecathlon.com`
[8] `https://brain-development.org/ixi-dataset`
[9] `https://github.com/neuronets/nobrainer`
[10] `https://github.com/NiftyTorch`
[11] `https://github.com/goeastagent/MildInt`
[12] `https://github.com/rundherum/pymia/tree/master`

neuroimaging data preprocessing with standard tools of the community (such as SPM[13], FreeSurfer[14] or ANTS[15]). In the same way as ClinicaDL, Clinica is a community-driven library which is meant to be extensible: even though not all possible modalities and data sets are handled, external users can add new pipelines, or ask the team to do it for them. ClinicaDL takes as inputs the images preprocessed by Clinica and convert them into tensors to train deep neural networks. Thus, the combination of these two tools allows performing an end-to-end neuroimaging analysis, from the download of raw data sets to the interpretation of trained networks, including neuroimaging preprocessing, quality check, label definition, architecture search, and network training and evaluation.

In this paper, we first present the common pitfalls encountered by deep learning users in the neuroimaging community and the solutions brought by ClinicaDL to avoid them. We think it is necessary to clearly state these issues as we witnessed in our previous work [1] that half of the studies of our domain (CNN for the classification of Alzheimer's disease using T1-weighted magnetic resonance imaging [MRI]) may be contaminated by methodological flaws. Then, we give an overview of the functionalities already implemented, or that can be easily added by external users in ClinicaDL. Real-world examples illustrating the functioning of the software can be found in our tutorial suite: `https://aramislab.paris.inria.fr/clinicadl/tuto/intro.html`.

2. Avoiding common pitfalls in deep learning studies with ClinicaDL

2.1. Formatting and preprocessing of neuroimaging data

One difficulty faced by data scientists is the manipulation of raw neuroimaging data sets as their organization can be quite difficult to understand. Moreover, raw images coming from different scanners may need some preprocessing to be handled by deep neural networks. These preprocessing steps are easier to perform and manage when data are organized in a standard manner. To allow any researcher to completely or partly reproduce the steps performed in a study with ClinicaDL, we decided to work with data set structures, described in the following sections, whose specifications are exhaustive. Moreover, our framework is adapted to the use of 3D images, as it allows the user to choose their own way to cut the image in smaller pieces (patches or slices) to ease network training (see ClinicaDL modes in section 2.1.3).

2.1.1. BIDS format

Clinica and ClinicaDL follow the Brain Imaging Data Structure, described in [7], to organize their data sets. The BIDS standard provides a list of specifications[16], which specify how files in a BIDS data set should be organized, named and formatted. It is widely adopted by the neuroimaging community, and more and more databases try to distribute

[13] `https://www.fil.ion.ucl.ac.uk/spm`
[14] `https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki`
[15] `http://stnava.github.io/ANTs`
[16] `https://bids-specification.readthedocs.io`

their data in BIDS format or approaching (see the OpenNeuro[17] platform for a list of BIDS formatted data sets). However, some databases still use a custom format that can be quite difficult to exploit. This is the case for example of the Alzheimer's Disease Neuroimaging Initiative (ADNI)[18], the Australian Imaging, Biomarker & Lifestyle Flagship Study of Ageing (AIBL)[19], or the Open Access Series of Imaging Studies (OASIS)[20], three databases used for the characterization of Alzheimer's disease dementia and its prodromal stage, or of the frontotemporal lobar degeneration neuroimaging initiative (NIFD), a database including patients with frontotemporal dementia (available from the same platform as ADNI and AIBL[21]). Clinica includes BIDS converters to format these four raw data sets to BIDS format to ease their use.

2.1.2. CAPS format

When a database is BIDS-formatted, it is possible to preprocess its images with Clinica pipelines. These pipelines can for example perform intensity and spatial normalization of brain images to allow the extraction and analysis of comparable features from images of different participants. Two pipelines have been developed specifically for deep learning use, though the outputs of the other pipelines can also be used with ClinicaDL. As deep learning is meant to work on data as raw as possible, these two pipelines were designed to perform a minimal preprocessing. Indeed, they mainly consist of a linear registration to a standard space, for two different modalities: T1-weighted MRI and positron emission tomography (PET) images. All pipelines output another folder whose structure is derived from BIDS: the ClinicA Processed Structure (CAPS)[22].

2.1.3. ClinicaDL modes

The preprocessing pipelines of Clinica operate at the image level, but many deep learning systems work with one or several parts of the original 3D image. Four possible uses of the image (modes) are currently implemented in ClinicaDL:

1. `image` uses the whole 3D image,
2. `patch` extracts 3D cubic patches with predefined size and stride to cover the whole image,
3. `roi` extracts specific 3D regions defined by binary masks generated by the user,
4. `slice` extracts 2D slices according to a neuroanatomical plane (sagittal, coronal or axial).

[17] `https://openneuro.org/`
[18] `http://adni.loni.usc.edu`
[19] `https://aibl.csiro.au`
[20] `https://www.oasis-brains.org`
[21] `https://ida.loni.usc.edu`
[22] `https://aramislab.paris.inria.fr/clinica/docs/public/latest/CAPS/Introduction/`

ClinicaDL computes the image-level performance by assembling the mode-level performance when it is different from `image`. Advanced users may want to implement their own modes. The documentation describes the steps to follow to implement and use custom modes.

2.1.4. MAPS format

Model Analysis and Processing Structure (MAPS) is the name of the output structure of the ClinicaDL train function. All the functions of ClinicaDL are meant to work on this structure to easily retrieve the parameters of the command line, the weights of the best models, the checkpoints, or the predictions made on the training and validation sets to compute the results at the image level on independent test sets. At the root of the hierarchy, the file `environment.txt` summarizes the environment used for training, and `maps.json` gathers the arguments provided to the command line.

This structure includes a hierarchy of three levels:

1. **Splits** The first level contains one folder per train / validation split. The training procedure of each split can be launched independently.
2. **Selection metrics** During the training procedure of a particular split, one network is selected per selection metric given in input. These networks correspond to the network having the best validation performance according to their metric during the training procedure.
3. **Data groups** Finally, the best networks selected are evaluated on data groups. The characteristics of these data groups (TSV file of participant and session IDs with label values, and path to the CAPS directory) are stored at the first level of the hierarchy in the `groups` folder. This specification ensures the consistency between the evaluations of different networks trained on different splits and selected on different metrics.

An example of the MAPS obtained when training a classification CNN trained on images is displayed in Table 1. The MAPS also stores training logs. Two different formats are available: they can be opened with Tensorboard[23] and are also available as TSV files.

2.1.5. Conclusion

Relying on BIDS allows easing the processing of neuroimaging data as it is a standard format. As already mentioned, many BIDS data sets are hosted on OpenNeuro[24]. For the others, tools have been developed to ease their conversion (a list of these tools is available on the BIDS website[25]).

