pymia: A Python package for data handling and evaluation in deep learning-based medical image analysis

Alain Jungo\textsuperscript{a,*}, Olivier Scheidegger\textsuperscript{b,c}, Mauricio Reyes\textsuperscript{a}, Fabian Balsiger\textsuperscript{a,*}

\textsuperscript{a}ARTORG Center for Biomedical Engineering Research, University of Bern, Bern, Switzerland
\textsuperscript{b}Department of Neurology, Inselspital, Bern University Hospital, University of Bern, Bern, Switzerland
\textsuperscript{c}Support Center for Advanced Neuroimaging (SCAN), Institute for Diagnostic and Interventional Neuroradiology, Inselspital, Bern University Hospital, University of Bern, Bern, Switzerland

Abstract

Background and Objective: Deep learning enables tremendous progress in medical image analysis. One driving force of this progress are open-source frameworks like TensorFlow and PyTorch. However, these frameworks rarely address issues specific to the domain of medical image analysis, such as 3-D data handling and distance metrics for evaluation. pymia, an open-source Python package, tries to address these issues by providing flexible data handling and evaluation independent of the deep learning framework.

Methods: The pymia package provides data handling and evaluation functionalities. The data handling allows flexible medical image handling in every commonly used format (e.g., 2-D, 2.5-D, and 3-D; full- or patch-wise). Even data beyond images like demographics or clinical reports can easily be integrated into deep learning pipelines. The evaluation allows stand-alone result calculation and reporting, as well as performance monitoring during training using a vast amount of domain-specific metrics for segmentation, reconstruction, and regression.

Results: The pymia package is highly flexible, allows for fast prototyping, and reduces the burden of implementing data handling routines and evaluation methods. While data handling and evaluation are independent of the deep learning framework used, they can easily be integrated into TensorFlow and PyTorch pipelines. The developed package was successfully used in a variety of research projects for segmentation, reconstruction, and regression.

Conclusions: The pymia package fills the gap of current deep learning frameworks regarding data handling and evaluation in medical image analysis. It is available at https://github.com/rundherum/pymia and can directly be installed from the Python Package Index using pip install pymia.

Keywords: Medical image analysis, deep learning, data handling, evaluation, metrics

1. Introduction

Deep learning has a tremendous impact on medical image analysis tasks like classification, segmentation, and reconstruction from 2015 onwards [1, 2, 3, 4]. This impact is mainly due to methodological developments like the AlexNet [5] or the U-Net [6], dedicated hardware (graphics processing units, GPUs), increased data availability, and open-source deep learning frameworks. In fact, open-source deep learning frameworks can be seen as one of the main driving forces leading to the wider adoption of deep learning in the medical image analysis community [1]. Current frameworks like TensorFlow [7] and PyTorch [8] allow researches to implement methods rather than implementing low-level GPU operations. Nevertheless, the adoption of deep learning methods, usually originating from the computer vision community, is often hindered by the 3-D nature of medical images, making, in particular, the data handling and evaluation very domain-specific and cumbersome.

A few open-source projects addressing medical image analysis with deep learning exist. The most
prominent project is likely NiftyNet [9], which enables fast development of medical image analysis methods based on the TensorFlow framework. Among others, it provides implementations of training routines, neural network architectures, and loss functions. Unfortunately, the project is not actively maintained anymore as of April 2020.

Similarly to NiftyNet, the deep learning toolkit (DLTK) [10] also provides implementations of common neural network architectures based on TensorFlow. But the last updates to the project date over a year back and it is incompatible with version 2 of TensorFlow. A more recent package is TorchIO [12], which provides pre-processing and data augmentation routines for medical images, as well as 3-D patch-based data handling within the scope of the PyTorch framework. MONAI (Medical Open Network for AI) is a PyTorch-based framework for deep learning in healthcare imaging. It is the predecessor of NiftyNet, and similarly, MONAI provides training routines, neural network architectures, and loss functions enabling entire deep learning pipelines from data loading to saving. Another framework is DeepNeuro [13], which provides a templating language for designing medial image analysis pipelines and a model deployment system based on TensorFlow. In summary, multiple open-source projects aim at facilitating deep learning-based medical image analysis by providing out-of-the-box training routines and neural network architectures. To date, TorchIO, MONAI, and DeepNeuro seem to be actively developed and the most prominent projects. Unfortunately, all projects rely on one particular deep learning framework (TensorFlow or PyTorch), making it potentially inflexible for fast switch to another framework.

