TorchIO: A Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning

Fernando Pérez-García [a,b,c,*], Rachel Sparks [c], Sébastien Ourselin [c]

[a] Department of Medical Physics and Biomedical Engineering, University College London, UK
[b] Wellcome / EPSRC Centre for Interventional and Surgical Sciences (WEISS), University College London, UK
[c] School of Biomedical Engineering & Imaging Sciences (BMEIS), King's College London, UK

ARTICLE INFO

ABSTRACT

Background and objective: Processing of medical images such as MRI or CT presents different challenges compared to RGB images typically used in computer vision. These include a lack of labels for large datasets, high computational costs, and the need of metadata to describe the physical properties of voxels. Data augmentation is used to artificially increase the size of the training datasets. Training with image subvolumes or patches decreases the need for computational power. Spatial metadata needs to be carefully taken into account in order to ensure a correct alignment and orientation of volumes.

Methods: We present TorchIO, an open-source Python library to enable efficient loading, preprocessing, augmentation and patch-based sampling of medical images for deep learning. TorchIO follows the style of PyTorch and integrates standard medical image processing libraries to efficiently process images during training of neural networks. TorchIO transforms can be easily composed, reproduced, traced and extended. Most transforms can be inverted, making the library suitable for test-time augmentation and estimation of aleatoric uncertainty in the context of segmentation. We provide multiple generic preprocessing and augmentation operations as well as simulation of MRI-specific artifacts.

Results: Source code, comprehensive tutorials and extensive documentation for TorchIO can be found at http://torchio.rtfd.io/. The package can be installed from the Python Package Index (PyPI) running `pip install torchio`. It includes a command-line interface which allows users to apply transforms to image files without using Python. Additionally, we provide a graphical user interface within a TorchIO extension in 3D Slicer to visualize the effects of transforms.

Conclusion: TorchIO was developed to help researchers standardize medical image processing pipelines and allow them to focus on the deep learning experiments. It encourages good open-science practices, as it supports experiment reproducibility and is version-controlled so that the software can be cited precisely. Due to its modularity, the library is compatible with other frameworks for deep learning with medical images.

1. Introduction

Recently, deep learning has become a ubiquitous research approach for solving image understanding and analysis problems. Convolutional neural networks (CNNs) have become the state of the art for many medical imaging tasks including segmentation [1], classification [2], reconstruction [3] and registration [4]. Many of the network architectures and techniques have been adopted from computer vision.

Compared to 2D red-green-blue (RGB) images typically used in computer vision, processing of medical images such as MRI, ultrasound (US) or CT presents different challenges. These include a lack of labels for large datasets, high computational costs (as the data is typically volumetric), and the use of metadata to describe the physical size and position of voxels.

Open-source frameworks for training CNNs with medical images have been built on top of TensorFlow [5–7]. Recently, the pop-

---

* Corresponding author.
  E-mail address: fernando.perezgarcia.17@ucl.ac.uk (F. Pérez-García).

ularity of PyTorch [8] has increased among researchers due to its improved usability compared to TensorFlow [9], driving the need for open-source tools compatible with PyTorch. To reduce duplication of effort among research groups, improve experimental reproducibility and encourage open-science practices, we have developed TorchIO: an open-source Python library for efficient loading, preprocessing, augmentation, and patch-based sampling of medical images designed to be integrated into deep learning workflows.

TorchIO is a compact and modular library that can be seamlessly used alongside higher-level deep learning frameworks for medical imaging, such as the Medical Open Network for AI (MONAI). It removes the need for researchers to code their own preprocessing pipelines from scratch, which might be error-prone due to the complexity of medical image representations. Instead, it allows researchers to focus on their experiments, supporting experiment reproducibility and traceability of their work, and standardization of the methods used to process medical images for deep learning.

1.1. Motivation

The nature of medical images makes it difficult to rely on a typical computer-vision pipeline for neural network training. In Section 1.1.1, we describe challenges related to medical images that need to be overcome when designing deep learning workflows. In Section 1.1.2, we justify the choice of PyTorch as the main deep learning framework dependency of TorchIO.

1.1.1. Challenges in medical image processing for deep learning

In practice, multiple challenges must be addressed when developing deep learning algorithms for medical images: 1) handling metadata related to physical position and size, 2) lack of large labeled datasets, 3) high computational costs due to data multidimensionality and 4) lack of consensus for best normalization practices. These challenges are very common in medical imaging and require certain features that may not be implemented in more general-purpose image processing frameworks such as Albumentations [10] or TorchVision [8].

Metadata. In computer vision, picture elements, or *pixels*, which are assumed to be square, have a spatial relationship that comprises proximity and depth according to both the arrangement of objects in the scene and camera placement. In comparison, medical images are reconstructed such that the location of volume elements, or cuboid-shaped *voxels*, encodes a meaningful 3D spatial relationship. In simple terms, for 2D natural images, pixel vicinity does not necessarily indicate spatial correspondence, while for medical images spatial correspondence between nearby voxels can often be assumed.

Metadata, which encodes the physical size, spacing, and orientation of voxels, determines spatial relationships between voxels [11]. This information can provide meaningful context when performing medical image processing, and is often implicitly or explicitly used in medical imaging software. Furthermore, metadata is often used to determine correspondence between images as well as voxels within an image. For example, registration algorithms for medical images typically work with physical coordinates rather than voxel indices.

Fig. 1 shows the superposition of an MRI and a corresponding brain parcellation [12] with the same size (181×181) but different origin, spacing and orientation. A native user would assume that, given that the superimposition looks correct and both images have the same size, they are ready for training. However, the visualization is correct only because 3D Slicer [13], the software used for visualization, is aware of the spatial metadata of the images. As CNNs generally do not take spatial metadata into account, training using these images without preprocessing would lead to poor results.

Medical images are typically stored in specialized formats such as Data Imaging and Communications in Medicine (DICOM) or Neuroimaging Informatics Technology Initiative (NIfTI) [11], and commonly read and processed by medical imaging frameworks such as SimpleITK [14] or NiBabel [15].

Limited training data. Deep learning methods typically require large amounts of annotated data, which are often scarce in clinical scenarios due to concerns over patient privacy, the financial and time burden associated with collecting data as part of a clinical trial, and the need for annotations from highly-trained and experienced raters. Data augmentation techniques can be used to increase the size of the training dataset artificially by applying different transformations to each training instance while preserving the relationship to annotations.

Data augmentation performed in computer vision typically aims to simulate variations in camera properties, field of view (FOV), or perspective. Traditional data augmentation operations applied in computer vision include geometrical transforms such as random rotation or zoom, color-space transforms such as random channel swapping or kernel filtering such as random Gaussian blurring. Data augmentation is usually performed on the fly, i.e., every time an image is loaded from disk during training.

Several computer vision libraries supporting data augmentation have appeared recently, such as Albumentations [10], or `imgaug` [16]. PyTorch also includes some computer vision transforms, mostly implemented as Pillow wrappers [17]. However, none of these libraries support reading or transformations for 3D images. Furthermore, medical images are almost always grayscale, therefore color-space transforms are not applicable. Additionally, cropping and scaling are more challenging to apply to medical images without affecting the spatial relationships of the data. Metadata should usually be considered when applying these transformations to medical images.

In medical imaging, the purpose of data augmentation is designed to simulate anatomical variations and scanner artifacts. Anatomical variation and sample position can be simulated using spatial transforms such as elastic deformation, lateral flipping, or affine transformations. Some artifacts are unique to specific medical image modalities. For example, ghosting artifacts will be present in MRI if the patient moves during acquisition, and metallic implants often produce streak artifacts in CT. Simulation of these artifacts can be useful when performing augmentation on medical images.

Computational costs. The number of pixels in 2D images used in deep learning is rarely larger than one million. For example, the input size of several popular image classification models is $224 \times 224 \times 3 = 150\,528$ pixels (588 KiB if 32 bits per pixel are used). In contrast, 3D medical images often contain hundreds of millions of voxels, and downsampling might not be acceptable when small details should be preserved. For example, the size of a high-resolution lung CT-scan used for quantifying chronic obstructive pulmonary disease (COPD) damage in a research setting, with spacing $0.66 \times 0.66 \times 0.30$ mm, is $512 \times 512 \times 1069 = 280\,231\,936$ voxels (1.04 GiB if 32 bits per voxel are used).