The other formats we introduced (CAPS and MAPS) are useful as their elements can be easily processed and retrieved with tools of Clinica or ClinicaDL. For example, ClinicaDL includes two quality check procedures taking as input the CAPS generated by pipelines of

[23] `https://www.tensorflow.org/tensorboard`
[24] `https://openneuro.org/public/datasets`
[25] `https://bids.neuroimaging.io/benefits.html#converters`

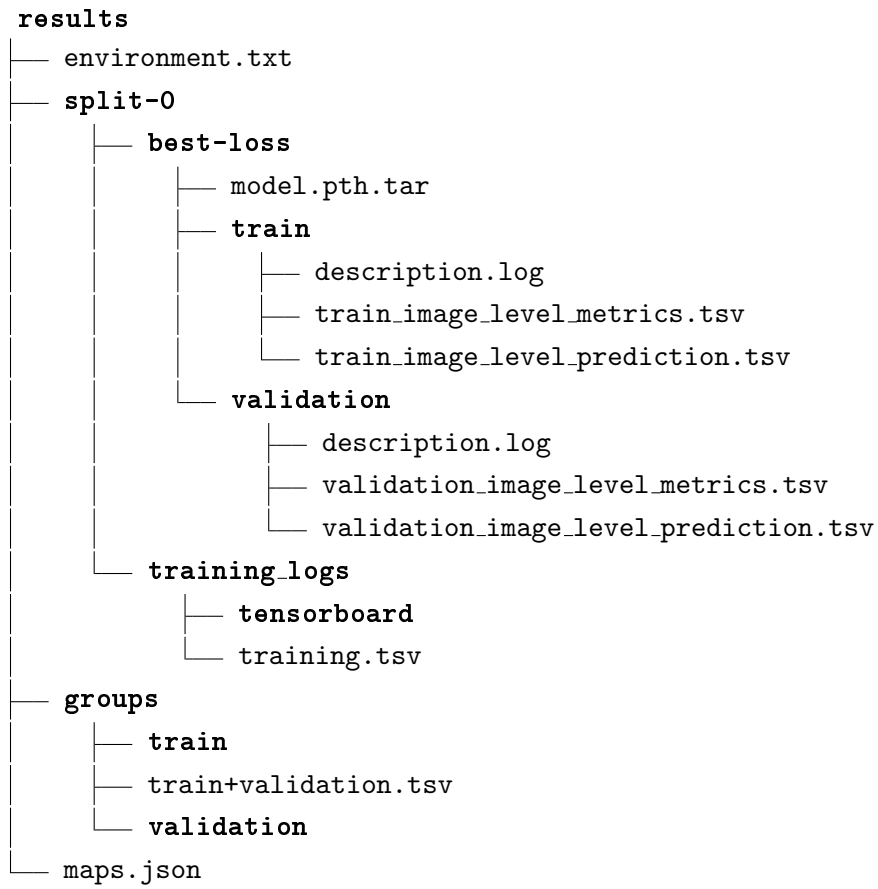

Table 1: Example of the Model Analysis and Processing Structure (MAPS) obtained when training a classification network on whole images. Folders are in bold.
Only the first split was trained (folder **split-0**) and one model was selected based on its validation loss (folder **best-loss**).
The only data groups are **train** and **validation**, which are automatically created during training. The characteristics of these groups are defined in **groups**, whereas the folder in **split-0/best-loss** contains the results for each input image (file `*_prediction.tsv`) and a set of metrics (file `*_metrics.tsv`) for each data group.
Finally, training logs are available for each split training in the folder **training_logs**. These logs are available in two different formats, Tensorboard compatible and TSV.
As the training procedure ended without raising an error, the checkpoints were erased (this allows saving storage space).

Clinica (`t1-linear`[26] and `t1-volume`[27]). Of note, the combination of Clinica and ClinicaDL functionalities leads to the generation and use of data sets in both formats (BIDS and CAPS). This is a limitation as it is storage costly, however it is necessary as the CAPS is only meant to include preprocessed images and does not include participants' meta-data. Raw images may be deleted by the users after preprocessing and merging meta-data if they want to save space, though it is generally recommended to keep raw data as it allows reproducibility.

2.2. Data leakage handling

As explained by Kaufman et al. [20], data leakage is "the introduction of information about the target of a data mining [a.k.a. machine learning] problem that should not be legitimately available to mine from". They give two main reasons for data leakage:

- leaking features, occurring for example when input data include features that are highly correlated to the target label due to a selection bias or if the target is a cause of the feature,
- leakage in training examples, occurring when data used for training is not legitimate towards data used for performance evaluation (for example, if there is an intersection between training and test data).

Let's take the example of the inference of the diagnosis from neuroimaging data. In this case, the leaking feature scenario may happen as a selection bias. For example, consider a data set that includes several sites. If each site has a different diagnosis distribution (in the worst case, one site only recruits patients, whereas another one only recruits control subjects), the site is a leaking feature for the diagnosis. Unfortunately, as these sites use different scanners, the site information may be retrieved from the neuroimaging data. This selection bias requires expert knowledge of the data set used to be avoided.

We mainly focus in this article on leakage in training examples, which is independent from the data sets used. In a previous study [1], we reported that data leakage contaminated nearly half of the studies using a CNN on T1-weighted MRI for the diagnosis of Alzheimer's disease. Other studies using deep learning in the health domain also mention that data leakage pollutes their field of application: [2] in breast cancer detection from mammograms, [3] for Covid-19 diagnosis from chest radiography and [4] for image classification in digital pathology. Finally, [5] quantified the difference between a biased and a right split between train and test sets on the test accuracy for several tasks using neuroimaging data. The differences they measured ranged from 25% on a large data set to 55% on a small data set.

We identified four scenarios of data leakage in our previous paper [1] and we add here a last one (biased ensemble learning) that has been identified afterwards.

[26] `https://aramislab.paris.inria.fr/clinica/docs/public/latest/Pipelines/T1_Linear/`
[27] `https://aramislab.paris.inria.fr/clinica/docs/public/latest/Pipelines/T1_Volume/`

1. **Absence of an independent test set** occurs when the classifier performance is evaluated on the training or the validation set.
2. **Biased split** occurs when highly correlated data (slices or patches extracted from the same volume, visits from the same patient, etc.) are both in the train and the test sets.
3. **Late split** occurs when another procedure (for example learning the size and stride of patches, the best location for regions of interest, or selecting deep learning hyperparameters) is performed prior to the data split on all images.
4. **Biased transfer learning** occurs when data is shared between the source and the target task and that the train / test split has been done differently. Some authors seem to find that there is no risk of data leakage when using transfer learning if the target and the source tasks are different, however it may happen if they share a subset of participants. In this case, participants of the training set of the source task may appear in the test set of the target task. Then they will be used for the evaluation of the network though they have already been used for training.
5. **Biased ensemble learning** occurs when parts of the images are selected / weighted thanks to the labels of the test set to deduce the image-level prediction of the test set. Let's take the example of a CNN trained on a set of patches. In this case, the user will retrieve the label at the image-level based on the combination of the labels of the different patches. However, all patches do not have the same relevance, then the user may want to exclude some patches or weight them differently. This can be done automatically based on the performance value of a patch location which is evaluated on a set of images. But if this set of images includes test images, then the result will be biased.

For clarity, these scenarios are illustrated in Figure 1.