The evaluation of results in medical image analysis is dependent on domain-specific metrics, also due to the physical properties of medical images such as the spacing between pixels. Prominent metrics are, for instance, the Dice coefficient [14] or the Hausdorff distance [15] for segmentation, and the peak signal-to-noise ratio or the structural similarity index measure [16] for image reconstruction. Such metrics are rarely found to be implemented in open-source deep learning frameworks, nor do the projects introduced in the last paragraph provide (exhaustive) implementations of metrics. Therefore, metrics are often taken from multiple independent projects. Notable projects covering metrics are certainly the Insight Toolkit (ITK) [17] with its Python variant SimpleITK [18] covering common segmentation metrics. Furthermore, the evaluate segmentation tool [19] provides an extensive implementation of segmentation metrics [3]. However, the project is C++-based, making it impractical to use with the current Python-based deep learning. A Python-based package is medpy [4] which features a small set of segmentation metrics. And, metrics beyond segmentation can be found in the Python packages scikit-image [20], scikit-learn [21], and SciPy [22]. Overall, a single Python package covering an exhaustive amount of metrics for segmentation, reconstruction, and regression in medical image analysis is lacking.

We believe that deep learning framework-agnostic data handling and evaluation is essential for medical image analysis research. In data handling, flexibility is highly desirable, meaning a simple and fast switch from, e.g., 2-D to 3-D processing, should be possible. For evaluation, performance monitoring during method development, and result calculation and reporting for further statistical analyses and visualization, encompassing domain-specific metrics with aspects like image spacing, is desirable. Ideally, the evaluation is completely decoupled from the deep learning frameworks such that it can be used for evaluation scripts only. Generally for prototyping, rewriting code when methods are adopted from open-source methods implemented in an arbitrary framework should not be necessary. Rather, the relevant code (i.e., the model, loss function, and optimizer), should be copied into an existing data handling and evaluation pipeline with minor to no adaptations of the existing code.

We present pymia, an open-source Python (py) package for deep learning-based medical image analysis (mia). The package addresses two main parts of deep learning pipelines: data handling and evaluation. The package is independent of the deep learning framework used but can easily be integrated into TensorFlow and PyTorch pipelines. Therefore, pymia is highly flexible, allows for fast

https://github.com/NifTK/NiftyNet
https://monai.io/
https://github.com/Visceral-Project/EvaluateSegmentation
https://loli.github.io/medpy/
prototyping, and facilitates implementing data handling and evaluation.

2. Methods

The intended use of pymia in the deep learning environment is depicted in Fig. 1. Its main components are the data and the evaluation package. The data package is used to extract data (images, labels, demography, etc.) from a dataset in the desired format (2-D, 3-D; full- or patch-wise) for feeding to a neural network. The output of the neural network is then assembled back to the original format before extraction, if necessary. The evaluation package provides both evaluation routines as well as metrics to assess predictions against references. These can be used both for stand-alone result calculation and reporting, and for monitoring of the training progress.

2.1. Data package

The purpose of the data package is to provide flexible, format independent, and fast access to data. First, flexible because the data should be accessible in various ways. Meaning that 3-D medical data like magnetic resonance (MR) or computed tomography (CT) images could be processed in 2-D, 3-D, or 2.5-D (i.e., the three anatomical planes axial, coronal, and sagittal) and further in its full or reduced spatial extent, i.e., as so-called patches. Second, the more format-independent the data access, the easier becomes prototyping and experimenting with clinical data beyond medical images. Meaning that demographic information, patient records, or even more uncommon formats such as electroencephalogram (EEG) data, laboratory results, point clouds, or meshes should be accessible. Third, fast because the data access should not slow down the training of the neural network, i.e., not resulting in idle GPU time. The three main components of the data package are creation, extraction, and assembly (Fig. 2), which are described hereafter.

Creation. A dataset is first created from the raw data, which can be seen as a database holding all information available or required for the training of a neural network. This dataset is a HDF5 (hierarchical data format version 5) file. The HDF format allows multiple different data types in one file and enables fast access of chunks of data without the need to load the data in its entirety (e.g., loading of a 2-D image slice from a 3-D image). The creation of a dataset is managed by the Traverser class, which processes the data of every subject (case) iteratively. It employs Load to load the raw data from the file system and Callback classes to write the required information to the dataset. Transform classes can be used to apply modifications to the data, e.g., an intensity normalization. By separating the concerns of the loading, writing, and transforming, maximal flexibility in the dataset creation is achieved. For the ease of use, default Callback and Load classes are implemented, which cover the most fundamental cases. By design, the dataset should only be created once and should, thus, contain as much information as possible. It might be suitable to create three distinct datasets for the training, validation, and testing subjects.