In computer vision applications, images used for training are grouped in batches whose size is often in the order of hundreds [18] or even thousands [19] of training instances, depending on the available graphics processing unit (GPU) memory. In medical image applications, batches rarely contain more than one [1] or two [20] training instances due to their larger memory footprint compared to natural images. This reduces the utility of techniques such as batch normalization, which rely on batches being large enough to estimate dataset variance appropriately [21]. Moreover,
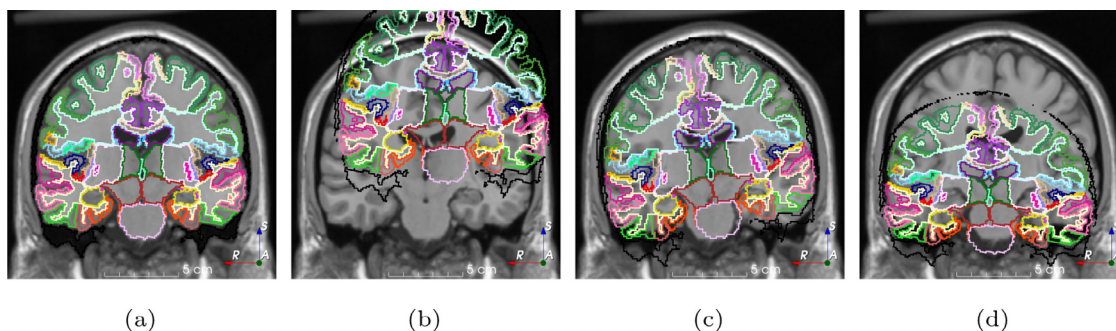
Fig. 1. Demonstration of the importance of spatial metadata in medical image processing. The size of both the MRI and the segmentation is 181×181. When spatial metadata is taken into account (a), images are correctly superimposed (only the borders of each region are shown for clarity purposes). Images are incorrectly superimposed if (b) origin, (c) orientation or (d) spacing are ignored.

large image size and small batches result in longer training time, hindering the experimental cycle that is necessary for hyperparameter optimization. In cases where GPU memory is limited and the network architecture is large, it is possible that not even the entirety of a single volume can be processed during a training iteration. To overcome this challenge, it is common in medical imaging to train using subsets of the image, or image *patches*, randomly extracted from the volumes.

Networks can be trained with 2D slices extracted from 3D volumes, aggregating the inference results to generate a 3D volume [22]. This can be seen as a specific case of patch-based training, where the size of the patches along a dimension is one. Other methods extract volumetric patches for training, that are often cubes, if the voxel spacing is isotropic [23], or cuboids adapted to the anisotropic spacing of the training images [24].

Transfer learning and normalization. One can pre-train a network on a large dataset of natural images such as ImageNet [25], which contains more than 14 million labeled images, and fine-tune on a custom, much smaller target dataset. This is a typical use of transfer learning in computer vision [26]. The literature has reported mixed results using transfer learning to apply models pretrained on natural images to medical images [27,28].

In computer vision, best practice is to normalize each training instance before training, using statistics computed from the whole training dataset [18]. Preprocessing of medical images is often performed on a per-image basis, and best practice is to take into account the bimodal nature of medical images (i.e., that an image has a background and a foreground).

Medical image voxel intensity values can be encoded with different data types and intensity ranges, and the meaning of a specific value can vary between different modalities, sequence acquisitions, or scanners. Therefore, intensity normalization methods for medical images often involve more complex parameterization of intensities than those used for natural images [29].

1.1.2. Deep learning frameworks. There are currently two major generic deep learning frameworks: TensorFlow [5] and PyTorch [8], primarily maintained by Google and Facebook, respectively. Although TensorFlow has traditionally been the primary choice for both research and industry, PyTorch has recently seen a substantial increase in popularity, especially among the research community [9].

PyTorch is often preferred by the research community as it is *pythonic*, i.e., its design, usage, and application programming interfaceAPI follow the conventions of plain Python. Moreover, the API for tensor operations follows a similar paradigm to the one for NumPy multidimensional arrays, which is the primary array programming library for the Python language [30]. In contrast, for TensorFlow, researchers need to become familiar with new design

elements such as sessions, placeholders, feed dictionaries, gradient tapes and static graphs. In PyTorch, objects are standard Python classes and variables, and a dynamic graph makes debugging intuitive and familiar to anyone already using Python. These differences have decreased with the recent release of TensorFlow 2, whose eager mode makes usage reminiscent of Python.

TorchIO was designed to be in the style of PyTorch and uses several of its tools to reduce the barrier to learning how to use TorchIO for those researchers already familiar with PyTorch.

1.2. Related work

NiftyNet [7] and the Deep Learning Toolkit (DLTK) [6] are deep learning frameworks designed explicitly for medical image processing using the TensorFlow 1 platform. Both of them are no longer being actively maintained. They provide implementations of some popular network architectures such as U-Net [1], and can be used to train 3D CNNs for different tasks. For example, NiftyNet was used to train a 3D residual network for brain parcellation [23], and DLTK was used to perform multi-organ segmentation on CT and MRI [31].

The `medicaltorch` library [32] closely follows the PyTorch design, and provides some functionalities for preprocessing, augmentation and training of medical images. However, it does not leverage the power of specialized medical image processing libraries, such as SimpleITK [14], to process volumetric images.

Similar to DLTK, this library has not seen much activity since 2018.

The `batchgenerators` library [33], used within the popular medical segmentation framework nn-UNet [34], includes custom dataset and data loader classes for multithreaded loading of 3D medical images, implemented before data loaders were available in PyTorch. In the usage examples from GitHub, preprocessing is applied to the whole dataset before training. Then, spatial data augmentation is performed at the volume level, from which one patch is extracted and intensity augmentation is performed at the patch level. In this approach, only one patch is extracted per volume, diminishing the efficiency of training pipelines. Transforms in `batchgenerators` are mostly implemented using NumPy [30] and SciPy [35].

More recently, a few PyTorch-based libraries for deep learning and medical images have appeared. There are two other libraries, developed in parallel to TorchIO, focused on data preprocessing and augmentation. Rising[1] is a library for data augmentation entirely written in PyTorch, which allows for gradients to be propagated through the transformations and perform all computations on the GPU. However, this means specialized medical imaging libraries such as SimpleITK cannot be used. `pymia` [36] provides

---
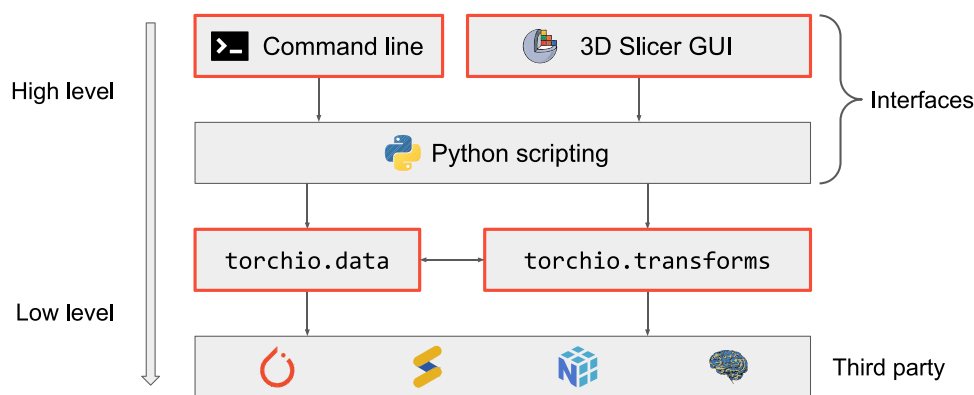
[1] https://github.com/PhoenixDL/rising.

Fig. 2. General diagram of TorchIO, its dependencies and its interfaces. Boxes with a red border (———) represent elements implemented in TorchIO. Logos indicate lower-level Python libraries used by TorchIO. ●: NiBabel [15]; ⚡: SimpleITK [14]; ▦: NumPy [30]; ○: PyTorch [8].

features for data handling (loading, preprocessing, sampling) and evaluation. It is compatible with TorchIO transforms, which are typically leveraged for data augmentation, as their data handling is more focused on preprocessing. pymia can be easily integrated into either PyTorch or TensorFlow pipelines. It was recently used to assess the suitability of evaluation metrics for medical image segmentation [37].