ClinicaDL prevents the user from these scenarios by implementing the following strategies:

1. Data splits are performed at the subject level and cannot be performed on-the-fly but must be done prior to training networks (to avoid a biased split).
2. Data splits are performed independently for each label. However, if labels B & C are subsets of a parent label A, transfer learning from a task implying A to a task implying B and/or C may result in a biased transfer learning. Therefore ClinicaDL splits B and C with respect to A split (see Figure 2 for more insight).
3. In the classification case, the image-level prediction is the weighted sum of parts of the image. These weights are computed from the predictions on the training or the validation sets, but no other set (to avoid biased ensemble learning).
4. At the root of the MAPS, the file `train+validation.tsv` comprises all the participant and session IDs seen during the training procedure. If transfer learning is performed, this list of IDs is updated to include the IDs of participants and sessions seen during the training of the source task. ClinicaDL prevents the user from creating a data group having common IDs with this list (to avoid biased data split and transfer learning).

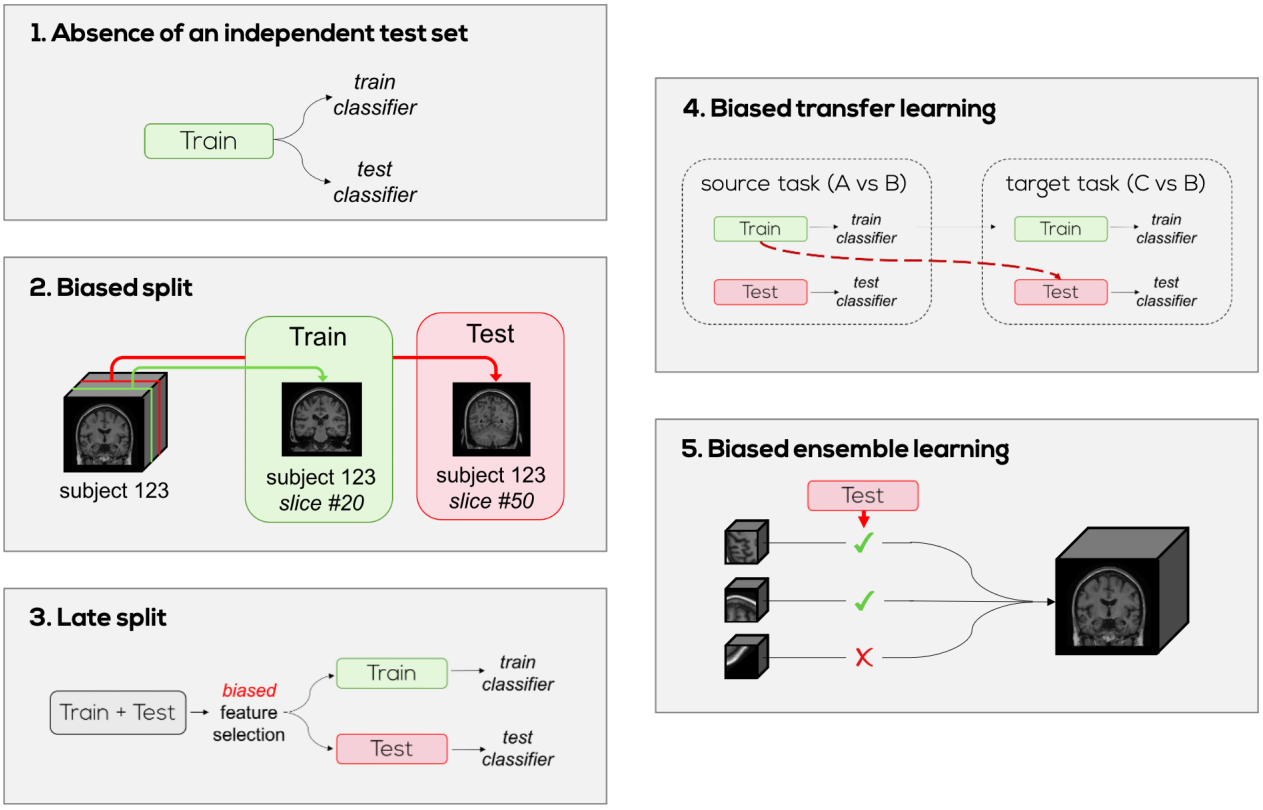

Figure 1: Illustration of the scenarios that can lead to data leakage.

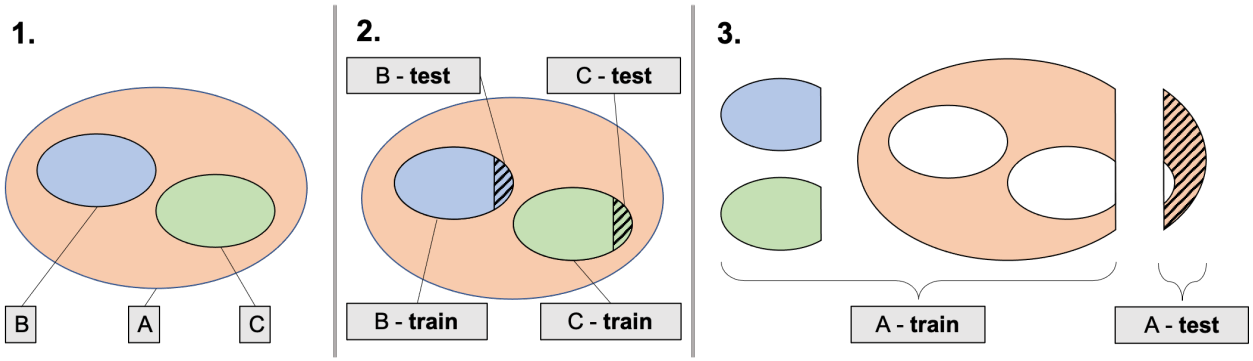

Figure 2: Sequence of data split when diagnostic labels (B and C) are subgroups of another diagnostic label (A). (1) Participants of each group are identified. (2) B and C subgroups are split between train and test data sets. (3) The rest of the parent group is split between train and test, and the train set of each subgroup is added to the parent train set.

The absence of an independent test set is still possible, as ClinicaDL does not force the user to give the test set to evaluate the performance in an unbiased way during training. We do not wish to enforce such system as the user could want to do some hyperparameter optimization based on the training and validation performance only. Then it is the user's responsibility to evaluate the final performance once they have done all the research they wanted on hyperparameters.

2.3. Reproducibility

In the same way as "deep learning", "reproducibility" is an-ill defined concept, often mentioned using similar words (repeatability, reproducibility, replicability) whose meanings may vary across articles. This is why we chose here to work with the definitions of Goodman et al. [21], in which three different levels of reproducibility are defined:

- **method reproducibility** (sometimes called repeatability) is the ability to repeat the same experiment using the same tools and data to obtain the same results,
- **result reproducibility** (sometimes called replicability) is the corroboration of results by other studies using the same experimental methods,
- **inferential reproducibility** (often not discussed) exists when different scientists deduce the same knowledge claims from a similar study or the re-analysis of the study.

As explained by the authors, inferential reproducibility may be impossible to guarantee as the analysis of results is peculiar to each scientist (and this is also what drives science forward).