Extraction. Once the dataset is created, it can be used for the training (or testing) routine. Data extraction from the dataset is managed by the PymiaDatasource class, which provides a flexible interface for retrieving data, or chunks of data, to form training samples. An IndexingStrategy is used to define how the data is indexed, meaning accessing, for instance, an image slice or a 3-D patch of an 3-D image. Extractor classes extract the data from the dataset, and Transform classes can be used to alter the extracted data. Processing medical images in chunks is typically required in deep learning due to the size of the images and the limitations in GPU memory. The IndexingStrategy provides a signature for any kind of chunks, e.g., 2-D image slices (SliceIndexing class) or 3-D patches of arbitrary size (PatchWiseIndexing class). It is sufficient to simply exchange the IndexingStrategy if, for instance, another indexing is desired. For each type of data in the dataset, a specific Extractor is used, e.g., a DataExtractor to extract the image data or a SubjectExtractor to extract the identification of a subject. In a sense, an Extractor is the reading counterpart to a Callback for writing during the dataset creation. Since Extractors are the first instance interacting with the data, they can also be used to perform specific data handling, such as padding (PadDataExtractor class) or selecting specific channels (e.g., different MR images) of the data.

---

5 Although in 3-D a (sub)volume would be a more appropriate term, it is often referred as a (3-D) patch in the literature.
Table 1: Overview of use cases for data handling and the corresponding classes to use. Slice: 2-D image slice of a 3-D image; Slab: Multiple consecutive 2-D image slices; 2.5-D: 2-D image slices in all three anatomical planes; Patch (equal): 3-D patch for a neural network with equal input and output size; Patch (padded): 3-D patch for a neural network with larger input than output size (overlapping inputs); Raw format: entire 3-D/2-D image.

| Use case       | IndexingStrategy | Extractor       | Assembler      |
|----------------|------------------|-----------------|----------------|
| Slice          | SliceIndexing    | DataExtractor   | SubjectAssembler |
| Slab           | PatchIndexing    | DataExtractor   | SubjectAssembler |
| 2.5-D          | SliceIndexing    | DataExtractor   | PlaneSubjectAssembler |
| Patch (equal)  | PatchWiseIndexing| DataExtractor   | SubjectAssembler |
| Patch (padded) | PatchWiseIndexing| PadDataExtractor| SubjectAssembler |
| Raw format     | EmptyIndexing    | DataExtractor   | -              |

(SelectiveDataExtractor class). Finally, the extracted data can be altered via Transform classes. Often, these are used to adapt the data for usage with a neural network (e.g., channel permutations, dimension modifications, or intensity modifications) and to alter the data for training purposes (e.g., data augmentation, masking).

Assembly. The output of a neural network usually needs to be assembled back to the original format for evaluation and storage, especially for validation and testing. For instance, a 3-D image instead of separate 2-D image slices are desired when chunks of data are predicted. The Assembler class manages the assembly of the predicted neural network outputs by using the identical indexing that was employed to extract the data by the PymiaDatasource class.

2.1.1. Flexibility & extendability

The modular design of the data package aims at providing high flexibility and extendability to as many use cases as possible. The flexibility is illustrated in Table 1 with use cases of data handling. Well-defined interfaces facilitate the extendability of creation, extraction, and assembly. For the creation of the dataset, new data formats (e.g., EEG, laboratory results) can be handled by a custom Load and might require custom Callback and Extractor implementations. Further, current indexing possibilities can easily be extended with a custom IndexingStrategy. Likewise, one can add customized data modifications by implementing a specific Transform.
Figure 2: Overview of the three main components of the data package. Arrows represent data flow, and the boxes represent class signatures.

2.1.2. Metadata dataset

The data is ideally written to a dataset, as described beforehand. However, there might be use cases such as a large amount of data or the use of very large patch sizes (or even entire 3-D images), which might question the usefulness of creating a dataset, i.e., ultimately only saving the data in another format. Usage of the data package without the creation of a dataset while having the same flexibility as with a dataset is possible. However, the minimum required information in a dataset is fairly small such that the data package can be used as intended. Only the metadata describing the subject identifiers, the file paths, and the shapes (size) of the image data need to be saved into the dataset, resulting in a metadata dataset. The PymiaDatasource class can then be parametrized to load the data from the file system instead from the dataset. The shapes are required such that the flexibility with the IndexingStrategy classes is retained.

2.1.3. Reproducibility & privacy

Reproducibility and privacy might be two important aspects when creating a dataset. Regarding reproducibility, creating a dataset allows writing the names and paths of the files stored in the dataset, which in many cases might be sufficient for reproducibility. For additional reproducibility, it would also be possible to store, for example, the hash value of the raw files, which would allow to verify at any time if a certain raw file was used to create and/or is contained in the dataset. Regarding privacy, as simple as it is to add additional information like the hash value, as simple can data be omitted when creating the dataset. For example, datasets can be created with image data only, and subject identifiers could simply be anonymized. Additionally, the concept of the transformation (Transform classes) would allow to apply image anonymization methods when creating the dataset, e.g., a defacing transform for head images.