MONAI [38] and Eisen [39] are PyTorch-based frameworks for deep learning workflows with medical images. Similar to NiftyNet and DLTK, they include implementation of network architectures, transforms, and higher-level features to perform training and inference. For example, MONAI was recently used for brain segmentation on fetal MRI [40]. As these packages are solving a large problem, i.e., that of workflow in deep learning for medical images, they do not contain all of the data augmentation transforms present in TorchIO. However, it is important to note that an end user does not need to select only one open-source package, as TorchIO transforms are compatible with both Eisen and MONAI.

TorchIO is a library that specializes in preprocessing and augmentation using PyTorch, focusing on ease of use for researchers. This is achieved by providing a PyTorch-like API, comprehensive documentation with many usage examples, and tutorials showcasing different features, and by actively addressing feature requests and bug reports from the many users that have already adopted TorchIO. This is in contrast with other modern libraries released after TorchIO such as MONAI, which aims to deliver a larger umbrella of functionalities including federated learning or active learning, but may have slower development and deployment.

2. Methods

We developed TorchIO, a Python library that focuses on data loading and augmentation of medical images in the context of deep learning.

TorchIO is a unified library to load and augment data that makes explicit use of medical image properties, and is flexible enough to be used for different loading workflows. It can accelerate research by avoiding the need to code a processing pipeline for medical images from scratch.

In contrast with Eisen or MONAI, we do not implement network architectures, loss functions or training workflows. This is to limit the scope of the library and to enforce modularity between training of neural networks and preprocessing and data augmentation.

Following the PyTorch philosophy [8], we designed TorchIO with an emphasis on simplicity and usability while reusing PyTorch classes and infrastructure where possible. Note that, although we designed TorchIO following PyTorch style, the library

could also be used with other deep learning platforms such as TensorFlow or Keras [41].

TorchIO makes use of open-source medical imaging software platforms. Packages were selected to reduce the number of required external dependencies and the need to re-implement basic medical imaging processing operations (image loading, resampling, etc.).

TorchIO features are divided into two categories: data structures and input/output (torchio.data), and transforms for preprocessing and augmentation (torchio.transforms). Fig. 2 represents a diagram of the codebase and the different interfaces to the library.

2.1. Data

2.1.1. Input/Output

TorchIO uses the medical imaging libraries NiBabel and SimpleITK to read and write images. Dependency on both is necessary to ensure broad support of image formats. For instance, NiBabel does not support reading Portable Network Graphics (PNG) files, while SimpleITK does not support some neuroimaging-specific formats.

TorchIO supports up to 4D images, i.e., 2D or 3D single-channel or multi-channel data such as X-rays, RGB histological slides, microscopy stacks, multispectral images, CT-scans, functional MRI (fMRI) and diffusion MRI (dMRI).

2.1.2. Data structures

Image. The Image class, representing one medical image, stores a 4D tensor, whose voxels encode, e.g., signal intensity or segmentation labels, and the corresponding affine transform, typically a rigid (Euclidean) transform, to convert voxel indices to world coordinates in millimeters. Arbitrary fields such as acquisition parameters may also be stored.

Subclasses are used to indicate specific types of images, such as ScalarImage and LabelMap, which are used to store, e.g., CT scans and segmentations, respectively.

An instance of Image can be created using a filepath, a PyTorch tensor, or a NumPy array. This class uses lazy loading, i.e., the data is not loaded from disk at instantiation time. Instead, the data is only loaded when needed for an operation (e.g., if a transform is applied to the image).

Fig. 3 shows two instances of Image. The instance of ScalarImage contains a 4D tensor representing a dMRI, which contains four 3D volumes (one per gradient direction), and the associated affine matrix. Additionally, it stores the strength and direction for each of the four gradients. The instance of LabelMap
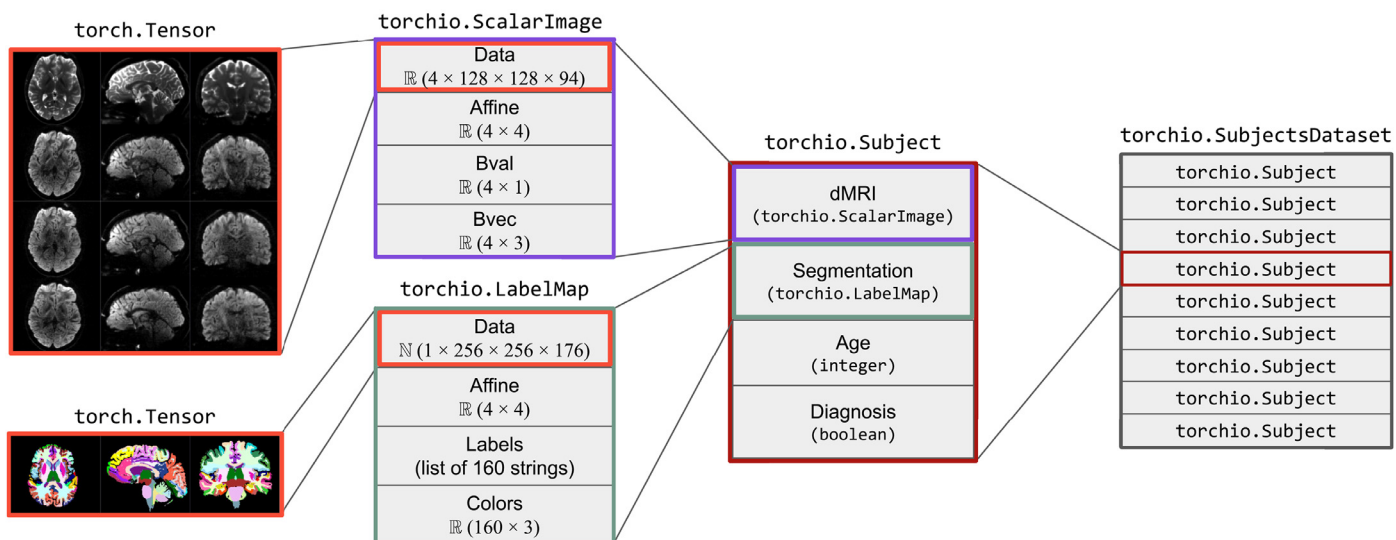
Fig. 3. Usage example of `ScalarImage`, `LabelMap`, `Subject` and `SubjectsDataset`. The images store a 4D dMRI and a brain parcellation, and other related metadata.

contains a brain parcellation of the same subject, the associated affine matrix, and the name and color of each brain structure.

Subject. The `Subject` class stores instances of `Image` associated to a subject, e.g., a human or a mouse. As in the `Image` class, `Subject` can store arbitrary fields such as age, diagnosis or ethnicity.

Subjects dataset. The `SubjectsDataset` inherits from the PyTorch `Dataset`. It contains the list of subjects and optionally a transform to be applied to each subject after loading. When `SubjectsDataset` is queried for a specific subject, the corresponding set of images are loaded, a transform is applied to the images and the instance of `Subject` is returned.

For parallel loading, a PyTorch `DataLoader` may be used. This loader spawns multiple processes, each of which contains a shallow copy of the `SubjectsDataset`. Each copy is queried for a different subject, therefore loading and transforming is applied to different subjects in parallel on the central processing unit (CPU) (Fig. 4a).

An example of subclassing `SubjectsDataset` is `torchio.datasets.IXI`, which may be used to download the Information eXtraction from Images (IXI) dataset.[2]

2.1.3. Patch-based training

Memory limitations often require training and inference steps to be performed using image subvolumes or *patches* instead of the whole volumes, as explained in Section 1.1.1.3. In this section, we describe how TorchIO implements patch-based training via image sampling and queueing.

Samplers. A sampler takes as input an instance of `Subject` and returns a version of it whose images have a reduced FOV, i.e., the new images are subvolumes, also called windows or *patches*. For this, a `PatchSampler` may be used.

Different criteria may be used to select the center voxel of each output patch. A `UniformSampler` selects a voxel as the center at random with all voxels having an equal probability of being selected. A `WeightedSampler` selects the patch center according to a probability distribution image defined over all voxels, which is passed as input to the sampler.