A description of reproducibility specific to neuroimaging was also proposed by the Organization for Human Brain Mapping [22]. They proposed other levels of reproducibility corresponding to items that can vary (measurements, analyses or materials and methods).

However, for both definitions, none of the reproducibility levels are guaranteed if the research is not transparent. Then what will mostly be discussed here is the way to achieve transparency, which consists in closely describing the different steps linking the prior hypothesis to the final claim.

The initial step to achieve transparency is to share usable code. This way Collberg and Proebsting [23] tried to locate and build the source code of studies of eight ACM conferences. They first noticed that only half of the source codes could be located. Among these codes, half of them could be built easily (i.e. in less than 30 minutes without the authors' help), others needed more time, the help of the authors, or could even fail (in rare cases). To prevent this pitfall, the source code of ClinicaDL is available on GitHub[28]. Moreover, tests are run at each commit to ensure that the code can be correctly installed and that the main functionalities can be run (see Section 3.1 for more information on tests).

However, sharing code is not enough to be fully transparent. For non-deterministic models such as deep learning, method reproducibility can only be achieved by setting a random seed

[28] `https://github.com/aramis-lab/ClinicaDL`

[24, 25]. Crane [24] also evaluated the impact of the computational setup, and explained that the software versions of all the system used, the GPU version and even threading should be explicit to allow method reproducibility. All these variables can be easily set and retrieved when using ClinicaDL. First, the code and documentation of ClinicaDL are versioned to allow the user to retrieve the version needed for method reproducibility. Then, as explained in Section 2.1.4, two files at the root of the experiment folder identify the software and dependencies' versions (`environment.txt`) and variables such as threading, GPU usage and random seed (`maps.json`). Moreover, the function `clinicadl train --config_file` was designed to repeat experiments based on this configuration file (see Section 3.2.5). However, we remind that it is still the users' responsibility to describe their GPU system.

As explained by Beam et al. [25] and Baker [26], documentation is also a crucial point to ensure transparency and code usability by other teams, which then allows result reproducibility. This is why ClinicaDL comes with different documentation supports, including tutorials (see Section 3.1.4).

Finally, Goodman et al. [21] and Stodden et al. [27] encourage others to report all the explored paths and negative results to be more transparent and to avoid potential bias in reporting. This process may also avoid unfair claims (inferential non-reproducibility) based on the comparisons with weak baselines [24]. Indeed, the performance of deep learning systems highly depends on the time spent on their design. This is why it could be interesting to report all the architectures trained to find the final system compared with the ones trained for the baseline one. Again, ClinicaDL allows easily compiling this information as the (hyper)parameters of all the networks trained are saved in their MAPS.

3. ClinicaDL overview

ClinicaDL is an open-source software platform entirely written in Python. It uses the PyTorch library as backbone. ClinicaDL extends PyTorch for neuroimaging applications where the data set structure plays a key role in the organization of the data and metadata. The software is publicly distributed as an easy-to-install package and is referenced in the Pypi package index[29]. Releases are performed on a periodic basis and the code follows the most standard current practices for software development. The functionalities described in this paper correspond to version 1.1.0. For more information on the versions of the dependencies, the reader can refer to the `poetry.lock` file.[30]

ClinicaDL has been designed to be used via the command line interface, with separate sub-commands performing the main tasks, as defined in a classical machine learning pipeline: `extract`, `train`, `predict`. Other sub-commands are available in order to allow the user to structure the data sets, create synthetic data, look for hyperparameters and interpret trained networks. These functionalities are also available through the command line (`tsvtool`,

[29] `https://pypi.org/project/clinicadl`

[30] `https://github.com/aramis-lab/clinicadl/blob/v1.1.0/poetry.lock`

`generate`, `random-search`, `interpret`).

3.1. Development Practices

ClinicaDL has adopted standard practices for software development and distribution of the software with the aim to facilitate the reproduction of experiments. The main functionalities of the software, the management of its inputs/outputs, the data flow and the way the program is used were designed with the objective of staying as close as possible to the definition of reproducibility, as previously given in Section 2.3. The main development practices are described below.

3.1.1. Distribution and Installation

The source code is hosted on Github[31]. It uses a version control system and the releases are strictly labeled with the version number. In consequence, the source code used in a specific experiment can be easily retrieved. Labeled versions of the code are released as Python packages that are permanently stored in the official Python Package Index. Good practices related to the version control system include atomic committing, clear commit messages and peer-reviewed contributions.

The installation of the released packages is performed with a single command (`pip install clinicadl`). As often when installing Python packages, users are advised to install it into a virtual environment to avoid requirement conflicts. Instructions for developer installation are also available in the `README` of the repository.

3.1.2. Continuous Integration and Deployment

Each contribution is peer-reviewed by a developer different from the original author. The resulting code is only integrated to the development branch if the post commit actions are executed in a satisfactory way. The ensemble of these actions is described in the Continuous Integration pipeline. This includes:

- **Environment and dependencies verification**: The creation of an environment with all the dependencies necessary to install the package is performed in this step.
- **User interface tests**: The command line interface is tested using the Pytest library. This library allows combining several sets of possible commands used in the user interface. These are systematically tested to avoid errors in the main interface of ClinicaDL.
- **Functional tests**: A different kind of tests is executed before the integration of new code. These tests are called functional tests and are designed to check for the proper operation of the different functionalities proposed by the software: e.g. "Train", "Transfer Learning", "Interpretation" and "Random Search" tests use a truncated data set to verify that these functionalities run properly on a GPU machine. Other functionalities

[31] `https://github.com/aramis-lab/ClinicaDL`

such as "Predict" to perform inference, "Generate" to create custom data sets or "TSV Tools" to generate files adapted to the task / data set are also checked.

- **Documentation build**: New contributions and/or modifications to the code are expected to be accompanied by the respective documentation. For that reason, documentation is built during the continuous integration pipeline. More details are explained in Section 3.1.4.

- **Deployment**: This step is only executed on labeled commits. Indeed, if a commit has a label to reference a version, a Python package is built and uploaded to the Python Package Index and a new version is published.

3.1.3. Model distribution

The work described in [1] used a preliminary version of ClinicaDL. Several pretrained models generated from the methods described in this paper are available to download via Zenodo[32]. These models are also publicly available via a classical https server[33] to facilitate interactive downloading. New versions of the software may induce changes on the organization of the available models and the way they are loaded and processed by ClinicaDL. For these reasons, new models are trained regularly and stored in folders named with the corresponding software version, e.g. `v1.0.5` corresponds to the models trained with the version 1.0.5 of ClinicaDL. Currently four CNN models for binary classification are available, one for each input mode of ClinicaDL (`image`, `patch`, `roi` and `slice`).

3.1.4. Documentation

The documentation of ClinicaDL is available online at `https://clinicadl.readthedocs.io`. It is automatically built after each commit by Read the Docs[34]. It can also be built locally by running the command `mkdocs serve` with mkdocs-material installed[35].

The documentation is versioned in the same way as the source code. All previous tags are easily accessible online with the version panel in the bottom right corner of any page.