2.2. Evaluation package

The purpose of the evaluation package is domain-specific evaluation for medical image analysis. Therefore a variety of metrics for image segmentation, image reconstruction, and regression are included. The functionalities of the evaluation package allow stand-alone result calculation and reporting, or performance monitoring during the training progress independent of the deep learning framework. The concept of the evaluation package is illustrated in Fig. 3. The metrics inherit from Metric and can be used with the Evaluator class to evaluate predictions against references. For instance, the SegmentationEvaluator class can be
Evaluation
prediction
Metric
Segmentation
Reconstruction
evaluation results
Evaluator
reference
Writer
CSV
Console
evaluation results

Figure 3: Overview of the evaluation package. Arrows represent data flow, and the boxes represent class signatures.

used to compare a prediction with a reference label image by calculating the metric(s) for every label one is interested in. The results can then be passed to a Writer to report the results. Currently, a CSVWriter class, writing results to a comma-separated values (CSV) file, and a ConsoleWriter class, writing results to the console, are implemented. Further, statistics over all evaluated subjects (and labels) can be calculated and written by using a CSVStatisticsWriter or a ConsoleStatisticsWriter. In both cases, the statistical functions can be arbitrary, with the only condition being to take a list of values and to return a scalar value (e.g., the mean or the standard deviation).

A variety of metrics are implemented (Table 2), which are categorized into categorical, i.e., for image segmentation, and continuous, i.e., for image reconstruction and regression. All metrics are implemented such that they work with at least 2-D and 3-D data, and if appropriate, also with lower or higher dimensions. Further, image spacing is considered whenever adequate (e.g., for distance metrics). The categorical data metrics are selected based on Taha and Hanbury [19]. The continuous data metrics are inspired by other Python packages like scikit-image [20], scikit-learn [21], and SciPy [22]. Image reconstruction-specific metrics follow the fastMRI challenge [24]. The reader is referred to these references for metric descriptions, mathematical definitions, and guidelines on how to select appropriate metrics.

2.3. Platform and dependencies

pymia is implemented in Python (Python Software Foundation, Wilmington, DA, U.S.) and requires version 3.6 or higher. It depends on the following packages: h5py, NumPy, scikit-image, SciPy, and SimpleITK. To use the data package with a deep learning framework, either PyTorch or TensorFlow is required further. Unit tests are implemented using pytest. To build the documentation, Sphinx, Read the Docs Sphinx Theme, Sphinx-copybutton, and nbsphinx are required.

3. Results

pymia is hosted on the Python Package Index (PyPI) for easy installation of the latest version using the command pip install pymia. The code is publicly available on GitHub [https://github.com/rundherum/pymia/tree/master/examples/] for the command pip install pymia. The code is publicly available on GitHub [https://github.com/rundherum/pymia/tree/master/examples/] or directly rendered in the documentation [https://pymia.readthedocs.io/en/latest/examples.html]. In all examples, MR images of the head of four subjects from the Human Connectome Project (HCP) [37] are used. Each subject has four 3-D images (in the MetaImage and Nifty format) and demographic information provided as a text file. The images are a T1-weighted MR image, a T2-weighted MR image, a label image (ground truth), and a brain mask image. The demographic information is artificially created age, gender, and grade point average (GPA). The label images contain annotations of five brain structures (white matter, gray matter, hippocampus, amygdala, and thalamus), automatically segmented by FreeSurfer 5.3 [38, 39]. Therefore, the examples mimic the problem of medical image segmentation of brain tissues. The next sections shortly summarize the examples that cover dedicated functionalities of pymia. In addition, training example scripts for the segmentation of brain tissues using a U-Net in TensorFlow and PyTorch, including training with data augmentation, evaluation, and logging, can be found on GitHub.

https://github.com/rundherum/pymia
https://pymia.readthedocs.io/en/latest/
Table 2: Overview of the currently implemented metrics in pymia. Categorical metrics can be used for image segmentation and continuous metrics for image reconstruction and regression. The abbreviations are used for reporting and can be adapted upon instantiating the metrics. A reference is given where appropriate.