At testing time, images are sampled such that a dense inference can be performed on the input volume. A `GridSampler` can be used to sample patches such that the center voxel is selected using a set stride. In this way, sampling over the entire volume is ensured. The potentially-overlapping inferred patches can be passed to a `GridAggregator` that builds the resulting volume patch by patch (or batch by batch).

Queue. A training iteration (i.e., forward and backward pass) performed on a GPU is usually faster than loading, preprocessing, augmenting, and cropping a volume on a CPU. Most preprocessing operations could be performed using a GPU, but these devices are typically reserved for training the CNN so that the batch size and input tensor can be as large as possible. Therefore, it is beneficial to prepare (i.e., load, preprocess and augment) the volumes using multiprocessing CPU techniques in parallel with the forward-backward passes of a training iteration.

Once a volume is appropriately prepared, it is computationally beneficial to sample multiple patches from a volume rather than having to prepare the same volume each time a patch needs to be extracted. The sampled patches are then stored in a buffer or *queue* until the next training iteration, at which point they are loaded onto the GPU to perform an optimization iteration. For this, TorchIO provides the `Queue` class, which inherits from the PyTorch `Dataset` (Fig. 4b). In this queueing system, samplers behave as generators that yield patches from volumes contained in the `SubjectsDataset`.

The end of a training epoch is defined as the moment after which patches from all subjects have been used for training. At the beginning of each training epoch, the subjects list in the `SubjectsDataset` is shuffled, as is typically done in machine learning pipelines to increase variance of training instances during model optimization. A PyTorch loader begins by shallow-copying the dataset to each subprocess. Each worker subprocess loads and applies image transforms to the volumes in parallel. A patches list is filled with patches extracted by the sampler, and the queue is shuffled once it has reached a specified maximum length so that batches are composed of patches from different subjects. The internal data loader continues querying the `SubjectsDataset` using multiprocessing. The patches list, when emptied, is refilled with new patches. A second data loader, external to the queue, may be used to collate batches of patches stored in the queue, which are passed to the neural network.

---

[2] https://brain-development.org/ixi-dataset/.
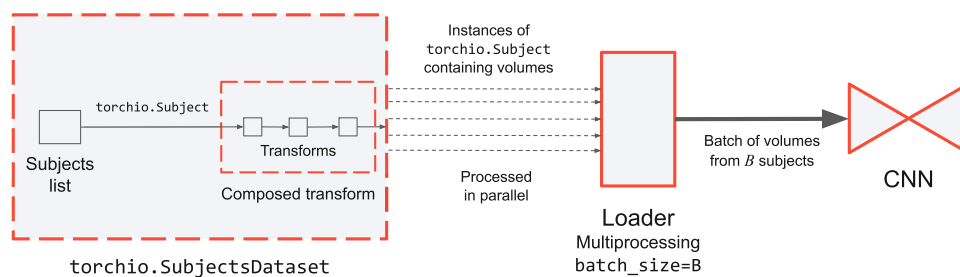
(a) Training with whole volumes
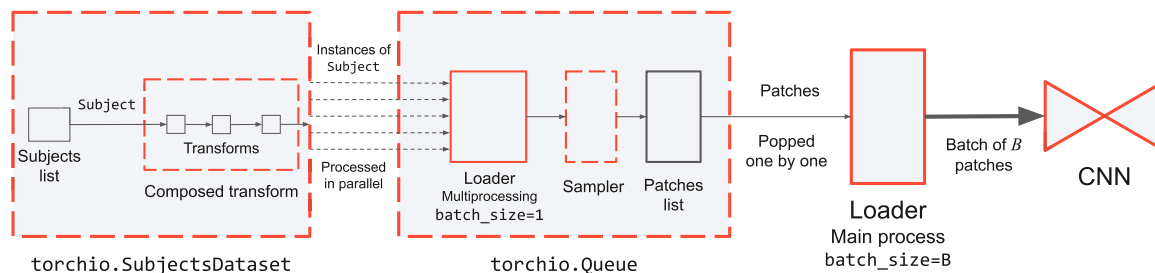
(b) Training with patches

Fig. 4. Diagram of data pipelines for training with whole volumes (top) and patches (bottom). Boxes with a red border represent PyTorch classes (——) or TorchIO classes that inherit from PyTorch classes (– – –).

2.2. Transforms

The transforms API was designed to be similar to the PyTorch `torchvision.transforms` module. TorchIO includes augmentations such as random affine transformation (Fig. 5e) or random blur (Fig. 5b), but they are implemented using medical imaging libraries [14,15] to take into account specific properties of medical images, namely their size, resolution, location, and orientation (see Section 1.1.1.1). Table 1 shows transforms implemented in TorchIO v0.18.0 and their main corresponding library dependencies.

Transforms are designed to be flexible regarding input and output types. Following a duck typing approach, they can take as input PyTorch tensors, SimpleITK images, NumPy arrays, Pillow images, Python dictionaries, and instances of `Subject` and `Image`, and will return an output of the same type.

TorchIO transforms can be classified into either spatial and intensity transforms, or preprocessing and augmentation transforms (Table 1). All are subclasses of the `Transform` base class. Spatial transforms and intensity transforms are related to the `SpatialTransform` and `IntensityTransform` classes, respectively. Transforms whose parameters are randomly chosen are subclasses of `RandomTransform`.

Instances of `SpatialTransform` typically modify the image bounds or spacing, and often need to resample the image using interpolation. They are applied to all image types. Instances of `IntensityTransform` do not modify the position of voxels, only their values, and they are only applied to instances of `ScalarImage`. For example, if a `RandomNoise` transform (which is a subclass of `IntensityTransform`) receives as input a `Subject` with a `ScalarImage` representing a CT scan and a `LabelMap` representing a segmentation, it will add noise to only the CT scan. On the other hand, if a `RandomAffine` transform (which is a subclass of `SpatialTransform`) receives the same input, the same affine transformation will be applied to both im-

ages, with nearest-neighbor interpolation always used to interpolate `LabelMap` objects.

2.2.1. Preprocessing

Preprocessing transforms are necessary to ensure spatial and intensity uniformity of training instances.

Spatial preprocessing is important as CNNs do not generally take into account metadata related to medical images (see Section 1.1.1.1), therefore it is necessary to ensure that voxels across images have similar spatial location and relationships before training. Spatial preprocessing transforms typically used in medical imaging include resampling (e.g., to make voxel spacing isotropic for all training samples) and reorientation (e.g., to orient all training samples in the same way). For example, the `Resample` transform can be used to fix the issue presented in Fig. 1.

Intensity normalization is generally beneficial for optimization of neural networks. TorchIO provides intensity normalization techniques including min-max scaling or standardization,[3] which are computed using pure PyTorch. A binary image, such as a mask representing the foreground or structures of interest, can be used to define the set of voxels to be taken into account when computing statistics for intensity normalization. We also provide a method for MRI histogram standardization [48], computed using NumPy, which may be used to overcome the differences in intensity distributions between images acquired using different scanners or sequences.

2.2.2. Augmentation

TorchIO includes spatial augmentation transforms such as random flipping using PyTorch and random affine and elastic deformation transforms using SimpleITK. Intensity augmentation trans-