In addition to the user documentation, some tutorials have been created to help users in their first steps with ClinicaDL. These tutorials are designed with Jupyter Books[36], a tool that mixes Markdown content with interactive notebooks. They are referenced in the documentation and GitHub main pages and are accessible online at `https://aramislab.paris.inria.fr/clinicadl/tuto`. The first sections introduce the clinical context of Alzheimer's disease and basics on deep learning classification. The rest of the book is made of interactive notebooks that present the main functionalities of ClinicaDL and can be easily run locally or on Google Colab (which provides free GPU environments).

[32] `https://zenodo.org/record/3491003`
[33] `https://aramislab.paris.inria.fr/clinicadl/files/models/`
[34] `https://readthedocs.org/`
[35] `https://squidfunk.github.io/mkdocs-material/getting-started`
[36] `https://jupyterbook.org/intro.html`

Finally, a discussion forum is available at `https://groups.google.com/g/clinica-user`. Users can interact with the development team. This tool should be replaced soon by `https://github.com/aramis-lab/clinicadl/discussions`.

3.2. Main functionalities

The main functionalities of ClinicaDL cover all the steps needed for deep learning experiments, from data set management to the evaluation of results and network interpretation. ClinicaDL's workflow is illustrated in Figure 3. In addition to pre-implemented options, the source code aims at being modular and the documentation helps users to easily implement their custom experiments[37]. Technical details for each command can be found in the user documentation.

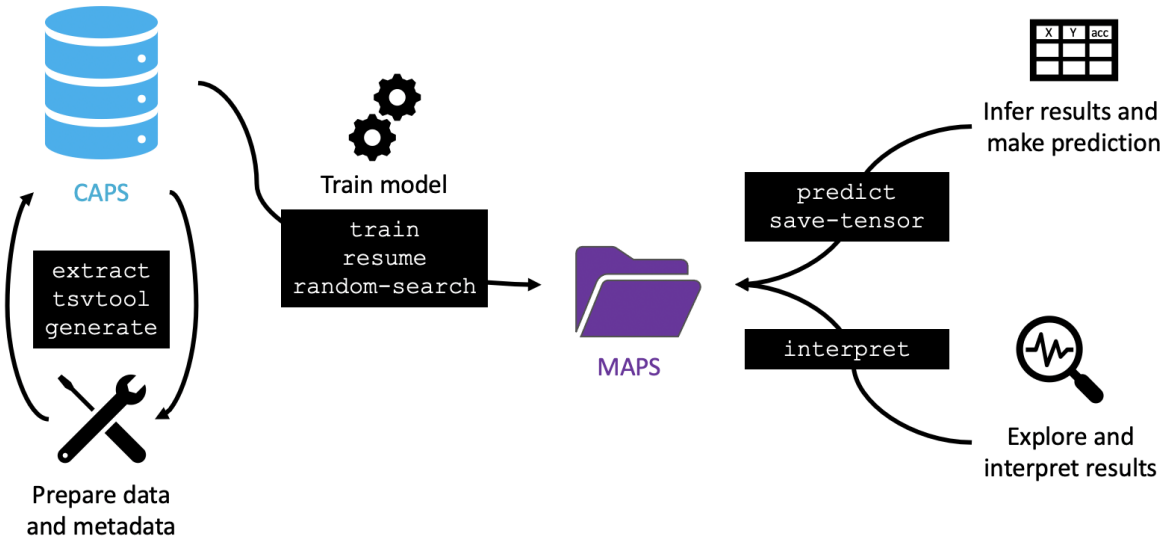

Figure 3: ClinicaDL main functionalities. `extract`, `tsvtool` and `generate` functionalities read and write in the ClinicA Processed Structure (CAPS), which contains neuroimaging data preprocessed by Clinica pipelines. ClinicaDL writes its own output, the Model Analysis and Processing Structure (MAPS), which contains the results of the training phase as well as inference on new data or the results of interpretability methods.

3.2.1. Preprocessing images

ClinicaDL works preferably with images that have been previously preprocessed but one can also perform experiments with unprocessed images, the only requirement is to convert these images to the right format (see Section 2.1.2). Preprocessed images can be obtained using Clinica for different imaging modalities. This software provides, in its current version (0.5), light preprocessing pipelines for T1-weighted MRI and PET images that output images suited for further analysis with deep learning. For example, the `t1-linear` pipeline mainly performs

[37] `https://clinicadl.readthedocs.io/en/latest/Contribute/Custom/`

bias field correction and spatial normalization to the MNI space of T1-weighted MR images, while the `pet-linear` pipeline mainly performs spatial normalization to the MNI space and intensity normalization of PET images. Clinica also provides more extensive preprocessing pipelines for T1-weighted MRI and PET, but also diffusion MRI [8]. As ClinicaDL and Clinica are fully compatible, outputs of the formerly mentioned pipelines can be introduced easily into train or predict functions of ClinicaDL.

ClinicaDL proposes a simple tool to transform NIfTI images into PyTorch tensors. The objective is to facilitate the training phase by decompressing the images beforehand (the NIfTI format usually provides compressed images). This functionality writes future input images for neural network training or inference formatted as tensors. The number and shape of these tensors depend on the mode chosen: `image`, `patch`, `roi` or `slice` (see Section 2.1.3).

The tool will run through the entire CAPS/BIDS folder searching for an imaging modality specified by the user and will apply the conversion and extraction of corresponding images. It will also produce a configuration file summarizing all the characteristics of the extraction procedure. The training procedure will then rely on this file to find the images needed for network training.

3.2.2. Generation of toy data sets

ClinicaDL facilitates the generation of semi-synthetic data for evaluation and verification purposes. The synthetic data can be used to test a binary classification task, and it is already organized in the CAPS format (see Section 2.1.2). Three types of data can be created:

- **Trivial data**: A mask is used to create incomplete images. By default, a mask based on a neuroanatomical atlas is used to create images where only half of the brain is present (half-left or half-right). Other kinds of tampering can be created by supplying a customized mask. The final result is the suppression of the region present in the mask.

- **Random data**: All the images belonging to this type of data are obtained from a single image, adding random white noise. The standard deviation of the noise is a parameter chosen by the user. Resulting images are then randomly distributed between two possible labels.

- **Shepp Logan data**: 2D images whose appearance is based on the Shepp-Logan phantom [28] are generated. The position, size and orientation of the different ellipses mimicking parts of the head (skull, ventricles, regions of interest) are random. In this synthetic data set, two classes are simulated: cognitively normal (CN) and Alzheimer's disease (AD). In the AD class, the size of some regions of interest is smaller than in the CN class (simulation of atrophy). Moreover the AD class can be made of two subtypes, in which this atrophy will not affect the same regions of interest.

These synthetic data sets have different purposes: the first two can be used to debug the functionalities of ClinicaDL and are used by the continuous integration workflow. Trivial data can also be used to ensure that a network is able to learn something from images with

similar characteristics but simpler than the true ones that will be used. Finally, the Shepp Logan data was used to simulate heterogeneous data sets: in this case one of the classes (AD) is made of two different groups. Then this data could be used to evaluate the reliability of deep learning methods on heterogeneous data sets.