| Category       | Metric                          | Abbreviation | Remarks         |
|----------------|---------------------------------|--------------|-----------------|
| Categorical    | Dice coefficient                  | DICE         | -               |
|                | Jaccard coefficient               | JACRD        | -               |
|                | Sensitivity                       | SNSVETY      | -               |
|                | Specificity                       | SPCFTY       | -               |
|                | Fallout                           | FALLOUT      | -               |
|                | False negative rate               | FNR          | -               |
|                | Accuracy                          | ACURCY       | -               |
|                | Precision                         | PRCISON      | -               |
|                | True positive                     | TP           | -               |
|                | False positive                    | FP           | -               |
|                | True negative                     | TN           | -               |
|                | False negative                    | FN           | -               |
|                | F-measure                         | FMEASR\[\beta\] | definable |
|                | Global consistency error          | GCOERR       | -               |
|                | Volume similarity                 | VOLSMTY      | -               |
|                | Rand index                        | RNDIND       | -               |
|                | Adjusted rand index               | ADJRIND      | -               |
|                | Mutual information                | MUTINFO      | -               |
|                | Variation of information          | VARINFO      | -               |
|                | Interclass correlation            | ICCORR       | -               |
|                | Probabilistic distance            | PROBDST      | -               |
|                | Cohen Kappa coefficient           | KAPPA        | -               |
|                | Area under curve                  | AUC          | -               |
|                | Hausdorff distance                | HDRFDST\[\%\] | percentile definable |
|                | Average distance                  | AVGDIST      | -               |
|                | Mahalanobis distance              | MAHLNBS      | -               |
|                | Surface overlap                   | SURFOVLP     | -               |
|                | Surface Dice overlap              | SURFDICE     | -               |
|                | Area                              | AREA         | for reference or prediction, image slice definable |
|                | Volume                            | VOL          | for reference or prediction |

### 3.1. Data handling

The example Creation of a dataset illustrates how to create a HDF5 dataset. Figure 4 shows the structure of the dataset resulting from the example data. The root is separated into data and meta groups. The data group contains the concatenated T1- and T2-weighted MR images (images group), the label image (labels group), the brain mask (mask group), the concatenated age and GPA (numerical group), and the gender (gender group). Note that each group consists of four entries as the example data has four subjects. The dimension of the images group is $Z \times Y \times X \times C = 181 \times 217 \times 181 \times 2$, where $C$ represents the channel dimension, i.e., the concatenated T1- and T2-weighted MR images. The labels group and the mask group have the same dimensions, but $C = 1$. The numerical group is of dimension 2 (age and GPA) and the gender group of dimension 1. The meta group contains an entry with the subject identifiers (subjects), the file paths (files group), the physical image information like direction, origin, and spacing (info group), the file identifiers (names group), and shape information (shape group). The file identifiers in this example are T1, T2, GT, MASK, AGE, GPA, and GENDER. They allow to associate the dimensions in the data group with the data type, e.g., that the MR images are concatenated in the order T1- and T2-weighted and not the other way around.
Figure 4: Exemplary HDF5 dataset with four subjects. The dataset consists of image data (images, labels, and mask groups), numerical data (age and GPA), and the gender of the subjects. The dimension of the images group is $Z \times Y \times X \times C = 181 \times 217 \times 181 \times 2$, where $C = 2$ represents the channel dimension, i.e., the concatenated T1- and T2-weighted MR images. The labels and mask groups have the same dimensions, but $C = 1$. Alongside the data, meta-information is stored in the dataset. The open-source software HDFView 3.1.0 was used to open the dataset.

The example Data extraction and assembly illustrates how to use pymia in a typical deep learning loop over the data samples. More specifically, it shows the case where 2-D image slices are extracted from a dataset in order to feed it to a neural network before assembling the predictions back to 3-D images. It also covers extracting 3-D patches and loading the data directly from the file system instead from a dataset (use case described in Section 2.1.2).

Using pymia, we benchmarked the performance of different ways of data loading: i) loading from a HDF5 dataset, ii) loading compressed MetaImages, iii) loading uncompressed MetaImages, and iv) loading NumPy files. The latter three ways load the data directly from the file system (Section 2.1.2). We further compared three loading strategies: i) entire 3-D image, ii) 3-D patches of size $84 \times 84 \times 84$, and iii) 2-D image slices. An artificial dataset was created with $n = 25$ subjects, each with a T1- and T2-weighted MR image of the example data (size of $181 \times 217 \times 181$). The loading times for one sample (i.e., concatenated 3-D images, concatenated 3-D patches, and concatenated 2-D image slices) were averaged over five entire runs over the dataset. The mean and standard deviation of the loading times are shown in Fig. 5. Clearly, the HDF5 dataset is the fastest loading method independent of the loading variant, followed by NumPy, uncompressed MetaImage, and compressed MetaImage. For the latter three methods, the loading times are almost equal for each loading strategy because loading the entire 3-D image is always necessary even if only a 3-D patch or a 2-D image slice needs to be loaded.

3.2. Evaluation

The example Evaluation of results illustrates how to evaluate segmentation results. A written CSV file with the evaluation results is shown in Fig. 6.