---

[3] In this context, standardization refers to correcting voxel intensity values to have zero mean and unit variance.
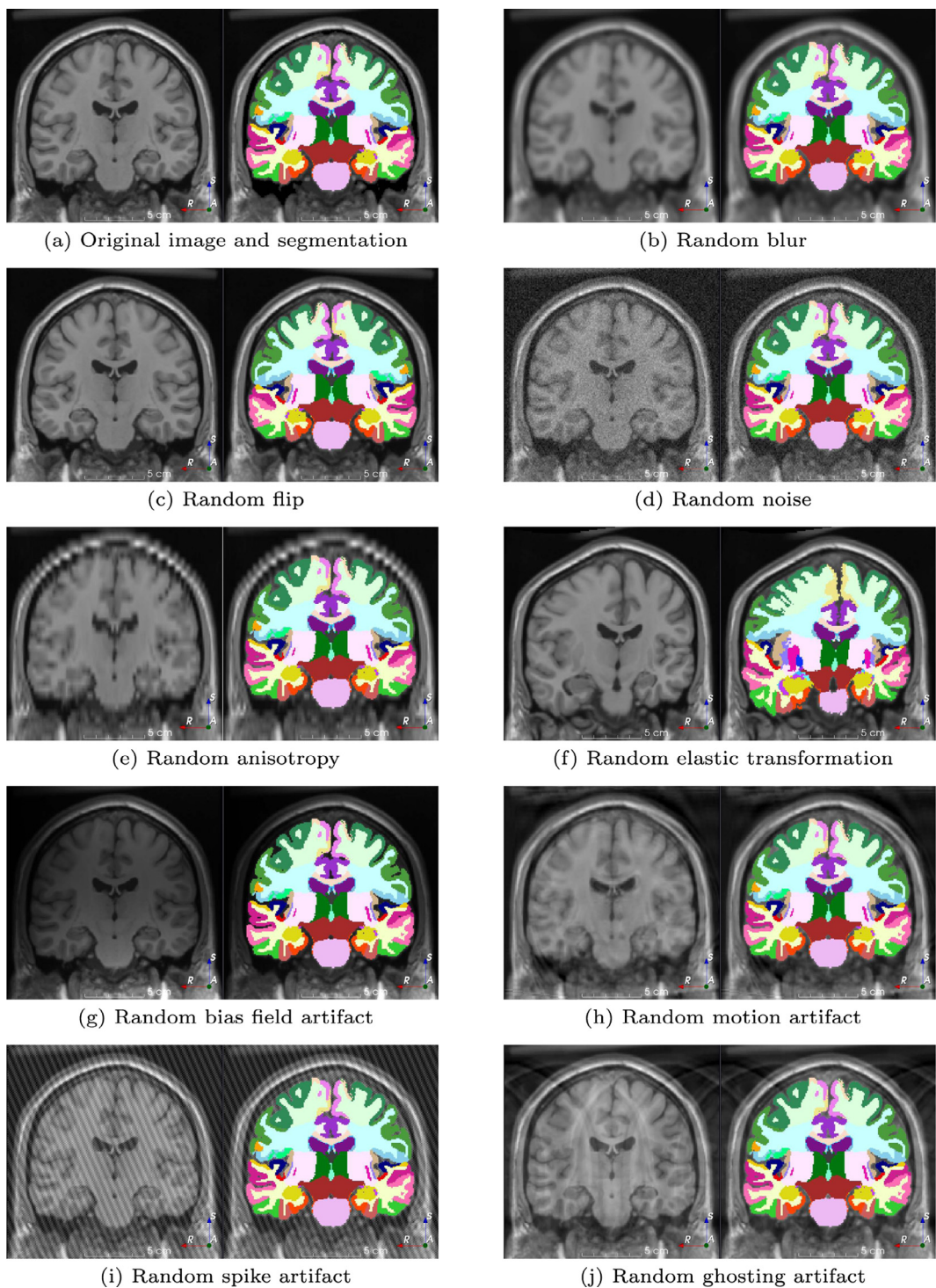
(a) Original image and segmentation

(b) Random blur

(c) Random flip

(d) Random noise

(e) Random anisotropy

(f) Random elastic transformation

(g) Random bias field artifact

(h) Random motion artifact

(i) Random spike artifact

(j) Random ghosting artifact

Fig. 5. A selection of data augmentation techniques available in TorchIO `v0.18.0`. Each example is presented as a pair of images composed of the transformed image and a corresponding transformed label map. Note that all screenshots are from a 2D coronal slice of the transformed 3D images. The MRI corresponds to the Montreal Neurological Institute (MNI) Colin 27 average brain [49], which can be downloaded using `torchio.datasets.Colin27`. Label maps were generated using an automated brain parcellation algorithm [12].

forms include random Gaussian blur using a SimpleITK filter (Fig. 5b) and addition of random Gaussian noise using pure PyTorch (Fig. 5d). All augmentation transforms are subclasses of `RandomTransform`.

Although current domain-specific data augmentation transforms available in TorchIO are mostly related to MRI, we encourage users to contribute physics-based data augmentation techniques for US or CT [50].

We provide several MRI-specific augmentation transforms related to k-space, which are described below. An MR image is usually reconstructed as the magnitude of the inverse Fourier transform of the k-space signal, which is populated with the signals generated by the sample as a response to a radio-frequency electromagnetic pulse. These signals are modulated using coils that create gradients of the magnetic field inside the scanner. Artifacts are created by using k-space transforms to perturb the

Table 1

Transforms included in TorchIO v0.18.0. Logos indicate the main library used to process the images. 🧠: NiBabel [15]; ⚡: SimpleITK [14]; 🧊: NumPy [30]; ⭕: PyTorch [8].

	Spatial	Intensity
Preprocessing	ToCanonical 🧠 Resample ⚡ Crop ⚡ Pad ⚡ CropOrPad ⚡	HistogramStandardization 🧊 [29] RescaleIntensity 🧊 ZNormalization ⭕
Augmentation	RandomAffine ⚡ RandomElasticDeformation ⚡ RandomFlip ⭕	RandomMotion 🧊 [42] RandomBiasField 🧊 [43] RandomGhosting 🧊 RandomSpike 🧊 [44] RandomBlur 🧊 RandomGamma ⭕ RandomNoise ⭕ RandomSwap ⭕ [45] RandomLabelsToImage ⭕ [46] RandomAnisotropy ⚡ [47]

Fourier space and generate corresponding intensity artifacts in image space. The forward and inverse Fourier transforms are computed using the Fast Fourier Transform (FFT) algorithm implemented in NumPy.

Random k-space spike artifact. Gradients applied at a very high duty cycle may produce bad data points, or noise spikes, in *k*-space [51]. These points in *k*-space generate a spike artifact, also known as Herringbone, crisscross or corduroy artifact, which manifests as uniformly-separated stripes in image space, as shown in Fig. 5i. This type of data augmentation has recently been used to estimate uncertainty through a heteroscedastic noise model [44].

Random k-space motion artifact. The *k*-space is often populated line by line, and the sample in the scanner is assumed to remain static. If a patient moves during the MRI acquisition, motion artifacts will appear in the reconstructed image. We implemented a method to simulate random motion artifacts (Fig. 5h) that has been used successfully for data augmentation to model uncertainty and improve segmentation [42].

Random k-space ghosting artifact. Organs motion such as respiration or cardiac pulsation may generate ghosting artifacts along the phase-encoding direction [51] (see Fig. 5j). We simulate this phenomenon by removing every *n*th plane of the *k*-space along one direction to generate *n* ghosts along that dimension, while keeping the center of *k*-space intact.

Random bias field artifact. Inhomogeneity of the static magnetic field in the MRI scanner produces intensity artifacts of very low spatial frequency along the entirety of the image. These artifacts can be simulated using polynomial basis functions [52], as shown in Fig. 5g.

2.2.3. Composability

All transforms can be composed in a linear fashion, as in the PyTorch torchvision library, or building a directed acyclic graphDAG using the OneOf transform (as in [10]). For example, a user might want to apply a random spatial augmentation transform to 50% of the samples using either an affine or an elastic transform, but they want the affine transform to be applied

to 80% of the augmented images, as the execution time is faster. Then, they might want to rescale the volume intensity for all images to be between 0 and 1. Fig. 6 shows a graph representing the transform composition. This transform composition can be implemented with just three statements:

```python
import torchio as tio

spatial_transforms = {
    tio.RandomElasticDeformation(): 0.2,
    tio.RandomAffine(): 0.8,
}

transform = tio.Compose([
    tio.OneOf(spatial_transforms, p=0.5),
    tio.RescaleIntensity((0, 1)),
])
```

Compose and OneOf are implemented as TorchIO transforms.

2.2.4. Extensibility

The Lambda transform can be passed an arbitrary callable object, which allows the user to augment the library with custom transforms without having a deep understanding of the underlying code.