3.2.3. Preparing metadata

To use the train and inference functionalities of the software or to analyse the data, inputs must be organized in the right way. A collection of tools to handle metadata of BIDS-formatted data sets is proposed with ClinicaDL. These tools are intended to provide the correct organization of the data: get the labels used in classification tasks, split the data to define test, validation and train subsets, and analyze the population of interest. This set of commands is available through the command `clinicadl tsvtool`. Some of them are still specific to the study of Alzheimer's disease:

- Generation of TSV files including only participants with particular restrictions on two Alzheimer's disease data sets (AIBL and OASIS).
- Extraction of labels specific to a particular diagnosis trajectory (e.g. participants labeled with an Alzheimer's disease diagnosis for all their sessions).

Other commands are more generic, and may be applied to any label list, even if they were not generated with ClinicaDL:

- Splitting labels to produce similar distributions from a specific population using as parameters sex and age.
- Splitting labels to perform k-fold cross validation.
- Writing reports to summarize the demographics and clinical distributions of a specific label.

3.2.4. Random search

Random search [29] is a procedure to find automatically the hyperparameters (architecture and other training hyperparameters) of a framework. It consists in randomly generating sets of hyperparameters to select the best set of hyperparameters as a result. This random generation is based on a hyperparameter space from which hyperparameter sets are sampled. In ClinicaDL, this hyperparameter space is described by a configuration file created by the user.

The main advantage of the random search is its easy parallelization, contrary to other optimization methods that may require successive runs and be time consuming. On the other hand, it is computationally costly and it requires minimum knowledge regarding the subspace of hyperparameters that may work to limit the search and find satisfactory results. Moreover, although it can significantly improve the performance of a framework, it will not lead to the optimum, which is very hard to find.

It is also possible to improve the results of a random search by using its results to initialize another technique (genetic algorithm, Bayesian optimization, etc.).

3.2.5. Training networks

The main functionality of ClinicaDL is to train neural networks to learn a task. These tasks can be:

1. **Classification** (of a categorical label, for example the diagnosis),
2. **Regression** (of a continuous label, for example the age),
3. **Image reconstruction**.

Segmentation is currently not handled by ClinicaDL. However, as the software is meant to be extensible, new tasks can be easily added by advanced users (see Section 4).

These tasks are highly dependent from the architecture. All tasks take as input the image or part of it (see Section 2.1.3); but for classification the output is a flattened array of size equal to the number of classes, for regression it is a single node, whereas for image reconstruction it has the same size as the input. When the user chooses a task and an architecture, the software will check if the task is compatible with the wanted architecture and will raise an error if it is not the case.

Some pre-built deep learning architectures for each task are available in ClinicaDL and their list and details can be displayed with the command `clinicadl train list_models`. However, an objective of the library is to allow the users to add and use their custom architectures easily. To this end, users can implement their custom networks by filling the abstract template, which includes specific methods that are used in ClinicaDL. The procedure of such addition is detailed in the documentation.

The models produced by ClinicaDL correspond to the ones that obtained the best performance on the validation set according to metrics chosen by the user. ClinicaDL saves at the end of each epoch the state of the network and of the optimizer. For each selection metric given in input, it replaces the corresponding current best model by the current state if the performance on the validation set is better than the current best value. To minimize the size of the produced MAPS, the checkpoints are deleted at the end of the training procedure. They are only used to resume a stopped job, thanks to the dedicated command `resume`.

The command line interface of ClinicaDL offers many options, as there is a large number of training parameters. This is why we tend to a parametrization by configuration files only. Currently, it is already possible to train a network parametrized by a configuration file instead of entering each parameter individually in the command line using `clinicadl train --config_file FILENAME`.

3.2.6. Performance evaluation

ClinicaDL provides specific functions to easily perform inference with models previously trained with the tool. This functionality is available in a specific sub menu of the command line (`clinicadl predict`). For example, one may want to evaluate the performance of a

trained model on a set of new samples. In this case, the command will load the best model, the input images (in a BIDS/CAPS-like format) and the list of subjects of the data group. Trained models are available within the MAPS produced during the training and the other information can be either integrated into this structure or provided as a command line option. The results are written in the MAPS as pre-formatted reports with the metric values at different levels (e.g. image-level and patch-level) and the output values computed for each input image of the data group.

The metrics computed depend on the task learnt by the network. The regression and reconstruction tasks are associated with the mean squared error and mean absolute error, whereas the classification task is evaluated thanks to balanced accuracy, accuracy, sensitivity, specificity, positive and negative predictive values. Advanced users can add any new metric by following the procedure described in the advanced user guide. Moreover as the output values are computed for each input image individually, users can easily compute any metric of evaluation without modifying the source code.

3.2.7. Interpretation

The most critical issue of deep learning methods is their lack of transparency. This is why some interpretability methods have been developed specifically for the field. These methods allow better understanding which patterns or zones of the images have been linked to the result produced by the network. Currently, only the gradient back-propagation method proposed in [30] is implemented in ClinicaDL. We plan to strengthen the content of this command in future releases.

4. Discussion

In this paper we presented ClinicaDL, a Python open-source software for neuroimaging data processing with deep learning. This software includes many functionalities, such as neuroimaging preprocessing, synthetic dataset generation, label definition, data split with similar demographics, architecture search, network training, performance evaluation and trained network interpretation. The three main objectives of ClinicaDL are to (1) help manipulate neuroimaging data sets, (2) prevent data leakage from biasing results and (3) reproduce deep learning experiments.

First, ClinicaDL relies on BIDS and CAPS formats to organize raw and processed data, respectively. These formats were first introduced for neuroimaging data management but they can be easily extended to any kind of medical imaging data, as it would only require renaming and formatting files of a data set. However, even though this data organization allows the user to easily test a large variety of options, (for example keeping the NIfTI versions of preprocessed images allow re-extracting a new type of tensors for another set of experiments), its main drawback is that the combination of these structures is storage-consuming. Indeed, this leads at least to the backup of three files per participant: one compressed NIfTI file for the raw version (BIDS) and a compressed NIfTI file with its tensor derivative for the preprocessed version (CAPS).

Secondly, ClinicaDL prevents data leakage as train and validation data characteristics are saved when the output structure (MAPS) is created. Then, when evaluating the performance of a trained model on a new data group, ClinicaDL checks that this data group does not share participants with the training and validation groups. However, this only works under the assumption that participants are always named in the same way across data groups.

Thirdly, ClinicaDL improves deep learning experiment reproducibility by sharing usable and tagged code, saving all parameters of the training set and data groups used for evaluation, and providing extensive documentation. But even though all these elements improve method reproducibility, reproducibility can still be easily broken. For example it is not possible to obtain the same results using different machines. However, one may be interested in having a deterministic setting on a given machine to correctly evaluate the impact of a particular property to improve their performance, and this is possible with ClinicaDL. Moreover, result reproducibility may also be broken by manual architecture search and the overuse of the same data set [31]. Indeed, research studies may be globally overfitting their data set and if one day another data set is released, performance of previous studies may collapse. This is why we implemented the random search method, although its very high computational cost may limit its reproducibility power.