The example Logging the training progress illustrates how to use the evaluation package to log the

---

8 Desktop computer with Ubuntu 18.04 LTS, 3.2 GHz Intel Core i7-3930K, 64 GB memory, Samsung EVO 850 500 GB SSD
Figure 5: Benchmark of the loading times of one sample for three loading variants and four methods. The bars represent the mean loading time ± the standard deviation.

Figure 6: CSV output of the evaluation example. Each line represents an evaluation result, here the Dice coefficient (DICE), 95th Hausdorff distance (HDFFDST95), and volume similarity (VOLSMTY) of a subject and label (e.g., gray matter of Subject 1).

Figure 6: Benchmark of the loading times of one sample for three loading variants and four methods. The bars represent the mean loading time ± the standard deviation.

| SUBJECT | LABEL      | DICE   | HDFFDST95 | VOLSMTY |
|---------|------------|--------|-----------|---------|
| 0       | Subject 1  | GREYWATER | 0.31373  | 0.165151 | 0.31373  |
| 1       | Subject 1  | THALAMUS       | 0.752252 | 2.000000 | 0.752252 |
| 2       | Subject 1  | WHITEMATTER   | 0.642021 | 6.768204 | 0.642021 |
| 3       | Subject 2  | GREYWATER     | 0.298358 | 10.862740 | 0.298358 |
| 4       | Subject 2  | THALAMUS      | 0.768488  | 2.000000  | 0.768488 |
| 5       | Subject 2  | WHITEMATTER   | 0.654239  | 6.000000  | 0.654239 |
| 6       | Subject 3  | GREYWATER     | 0.287460  | 6.717798  | 0.287460 |
| 7       | Subject 3  | THALAMUS      | 0.760978  | 2.000000  | 0.760978 |
| 8       | Subject 3  | WHITEMATTER   | 0.641251  | 6.164414  | 0.641251 |
| 9       | Subject 4  | GREYWATER     | 0.258504  | 8.660254  | 0.258504 |
| 10      | Subject 4  | THALAMUS      | 0.780754  | 2.000000  | 0.780754 |
| 11      | Subject 4  | WHITEMATTER   | 0.649203  | 6.000000  | 0.649203 |

4. Discussion

We developed pymia, a Python package for deep learning-based research in medical image analysis. pymia addresses flexible domain-specific data handling and evaluation, a gap of existing open-source projects, and especially current deep learning frameworks. The development emphasized independence to the deep learning frameworks, which allows for simplified adoptions of open-source methods (e.g., a novel model presented in a paper) independent of the framework without rewriting the entire data handling and evaluation. Therefore, fast prototyping is possible as many methods can easily be tested without the need to worry about the framework used.

The data package enables very flexible and fast access to medical data. The flexibility manifests in the simple change from, e.g., 2-D to 3-D; full- or patch-wise (Table 1). Even non-imaging data can easily be integrated. The modular design ensures flexibility of the data package, enabling extension and handling of custom data formats. Empirically, the data loading, relying on a HDF5 dataset, was measured to be faster than other common loading methods (Fig. 5). Therefore, the data package smoothly integrates into the framework-specific training routines of the current deep learning environment.

The evaluation package provides a simple way to evaluate predictions against references with a considerable amount of metrics for medical image analysis covering segmentation, reconstruction, and regression (Table 2). It can either be used standalone or in conjunction with a deep learning framework for performance monitoring (e.g., logging to the TensorBoard). Writers allow to save the evaluation results in the commonly used CSV format. The saved CSV files can easily be loaded into common statistical software for statistical analysis and visualization. For instance, it could also be used with the challengeR framework [40] for analyzing and visualizing the results of biomedical challenges.

pymia was successfully used for multiple research projects in medical image analysis, demonstrating its versatility. For medical image segmentation, pymia was applied to 2-D segmentation of peripheral nerves in thigh MR [41], 2-D segmentation of skin lesions [42], 2.5-D and slab-based segmentation of brain tumors [44] from MR images, and 2.5-D brain tumor resection cavity segmentation [45, 46, 47]. For image reconstruction, pymia was used for reconstruction of MR fingerprinting [48, 49, 50], demonstrating the handling of large 5-D tensors (350 × 350 × 5 × 175 × 2). In regression, pymia was applied to survival prediction of brain tumor patients in the 2017 BRATS chal-
lenge (2nd rank in the 2017 BRATS overall survival prediction challenge) and 2018 BRATS challenge where non-imaging data was used alongside MR images. Lastly, even 3-D point cloud data was handled by pymia for the refinement of peripheral nerve segmentation. Most of these publications have public code available and can serve as an additional point of reference complementing the pymia documentation. Due to the experience with these diverse projects, we consider the current state of the pymia package as stable and useful for deep learning-based research in medical image analysis. Indeed, pymia could also be applied in other domains such as video processing or industrial manufacturing. Future plans include mainly extending the examples, increasing code coverage by unit tests, and ensuring compatibility with future versions of the most used deep learning frameworks. With a growing user base, however, there will certainly emerge feature requests, but we aim at keeping simplicity and modularity in mind for future releases. For instance, it would be beyond the scope of the project to implement neural network architectures and loss functions as projects like MONAI and DeepNeuro do. However, stronger integration of projects like TorchIO and batchgenerators for data augmentation would certainly be interesting and valuable for the intended use of pymia.