Additionally, more complex transforms can be developed. For example, we implemented a TorchIO transform to simulate brain resection cavities from preoperative MR images within a self-supervised learning pipeline [53]. The RandomLabelsToImage transform may be used to simulate an image from a tissue segmentation. It can be composed with RandomAnisotropy to train neural networks agnostic to image contrast and resolution [46,47,54].
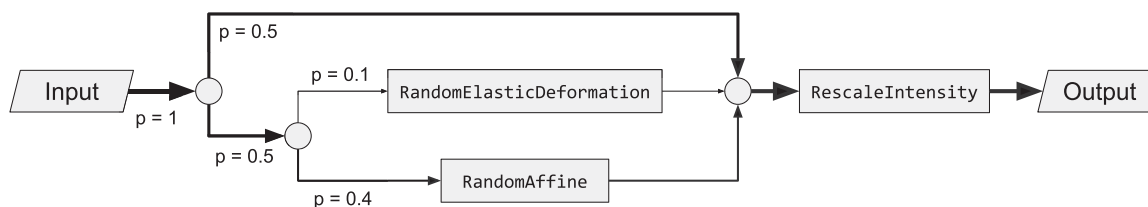
Fig. 6. Graph representation of the composed transform described in Section 2.2.3.

2.2.5. Reproducibility and traceability

To promote open science principles, we designed TorchIO to support experiment reproducibility and traceability.

All transforms support receiving Python primitives as arguments, which makes TorchIO suitable to be used with a configuration file associated to a specific experiment.

A history of all applied transforms and their computed random parameters is saved in the transform output so that the path in the DAG and the parameters used can be traced and reproduced. Furthermore, the `Subject` class includes a method to compose the transforms history into a single transform that may be used to reproduce the exact result (Section 2.2.3).

2.2.6. Invertibility

Inverting transforms is especially useful in scenarios where one needs to apply some transformation, infer a segmentation on the transformed data and then apply the inverse transformation to bring the inference into the original image space. The `Subject` class includes a method to invert the transformations applied. It does this by first inverting all transforms that are invertible, discarding the ones that are not. Then, it composes the invertible transforms into a single transform.

Transforms invertibility is most commonly applied to test-time augmentation [55] or estimation of aleatoric uncertainty [56] in the context of image segmentation.

3. Results

3.1. Code availability

All the code for TorchIO is available on GitHub[4]. We follow the semantic versioning system [57] to tag and release our library. Releases are published on the Zenodo data repository[5] to allow users to cite the specific version of the package they used in their experiments. The version described in this paper is `v0.18.0` [58]. Detailed API documentation is hosted on Read the Docs and comprehensive Jupyter notebook tutorials are hosted on Google Colaboratory, where users can run examples online. The library can be installed with a single line of code on Windows, macOS or Linux using the Pip Installs Packages (PIP) package manager: `pip install torchio`.

TorchIO has a strong community of users, with more than 900 stars on GitHub and more than 7000 Python Package Index (PyPI) downloads per month[6] as of July 2021.

3.1.1. Additional interfaces

The provided command-line interface (CLI) tool `torchio-transform` allows users to apply a transform to an image file without using Python. This tool can be used to visualize only the preprocessing and data augmentation pipelines and aid in experimental design for a given application. It can also be used in shell scripts to preprocess and augment datasets in cases where large storage is available and on-the-fly loading needs to be faster.

Additionally, we provide a graphical user interface (GUI) implemented as a Python scripted module within the *TorchIO* extension available in 3D Slicer [13]. It can be used to visualize the effect of the transforms parameters without any coding (Fig. 7). As with the CLI tool, users can experimentally assess preprocessing and data augmentation before network training to ensure the preprocessing pipeline is suitable for a given application.

3.2. Usage examples

In this section, we briefly describe the implementations of two medical image computing papers from the literature, pointing out the TorchIO features that could be used to replicate their experiments.

3.2.1. Super-resolution and synthesis of MRI

In [54], a method is proposed to simulate high-resolution T_1-weighted MRIs from images of different modalities and resolutions.

First, brain regions are segmented on publicly available datasets of brain MRI. During training, an MRI (`ScalarImage`) and the corresponding segmentation (`LabelMap`) corresponding to a specific subject (`Subject`) are sampled from the training dataset (`SubjectsDataset`). Next, the same spatial augmentation transform is applied to both images by composing an affine transform (`RandomAffine`) and a nonlinear diffeomorphic transform (`RandomElasticDeformation`). Then, a Gaussian mixture modelGMM conditioned on the labels is sampled at each voxel location to simulate an MRI of arbitrary contrast (`RandomLabelsToImage`) [46]. Finally, multiple degrading phenomena are simulated on the synthetic image: variability in the coordinate frames (`RandomAffine`), bias field inhomogeneities (`RandomBiasField`), partial-volume effects due to a large slice thickness during acquisition [47] (`RandomAnisotropy`), registration errors (`RandomAffine`), and resampling artifacts (`Resample`).

3.2.2. Adaptive sampling for segmentation of CT scans

In [59], CT scans that are too large to fit on a GPU are segmented using patch-based training with weighted sampling of patches. Discrepancies between labels and predictions are used to create error maps and patches are preferentially sampled from voxels with larger error.

During training, a CT scan (`ScalarImage`) and its corresponding segmentation (`LabelMap`) from a subject (`Subject`) are loaded and the same augmentation is performed to both by applying random rotations and scaling (`RandomAffine`). Then, voxel intensities are clipped to [−1000, 1000] (`RescaleIntensity`) and divided by a constant factor representing the standard deviation of the dataset (can be implemented with `Lambda`). As the CT scans are too large to fit in the GPU, patch-based training is used (`Queue`). To obtain high-resolution predictions and a large receptive field simultaneously, two patches of similar size but differ-

---

[4] https://github.com/fepegar/torchio.

[5] https://zenodo.org/.

[6] https://pypistats.org/packages/torchio.

Fig. 7. GUI for TorchIO, implemented as a 3D Slicer extension. In this example, the applied transforms are `RandomBiasField`, `RandomGhosting`, `RandomMotion`, `RandomAffine` and `RandomElasticDeformation`.

ent FOV are generated from each sampled patch: a context patch generated by downsampling the original patch (`Resample`) and a full-resolution patch with a smaller FOV (`CropOrPad`). At the end of each epoch, error maps for each subject (`Subject`) are computed as the difference between the labels and predictions. The error maps are used in the following epoch to sample patches with large errors more often (`WeightedSampler`). At inference time, a sliding window (`GridSampler`) is used to predict the segmentation patch by patch, and patches are aggregated to build the prediction for the whole input volume (`GridAggregator`).

4. Discussion

We have presented TorchIO, a new library to efficiently load, preprocess, augment and sample medical imaging data during the training of CNNs. It is designed in the style of the deep learning framework PyTorch to provide medical imaging specific preprocessing and data augmentation algorithms.

The main motivation for developing TorchIO as an open-source toolkit is to help researchers standardize medical image processing pipelines and allow them to focus on the deep learning experiments. It also encourages good open-science practices, as it supports experiment reproducibility and is version-controlled so that the software can be cited precisely.

The library is compatible with other higher-level deep learning frameworks for medical imaging such as MONAI. For example, users can benefit from TorchIO's MRI transforms and patch-based sampling while using MONAI's networks, losses, training pipelines and evaluation metrics.

The main limitation of TorchIO is that most transforms are not differentiable. The reason is that PyTorch tensors stored in TorchIO data structures must be converted to SimpleITK images or NumPy arrays within most transforms, making them not compatible with PyTorch's automatic differentiation engine. However, compatibility between PyTorch and ITK has recently been improved, partly thanks to the appearance of the MONAI project [60]. Therefore, TorchIO might provide differentiable transforms in the future, which could be used to implement, e.g., spatial transformer networks for image registration [61]. Another limitation is that many more transforms that are MRI-specific exist than for other imaging modalities such as CT or US. This is in part due to more users working on MRI applications and requesting MRI-specific transforms. However, we welcome contributions for other modalities as well.

In the future, we will work on extending the preprocessing and augmentation transforms to different medical imaging modalities such as CT or US, and improving compatibility with related works. The source code, as well as examples and documentation, are made publicly available online, on GitHub. We welcome feedback, feature requests, and contributions to the library, either by creating issues on the GitHub repository or by emailing the authors.

Declaration of Competing Interest

The authors declare no conflicts of interest.

References

[1] O. Çiçek, A. Abdulkadir, S.S. Lienkamp, T. Brox, O. Ronneberger, 3D U-Net: learning dense volumetric segmentation from sparse annotation, in: S. Ourselin, L. Joskowicz, M.R. Sabuncu, G. Unal, W. Wells (Eds.), Medical Image Computing and Computer-Assisted Intervention MICCAI 2016, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2016, pp. 424–432.