Among the three main issues tackled by ClinicaDL, the one which is the most often addressed by other tools is the first (data management). For example, TorchIO and Monai ease the use of some public data sets (MedMNIST, medical segmentation decathlon challenge, IXI and EPISURF), then other data sets can be plugged to the library components by specifying individually the paths to images and labels (Nobrainer only offers this second option), which is not necessary in ClinicaDL. This way, newcomers can easily begin to handle the libraries by running examples based on integrated data sets, and then try to use their own. The main default of this system is the lack of reproducibility: the list of participants used must be saved by the user independently, and the preprocessing information may be lost. This is not the case with ClinicaDL as the characteristics of each group are saved in the MAPS and the preprocessing is fully described in the configuration file. TorchIO and Monai also deal with reproducibility: TorchIO guarantees that transforms with a random factor can be reproduced as one can get the transforms' history, and Monai allows setting a random seed to compute a deterministic training. However, they do not propose any system similar to the MAPS, thus experiment settings and environment versions may be lost by the user. Finally, none of the libraries reviewed proposed systems to avoid data leakage, though it is a crucial issue in our domain.

As exposed in Section 3.2, the association of Clinica and ClinicaDL covers a large variety of procedures needed in deep learning experiments, that starts from the raw data format and ends with network interpretability. On the contrary, TorchIO is more specialized as it focuses on medical imaging transforms, particularly for data augmentation. This focus results in a larger amount of options in this particular domain. This is why we consider integrating modules from TorchIO for data augmentation in the future. Monai is more complete with the possibility to transform images, and train, evaluate and interpret networks. They also

provide a large amount of options for some functionalities which are not customizable yet in ClinicaDL. We would also like to enable the parametrization of more functionalities, whose options could be easily added by advanced users. Moreover, Monai implements multi-GPU processing, which is not the case in ClinicaDL yet, and that we may also implement by relying on torch-lightning[38]. Finally, future work should propose an evolution of the MAPS structure in order to integrate new logging tools and facilitate the tracking of the training parameters and the automatic management of the generated models.

ClinicaDL aims at being flexible, thus a section of the documentation is dedicated to the addition of new options on top of the main functionalities. Currently, it is possible to customize:

- the architecture of the network (dependent from the task learnt),
- the mode extracted from the 3D image (current options: image, patch, roi, slice),
- the task learnt by the network (current options: classification, regression, image reconstruction),
- the metrics used for evaluation and best model selection (dependent from the task learnt).

Even if some options are not already integrated to the framework, we hope that advanced users will be prone to propose new pipelines to the repository.

5. Conclusion

In this paper we presented ClinicaDL, an open-source software for deep learning processing on neuroimaging data. With this software, we help deep learning users handling the three main issues encountered by non-specialists of the neuroimaging domain: (1) the data management and preprocessing of neuroimaging data sets, (2) the contamination of results by data leakage and (3) the lack of reproducibility of deep learning experiments.

Acknowledgments

The authors would like to thank Junhao Wen for his initial contribution to what became ClinicaDL, Simona Bottani and Omar El Rifai for their contributions and feedback, and Igor Koval for his soothing support.

Statements of ethical approval

Not applicable to this study.

[38] `https://www.pytorchlightning.ai`

Funding

The research leading to these results has received funding from the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and reference ANR-10-IAIHU-06 (Agence Nationale de la Recherche-10-IA Institut Hospitalo-Universitaire-6).

Competing interests

The authors do not have any competing interests to declare.

References

[1] J. Wen, E. Thibeau-Sutre, M. Diaz-Melo, J. Samper-González, A. Routier, S. Bottani, D. Dormont, S. Durrleman, N. Burgos, O. Colliot, Convolutional neural networks for classification of Alzheimer's disease: Overview and reproducible evaluation, Medical Image Analysis 63 (2020) 101694. doi:10.1016/j.media.2020.101694.

[2] R. K. Samala, H.-P. Chan, L. Hadjiiski, S. Koneru, Hazards of data leakage in machine learning: a study on classification of breast cancer using deep neural networks, in: Medical Imaging 2020: Computer-Aided Diagnosis, Vol. 11314, International Society for Optics and Photonics, 2020, p. 1131416. doi:10.1117/12.2549313.

[3] H. Panwar, P. K. Gupta, M. K. Siddiqui, R. Morales-Menendez, V. Singh, Application of deep learning for fast detection of COVID-19 in X-Rays using nCOVnet, Chaos, Solitons & Fractals 138 (2020) 109944. doi:10.1016/j.chaos.2020.109944.

[4] N. Bussola, A. Marcolini, V. Maggio, G. Jurman, C. Furlanello, AI slipping on tiles: data leakage in digital pathology (2020). arXiv:1909.06539.

[5] E. Yagis, S. W. Atnafu, A. G. S. de Herrera, C. Marzi, M. Giannelli, C. Tessa, L. Citi, S. Diciotti, Deep Learning in Brain MRI: Effect of Data Leakage Due to Slice-Level Split Using 2D Convolutional Neural Networks, Preprint, In Review (2021). doi:10.21203/rs.3.rs-464091/v1.

[6] M. Hutson, Artificial intelligence faces reproducibility crisis, Science 359 (6377) (2018) 725–726. doi:10.1126/science.359.6377.725.

[7] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko, D. A. Handwerker, M. Hanke, D. Keator, X. Li, Z. Michael, C. Maumet, B. N. Nichols, T. E. Nichols, J. Pellman, J.-B. Poline, A. Rokem, G. Schaefer, V. Sochat, W. Triplett, J. A. Turner, G. Varoquaux, R. A. Poldrack, The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments, Scientific Data 3 (2016) 160044. doi:10.1038/sdata.2016.44.

[8] A. Routier, N. Burgos, M. Díaz, M. Bacci, S. Bottani, O. El-Rifai, S. Fontanella, P. Gori, J. Guillon, A. Guyot, R. Hassanaly, T. Jacquemont, P. Lu, A. Marcoux, T. Moreau, J. Samper-González, M. Teichmann, E. Thibeau-Sutre, G. Vaillant, J. Wen, A. Wild, M.-O. Habert, S. Durrleman, O. Colliot, Clinica: An Open-Source Software Platform for Reproducible Clinical Neuroscience Studies, Frontiers in Neuroinformatics 15 (2021) 39. doi:10.3389/fninf.2021.689675.

[9] K. J. Gorgolewski, F. Alfaro-Almagro, T. Auer, P. Bellec, M. Capotă, M. M. Chakravarty, N. W. Churchill, A. L. Cohen, R. C. Craddock, G. A. Devenyi, A. Eklund, O. Esteban, G. Flandin, S. S. Ghosh, J. S. Guntupalli, M. Jenkinson, A. Keshavan, G. Kiar, F. Liem, P. R. Raamana, D. Raffelt, C. J. Steele, P.-O. Quirion, R. E. Smith, S. C. Strother, G. Varoquaux, Y. Wang, T. Yarkoni, R. A. Poldrack, BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods, PLOS Computational Biology 13 (3) (2017) e1005209. doi:10.1371/journal.pcbi.1005209.