In conclusion, pymia was developed to fill the gaps of existing deep learning frameworks with regards to medical image analysis. The data package facilitates the handling of medical data independent of the used deep learning framework. The evaluation package allows the evaluation of results using the prevalent metrics in medical imaging or performance monitoring during method development.

Conflict of interest statement

The authors declare no conflicts of interest.

Acknowledgement

The authors thank all the contributors to pymia and acknowledge the valuable feedback by Florian Koller. This research was partially supported by the Swiss National Science Foundation (SNSF) under the grant numbers 169607 and 184273, and the Swiss Foundation for Research on Muscle Diseases (ssm).

References

[1] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, C. I. Sánchez, A survey on deep learning in medical image analysis, Medical Image Analysis 42 (2017) 60–88. doi:10.1016/j.media.2017.07.005.
[2] D. Shen, G. Wu, H.-I. Suk, Deep Learning in Medical Image Analysis, Annual Review of Biomedical Engineering 19 (1) (2017) 221–248. doi:10.1146/annurev-bioeng-071516-044442.
[3] A. Maier, C. Syben, T. Lasser, C. Riess, A Gentle Introduction to Deep Learning in Medical Image Processing, Zeitschrift für Medizinische Physik 29 (2) (2018) 86–101. doi:10.1016/j.zemedi.2018.12.003.
[4] A. S. Lundervold, A. Lundervold, An overview of deep learning in medical imaging focusing on MRI, Zeitschrift für Medizinische Physik 29 (2) (2019) 102–127. doi:10.1016/j.zemedi.2018.11.002.
[5] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, 2012, pp. 1097–1105.
[6] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, in: N. Navab, J. Hornegger, W. M. Wells, A. F. Frangi (Eds.), Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015, Vol. 9351 of Lecture Notes in Computer Science, Springer, 2015, pp. 234–244. doi:10.1007/978-3-319-24574-4_28.
[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, J. Shlens, B. Steiner, I. Sutskever, P. Tucker, V. Vanhoucke, V. Vasudevan, O. Vinyals, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, arXiv preprint arXiv:1603.04467 1 (2015) 19.
[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, 2019, pp. 8024–8035.
[9] E. Gibson, W. Li, C. Sudre, L. Fidone, 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.
[10] 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, in: Medical Imaging Meets NIPS, 2017.
[38] B. Fischl, FreeSurfer, NeuroImage 62 (2) (2012) 774–781. doi:10.1016/j.neuroimage.2012.01.021

[39] B. Fischl, D. H. Salat, E. Busa, M. Albert, M. Dieterich, C. Hasegawa, A. Van Der Kouwe, R. Kiliany, D. Kennedy, S. Klaveness, A. Montillo, N. Makris, B. Rosen, A. M. Dale, Whole brain segmentation: Automated labeling of neuroanatomical structures in the human brain, NeuroImage 33 (3) (2002) 341–355. doi:10.1016/j.neuroimage.2006.02.059

[40] M. Wiesendanger, A. Reineke, B. A. Landman, M. J. Cardoso, L. Maier-Hein, A. Kopp-Schneider, Methods and open-source toolkit for analyzing and visualizing challenge results, arXiv preprint arXiv:1910.05121.

[41] F. Balsiger, C. Steindel, M. Arn, B. Wagner, L. Gruninger, B. Fischl, D. H. Salat, E. Busa, M. Albert, M. Dieterich, C. Hasegawa, A. Van Der Kouwe, R. Kiliany, D. Kennedy, S. Klaveness, A. Montillo, N. Makris, B. Rosen, A. M. Dale, Whole brain segmentation: Automated labeling of neuroanatomical structures in the human brain, NeuroImage 33 (3) (2002) 341–355. doi:10.1016/j.neuroimage.2006.02.059

[42] A. Jungo, M. Reyes, Assessing Reliability and Challenges of Uncertainty Estimations for Medical Image Segmentation, in: D. Shen, T. Liu, T. M. Peters, L. H. Stalb, C. Essert, S. Zhou, P.-T. Yap, A. Khan (Eds.), Medical Image Computing and Computer Assisted Intervention - MICCAI 2019, Vol. 11765 of Lecture Notes in Computer Science, Springer, Cham, 2019, pp. 48–56. doi:10.1007/978-3-030-32245-8_6