[2] D. Lu, K. Popuri, G.W. Ding, R. Balachandar, M.F. Beg, Alzheimers Disease Neuroimaging Initiative, Multimodal and multiscale deep neural networks for the early diagnosis of Alzheimer's disease using structural MR and FDG-PET images, Sci. Rep. 8 (1) (2018) 5697, doi:10.1038/s41598-018-22871-z.

[3] F. Chen, V. Taviani, I. Malkiel, J.Y. Cheng, J.I. Tamir, J. Shaikh, S.T. Chang, C.J. Hardy, J.M. Pauly, S.S. Vasanawala, Variable-density single-shot fast spin-echo MRI with deep learning reconstruction by using variational networks, Radiology 289 (2) (2018) 366–373, doi:10.1148/radiol.2018180445.

[4] S. Shan, W. Yan, X. Guo, E.I.-C. Chang, Y. Fan, Y. Xu, Unsupervised end–to-end learning for deformable medical image registration, arXiv:1711.08608 [cs] (2018). ArXiv: 1711.08608.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, in: OSDI'16, USENIX Association, USA, 2016, pp. 265–283.

[6] 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, arXiv:1711.06853 [cs] (2017). ArXiv: 1711.06853.

[7] 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, Comput. Methods Programs Biomed. 158 (2018) 113–122, doi:10.1016/j.cmpb.2018.01.025.

[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. Alch-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8026–8037.

[9] H. He, The State of Machine Learning Frameworks in 2019, 2019, http://bit.ly/3cjpliJ.

[10] A. Buslaev, V.I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, A.A. Kalinin, Albumentations: Fast and Flexible Image Augmentations, Information 11 (2) (2020) 125, doi:10.3390/info11020125. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute

[11] M. Larobina, L. Murino, Medical image file formats, J. Digit. Imaging 27 (2) (2014) 200–206, doi:10.1007/s10278-013-9657-9.

[12] M.J. Cardoso, M. Modat, R. Wolz, A. Melbourne, D. Cash, D. Rueckert, S. Ourselin, Geodesic information flows: spatially-variant graphs and their application to segmentation and fusion, IEEE Trans. Med. Imaging 34 (9) (2015) 1976–1988, doi:10.1109/TMI.2015.2418298.

[13] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J.V. Miller, S. Pieper, R. Kikinis, 3D slicer as an image computing platform for the quantitative imaging network, Magn. Reson. Imaging 30 (9) (2012) 1323–1341, doi:10.1016/j.mri.2012.05.001.

[14] B.C. Lowekamp, D.T. Chen, L. Ibez, D. Blezek, The design of SimpleITK, Front. Neuroinformatics 7 (2013) 45, doi:10.3389/fninf.2013.00045.

[15] M. Brett, C.J. Markiewicz, M. Hanke, M.-A. Ct, B. Cipollini, P. McCarthy, C.P. Cheng, Y.O. Halchenko, M. Cottaar, S. Ghosh, E. Larson, D. Wassermann, S. Gerhard, G.R. Lee, H.-T. Wang, E. Kastman, A. Rokem, C. Madison, F.C. Morency, B. Moloney, M. Goncalves, C. Riddell, C. Burns, J. Millman, A. Gramfort, J. Leppkangas, R. Markello, J.J. van den Bosch, R.D. Vincent, H. Braun, K. Subramaniam, D. Jarecka, K.J. Gorgolewski, P.R. Raamana, B.N. Nichols, E.M. Baker, S. Hayashi, B. Pinsard, C. Haselgrove, M. Hymers, O. Esteban, S. Koudoro, N.N. Oosterhof, B. Amirbekian, I. Nimmo-Smith, L. Nguyen, S. Reddigari, S. St-Jean, E. Panfilov, E. Garyfallidis, G. Varoquaux, J. Kaczmarzyk, J.H. Legarreta, K.S. Hahn, O.P. Hinds, B. Fauber, J.-B. Poline, J. Stutters, K. Jordan, M. Cieslak, M.E. Moreno, V. Haenel, Y. Schwartz, B.C. Darwin, B. Thirion, D. Papadopoulos Orfanos, F. Pérez-García, I. Solovey, I. Gonzalez, J. Palasubramaniam, J. Lecher, K. Leinweber, K. Raktivan, P. Fischer, P. Gervais, S. Gadde, T. Ballinger, T. Roos, V.R. Reddam, freec84, nipy/nibabel: 3.0.1, 2020, https://zenodo.org/record/3628482.XlyGkJP7S8o. doi:10.5281/zenodo.3628482

[16] A.B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernndez, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, others, imgaug, 2020, https://github.com/aleju/imgaug.

[17] wiredfool, A. Clark, Hugo, A. Murray, A. Karpinsky, C. Gohlke, B. Crowell, D. Schmidt, A. Houghton, S. Johnson, S. Mani, J. Ware, D. Caro, S. Kossouho, E.W. Brown, A. Lee, M. Korobov, M. Grny, E.S. Santana, N. Pieuchot, O. Tonnhofer, M. Brown, B. Pierre, J.C. Abela, L.J. Solberg, F. Reyes, A. Buzanov, Y. Yu, eliempje, F. Tolf, Pillow: 3.1.0, 2016, https://zenodo.org/record/44297.Xlx04pP7S8o. doi:10.5281/zenodo.44297.

[18] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, in: NIPS'12, Curran Associates Inc., USA, 2012, pp. 1097–1105.

[19] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A simple framework for contrastive learning of visual representations, in: International Conference on Machine Learning, PMLR, 2020, pp. 1597–1607. ISSN: 2640-3498

[20] F. Milletari, N. Navab, S.-A. Ahmadi, V-Net: fully convolutional neural networks for volumetric medical image segmentation, in: 2016 Fourth International Conference on 3D Vision (3DV), 2016, pp. 565–571, doi:10.1109/3DV.2016.79.

[21] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, PMLR, 2015, pp. 448–456. ISSN: 1938-7228

[22] O. Lucena, R. Souza, L. Rittner, R. Frayne, R. Lotufo, Convolutional neural networks for skull-stripping in brain MR imaging using silver standard masks, Artif. Intell. Med. 98 (2019) 48–58, doi:10.1016/j.artmed.2019.06.008.

[23] W. Li, G. Wang, L. Fidon, S. Ourselin, M.J. Cardoso, T. Vercauteren, On the compactness, efficiency, and representation of 3d convolutional networks: brain parcellation as a pretext task, in: M. Niethammer, M. Styner, S. Aylward, H. Zhu, I. Oguz, P.-T. Yap, D. Shen (Eds.), Information Processing in Medical Imaging, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 348–360, doi:10.1007/978-3-319-59050-9_28.

[24] S. Nikolov, S. Blackwell, R. Mendes, J. De Fauw, C. Meyer, C. Hughes, H. Askham, B. Romera-Paredes, A. Karthikesalingam, C. Chu, D. Carnell, C. Boon, D. D'Souza, S.A. Moinuddin, K. Sullivan, D.R. Consortium, H. Montgomery, G. Rees, R. Sharma, M. Suleyman, T. Back, J.R. Ledsam, O. Ronneberger, Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy, arXiv:1809.04430 [physics, stat] (2018). ArXiv: 1809.04430

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: a large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255, doi:10.1109/CVPR.2009.5206848. ISSN: 1063-6919

[26] K. Weiss, T.M. Khoshgoftaar, D. Wang, A survey of transfer learning, J. Big Data 3 (1) (2016) 9, doi:10.1186/s40537-016-0043-6.

[27] V. Cheplygina, Cats or CAT scans: transfer learning from natural or medical image source data sets? Curr. Opin. Biomed. Eng. 9 (2019) 21–27, doi:10.1016/j.cobme.2018.12.005.

[28] M. Raghu, C. Zhang, J. Kleinberg, S. Bengio, Transfusion: understanding transfer learning for medical imaging, in: H. Wallach, H. Larochelle, A. Beygelzimer, F.d. Alch-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019.

[29] L.G. Nyl, J.K. Udupa, On standardizing the MR image intensity scale, Magn. Reson. Med. 42 (6) (1999) 1072–1081, doi:10.1002/(sici)1522-2594(199912)42:6<1072::aid-mrm11>3.0.co;2-m.

[30] S. van der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: a structure for efficient numerical computation, Comput. Sci. Eng. 13 (2) (2011) 22–30, doi:10.1109/MCSE.2011.37. Conference Name: Computing in Science Engineering.