[10] P. Lakhani, D. L. Gray, C. R. Pett, P. Nagy, G. Shih, Hello World Deep Learning in Medical Imaging, Journal of Digital Imaging 31 (3) (2018) 283–289. doi:10.1007/s10278-018-0079-6.

[11] E. Gibson, W. Li, C. Sudre, L. Fidon, D. I. Shakir, G. Wang, Z. Eaton-Rosen, R. Gray, T. Doel, Y. Hu, T. Whyntie, P. Nachev, M. Modat, D. C. Barratt, S. Ourselin, M. J. Cardoso, T. Vercauteren, NiftyNet: a deep-learning platform for medical imaging, Computer Methods and Programs in Biomedicine 158 (2018) 113–122. doi:10.1016/j.cmpb.2018.01.025.

[12] A. Beers, J. Brown, K. Chang, K. Hoebel, J. Patel, K. I. Ly, S. M. Tolaney, P. Brastianos, B. Rosen, E. R. Gerstner, J. Kalpathy-Cramer, DeepNeuro: An open-source deep learning toolbox for neuroimaging, Neuroinformatics 19 (1) (2021) 127–140. doi:10.1007/s12021-020-09477-5.

[13] N. Pawlowski, S. I. Ktena, M. C. H. Lee, B. Kainz, D. Rueckert, B. Glocker, M. Rajchl, DLTK: State of the Art Reference Implementations for Deep Learning on Medical Images (2017). arXiv:1711.06853.

[14] M. Antonelli, A. Reinke, S. Bakas, K. Farahani, AnnetteKopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, B. van Ginneken, M. Bilello, P. Bilic, P. F. Christ, R. K. G. Do, M. J. Gollub, S. H. Heckers, H. Huisman, W. R. Jarnagin, M. K. McHugo, S. Napel, J. S. G. Pernicka, K. Rhode, C. Tobon-Gomez, E. Vorontsov, H. Huisman, J. A. Meakin, S. Ourselin, M. Wiesenfarth, P. Arbelaez, B. Bae, S. Chen, L. Daza, J. Feng, B. He, F. Isensee, Y. Ji, F. Jia, N. Kim, I. Kim, D. Merhof, A. Pai, B. Park, M. Perslev, R. Rezaiifar, O. Rippel, I. Sarasua, W. Shen, J. Son, C. Wachinger, L. Wang, Y. Wang, Y. Xia, D. Xu, Z. Xu, Y. Zheng, A. L. Simpson, L. Maier-Hein, M. J. Cardoso, The Medical Segmentation Decathlon (2021). arXiv:2106.05735.

[15] A. L. Simpson, M. Antonelli, S. Bakas, M. Bilello, K. Farahani, B. van Ginneken, A. Kopp-Schneider, B. A. Landman, G. Litjens, B. Menze, O. Ronneberger, R. M. Summers, P. Bilic, P. F. Christ, R. K. G. Do, M. Gollub, J. Golia-Pernicka, S. H. Heckers, W. R. Jarnagin, M. K. McHugo, S. Napel, E. Vorontsov, L. Maier-Hein, M. J. Cardoso, A large annotated

medical image dataset for the development and evaluation of segmentation algorithms (2019). arXiv:1902.09063.

[16] F. Pérez-García, R. Sparks, S. Ourselin, TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning, Computer Methods and Programs in Biomedicine 208 (2021) 106236. doi:10.1016/j.cmpb.2021.106236.

[17] F. Pérez-García, R. Rodionov, A. Alim-Marvasti, R. Sparks, J. Duncan, S. Ourselin, EPISURG: a dataset of postoperative magnetic resonance images (MRI) for quantitative analysis of resection neurosurgery for refractory epilepsy (2020). doi:10.5522/04/9996158.v1.

[18] G. Lee, B. Kang, K. Nho, K.-A. Sohn, D. Kim, MildInt: Deep Learning-Based Multimodal Longitudinal Data Integration Framework, Frontiers in Genetics 10 (2019). doi:10.3389/fgene.2019.00617.

[19] A. Jungo, O. Scheidegger, M. Reyes, F. Balsiger, Pymia: A Python package for data handling and evaluation in deep learning-based medical image analysis, Computer Methods and Programs in Biomedicine 198 (2021) 105796. doi:10.1016/j.cmpb.2020.105796.

[20] S. Kaufman, S. Rosset, C. Perlich, O. Stitelman, Leakage in data mining: Formulation, detection, and avoidance, ACM Transactions on Knowledge Discovery from Data 6 (4) (2012) 15:1–15:21. doi:10.1145/2382577.2382579.

[21] S. N. Goodman, D. Fanelli, J. P. A. Ioannidis, What does research reproducibility mean?, Science Translational Medicine 8 (341) (2016) 341ps12–341ps12. doi:10.1126/scitranslmed.aaf5027.

[22] T. E. Nichols, S. Das, S. B. Eickhoff, A. C. Evans, T. Glatard, M. Hanke, N. Kriegeskorte, M. P. Milham, R. A. Poldrack, J.-B. Poline, E. Proal, B. Thirion, D. C. Van Essen, T. White, B. T. T. Yeo, Best practices in data analysis and sharing in neuroimaging using MRI, Nature Neuroscience 20 (3) (2017) 299–303. doi:10.1038/nn.4500.

[23] C. Collberg, T. A. Proebsting, Repeatability in computer systems research, Communications of the ACM 59 (3) (2016) 62–69. doi:10.1145/2812803.

[24] M. Crane, Questionable Answers in Question Answering Research: Reproducibility and Variability of Published Results, Transactions of the Association for Computational Linguistics 6 (2018) 241–252. doi:10.1162/tacl_a_00018.

[25] A. L. Beam, A. K. Manrai, M. Ghassemi, Challenges to the Reproducibility of Machine Learning Models in Health Care, JAMA 323 (4) (2020) 305–306. doi:10.1001/jama.2019.20866.

[26] M. Baker, 1,500 scientists lift the lid on reproducibility, Nature News 533 (7604) (2016) 452. doi:10.1038/533452a.

[27] V. Stodden, M. McNutt, D. H. Bailey, E. Deelman, Y. Gil, B. Hanson, M. A. Heroux, J. P. A. Ioannidis, M. Taufer, Enhancing Reproducibility for Computational Methods, Science 354 (6317) (2016) 1240–1241. doi:10.1126/science.aah6168.

[28] L. A. Shepp, B. F. Logan, The Fourier reconstruction of a head section, IEEE Transactions on Nuclear Science 21 (3) (1974) 21–43. doi:10.1109/TNS.1974.6499235.

[29] J. Bergstra, Y. Bengio, Random Search for Hyper-Parameter Optimization, Journal of Machine Learning Research 13 (10) (2012) 281–305.
URL http://jmlr.org/papers/v13/bergstra12a.html

[30] K. Simonyan, A. Vedaldi, A. Zisserman, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps (2013). arXiv:1312.6034.

[31] W. H. Thompson, J. Wright, P. G. Bissett, R. A. Poldrack, Dataset decay and the problem of sequential analyses on open datasets, eLife 9 (2020) e53498. doi:10.7554/eLife.53498.