[43] A. Jungo, R. McKinley, R. Meier, U. Knecht, L. Vera, J. Pérez-Beteta, D. Molina-García, V. M. Pérez-García, R. Wiest, M. Reyes, Towards uncertainty-assisted brain tumor segmentation and survival prediction, in: A. Crimi, S. Bakas, H. Kuijf, B. Menze, M. Reyes (Eds.), Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, BrainLes 2017, Vol. 10670 of Lecture Notes in Computer Science, Springer, 2018, pp. 474–485. doi:10.1007/978-3-319-75238-9_40

[44] A. Jungo, F. Balsiger, M. Reyes, Analyzing the Quality and Challenges of Uncertainty Estimations for Brain Tumor Segmentation, Frontiers in Neuroscience 14 (2020) 282. doi:10.3389/fnins.2020.00282

[45] A. Jungo, R. Meier, E. Ermis, M. Blatti-Moreno, E. Herrmann, R. Wiest, M. Reyes, On the effect of inter-observer variability for a reliable estimation of uncertainty of medical image segmentation, in: Medical Image Computing and Computer Assisted Intervention - MICCAI 2018, Vol. 11070 of Lecture Notes in Computer Science, Springer, Cham, 2018, pp. 682–690. doi:10.1007/978-3-030-32245-8_17

[46] A. Jungo, R. Meier, E. Ermis, E. Herrmann, M. Reyes, Uncertainty-driven Sanity Check: Application to Postoperative Brain Tumor Cavity Segmentation, in: Medical Imaging with Deep Learning - MIDL 2018, 2018.

[47] E. Ermis, A. Jungo, R. Poel, M. Blatti-Moreno, R. Meier, U. Knecht, D. M. Aebersold, M. K. Fix, P. Manser, M. Reyes, E. Herrmann, Fully automated brain resection cavity delineation for radiation target volume definition in glioblastoma patients using deep learning, Radiation Oncology 15 (1). doi:10.1186/s13014-020-01553-x

[48] F. Balsiger, A. Shridhar Konar, S. Chilop, V. Chandran, O. Scheidegger, S. Geethanath, M. Reyes, Magnetic Resonance Fingerprinting Reconstruction via Spatiotemporal Convolutional Neural Networks, in: F. Knoll, A. Maier, D. Rueckert (Eds.), Machine Learning for Medical Image Reconstruction, Vol. 11074 of Lecture Notes in Computer Science, Springer, Cham, 2018, pp. 39–46. doi:10.1007/978-3-030-00129-2_5

[49] F. Balsiger, O. Scheidegger, P. G. Carlier, B. Marty, M. Reyes, On the Spatial and Temporal Influence for the Reconstruction of Magnetic Resonance Fingerprinting, in: M. J. Cardoso, A. Feragen, B. Glocker, E. Konukoglu, I. Ozug, G. Unal, T. Vercauteren (Eds.), Proceedings of The 2nd International Conference on Medical Imaging with Deep Learning, Vol. 102 of Proceedings of Machine Learning Research, PMLR, London, 2019, pp. 27–38.

[50] F. Balsiger, A. Jungo, O. Scheidegger, P. G. Carlier, M. Reyes, B. Marty, Spatially Regularized Parametric Map Reconstruction for Fast Magnetic Resonance Finger-printing, Medical Image Analysis 64 (2020) 101741. doi:10.1016/j.media.2020.101741

[51] Y. Suter, A. Jungo, M. Rebsamen, U. Knecht, E. Herrmann, R. Wiest, M. Reyes, Deep learning versus classical regression for brain tumor patient survival prediction, in: A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, T. V. Walsum (Eds.), Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. BrainLes 2018, Vol. 11384 of Lecture Notes in Computer Science, Springer, Cham, 2019, pp. 429–440. doi:10.1007/978-3-030-11726-9_38

[52] F. Balsiger, Y. Soom, O. Scheidegger, M. Reyes, Learning Shape Representation on Sparse Point Clouds for Volumetric Image Segmentation, in: D. Shen, T. Liu, T. M. Peters, L. H. Stalb, C. Essert, S. Zhou, P.-T. Yap, A. Khan (Eds.), Medical Image Computing and Computer Assisted Intervention - MICCAI 2019, Vol. 11765 of Lecture Notes in Computer Science, Springer, Cham, 2019, pp. 273–281. doi:10.1007/978-3-030-32245-8_31

[53] F. Isensee, P. Jäger, J. Wasserthal, D. Zimmerer, J. Peterson, S. Kohl, J. Schock, A. Klein, T. Roß, S. Wirkert, P. Neher, S. Dinkelacker, G. Köhler, K. Maier-Hein, batchgenerators - a python framework for data augmentation (2020). doi:10.5281/ZENODO.3632567