[31] V.V. Valindria, N. Pawlowski, M. Rajchl, I. Lavdas, E.O. Aboagye, A.G. Rockall, D. Rueckert, B. Glocker, Multi-modal learning from unpaired images: application to multi-organ segmentation in CT and MRI, in: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), 2018, pp. 547–556, doi:10.1109/WACV.2018.00066.

[32] C.S. Perone, cclauss, E. Saravia, P.L. Ballester, MohitTare, perone/medicaltorch: Release v0.2, 2018, https://zenodo.org/record/1495335.XlqwUZP7S8o. doi:10.5281/zenodo.1495335.

[33] F. Isensee, P. Jger, J. Wasserthal, D. Zimmerer, J. Petersen, S. Kohl, J. Schock, A. Klein, T. Ro, S. Wirkert, P. Neher, S. Dinkelacker, G. Köhler, K. Maier-Hein, batchgenerators - a python framework for data augmentation, 2020, https://zenodo.org/record/3632567.Xlqnb5P7S8o. doi:10.5281/zenodo.3632567.

[34] F. Isensee, P.F. Jaeger, S.A.A. Kohl, J. Petersen, K.H. Maier-Hein, nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation, Nat. Methods 18 (2) (2021) 203–211, doi:10.1038/s41592-020-01008-z. Number: 2 Publisher: Nature Publishing Group

[35] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0: fundamental algorithms for scientific computing in Python, Nat. Methods (2020) 1–12, doi:10.1038/s41592-019-0686-2.

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

[37] F. Kofler, I. Ezhov, F. Isensee, F. Balsiger, C. Berger, M. Koerner, J. Paetzold, H. Li, S. Shit, R. McKinley, S. Bakas, C. Zimmer, D. Ankerst, J. Kirschke, B. Wiestler, B.H. Menze, Are we using appropriate segmentation metrics? Identifying correlates of human expert perception for CNN training beyond rolling the DICE coefficient, arXiv:2103.06205 [cs, eess] (2021). ArXiv: 2103.06205

[38] N. Ma, W. Li, R. Brown, Y. Wang, B. Gorman, Behrooz, H. Johnson, I. Yang, E. Kerfoot, Y. Li, M. Adil, Y.-T. Hsieh, charliebudd, A. Aggarwal, C. Trentz, adam aji, B. Murray, G. Daroach, P.-D. Tudosiu, myron, M. Graham, Balamurali, C. Baker, J. Sellner, L. Fidon, A. Powers, G. Leroy, Alxaline, D. Schulz, Project-MONAI/MONAI: 0.5.0, 2021, https://zenodo.org/record/4679866.YImZHZNKgWo. doi:10.5281/zenodo.4679866.

[39] F. Mancolo, Eisen: a python package for solid deep learning, arXiv:2004.02747 [cs, eess] (2020). ArXiv: 2004.02747

[40] M.B.M. Ranzini, L. Fidon, S. Ourselin, M. Modat, T. Vercauteren, MONAIfbs: MONAI-based fetal brain MRI deep learning segmentation, arXiv:2103.13314 [cs, eess] (2021). ArXiv: 2103.13314

[41] F. Chollet, others, Keras, 2015.

[42] R. Shaw, C. Sudre, S. Ourselin, M.J. Cardoso, MRI k-space motion artefact augmentation: model robustness and task-specific uncertainty, in: International Conference on Medical Imaging with Deep Learning, 2019, pp. 427–436. http://proceedings.mlr.press/v102/shaw19a.html..

[43] C.H. Sudre, M.J. Cardoso, S. Ourselin, Longitudinal segmentation of age-related white matter hyperintensities, Med. Image Anal. 38 (2017) 50–64, doi:10.1016/j.media.2017.02.007.

[44] R. Shaw, C.H. Sudre, S. Ourselin, M.J. Cardoso, A heteroscedastic uncertainty model for decoupling sources of MRI image quality, in: Medical Imaging with Deep Learning, PMLR, 2020, pp. 733–742. http://proceedings.mlr.press/v121/shaw20a.html.. ISSN: 2640-3498

[45] L. Chen, P. Bentley, K. Mori, K. Misawa, M. Fujiwara, D. Rueckert, Self-supervised learning for medical image analysis using image context restoration, Med. Image Anal. 58 (2019) 101539, doi:10.1016/j.media.2019.101539.

[46] B. Billot, D.N. Greve, K.V. Leemput, B. Fischl, J.E. Iglesias, A. Dalca, A learning strategy for contrast-agnostic MRI segmentation, in: Medical Imaging with Deep Learning, PMLR, 2020, pp. 75–93. ISSN: 2640-3498

[47] B. Billot, E. Robinson, A.V. Dalca, J.E. Iglesias, Partial volume segmentation of brain MRI scans of any resolution and contrast, in: A.L. Martel, P. Abolmaesumi, D. Stoyanov, D. Mateus, M.A. Zuluaga, S.K. Zhou, D. Racoceanu, L. Joskowicz (Eds.), Medical Image Computing and Computer Assisted Intervention MICCAI 2020, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 177–187, doi:10.1007/978-3-030-59728-3_18.

[48] L.G. Nyl, J.K. Udupa, X. Zhang, New variants of a method of MRI scale standardization, IEEE Trans. Med. Imaging 19 (2) (2000) 143–150, doi:10.1109/42.836373.

[49] C.J. Holmes, R. Hoge, L. Collins, R. Woods, A.W. Toga, A.C. Evans, Enhancement of MR images using registration for signal averaging, J. Comput. Assist. Tomogr. 22 (2) (1998) 324–333, doi:10.1097/00004728-199803000-00032.

[50] A.O. Omigbodun, F. Noo, M. McNitt-yy, W. Hsu, S.S. Hsieh, The effects of physics-based data augmentation on the generalizability of deep neural networks: demonstration on nodule false-positive reduction, Med. Phys. 46 (10) (2019) 4563–4574, doi:10.1002/mp.13755.

[51] J. Zhuo, R.P. Gullapalli, MR artifacts, safety, and quality control, RadioGraphics 26 (1) (2006) 275–297, doi:10.1148/rg.261055134.

[52] K. Van Leemput, F. Maes, D. Vandermeulen, P. Suetens, Automated model-based tissue classification of MR images of the brain, IEEE Trans. Med. Imaging 18 (10) (1999) 897–908, doi:10.1109/42.811270.

[53] F. Pérez-García, R. Rodionov, A. Alim-Marvasti, R. Sparks, J.S. Duncan, S. Ourselin, Simulation of brain resection for cavity segmentation using self--supervised and semi-supervised learning, in: A.L. Martel, P. Abolmaesumi, D. Stoyanov, D. Mateus, M.A. Zuluaga, S.K. Zhou, D. Racoceanu, L. Joskowicz (Eds.), Medical Image Computing and Computer Assisted Intervention MICCAI 2020, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 115–125.

[54] J.E. Iglesias, B. Billot, Y. Balbastre, A. Tabari, J. Conklin, D. Alexander, P. Golland, B. Edlow, B. Fischl, Joint super-resolution and synthesis of 1 mm isotropic MP-RAGE volumes from clinical MRI exams with scans of different orientation, resolution and contrast, arXiv preprint arXiv:2012.13340 (2020).

[55] N. Moshkov, B. Mathe, A. Kertesz-Farkas, R. Hollandi, P. Horvath, Test-time augmentation for deep learning-based cell segmentation on microscopy images, Sci. Rep. 10 (1) (2020) 5068, doi:10.1038/s41598-020-61808-3. Number: 1 Publisher: Nature Publishing Group

[56] G. Wang, W. Li, M. Aertsen, J. Deprest, S. Ourselin, T. Vercauteren, Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks, Neurocomputing 338 (2019) 34–45, doi:10.1016/j.neucom.2019.01.103.

[57] T. Preston-Werner, Semantic Versioning 2.0.0, 2020, Library Catalog: semver.org, https://semver.org/.

[58] F. Pérez-García, fepegar/torchio: TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning (Nov. 2020). doi:10.5281/zenodo.4296288

[59] L. Berger, H. Eoin, M.J. Cardoso, S. Ourselin, An adaptive sampling scheme to efficiently train fully convolutional networks for semantic segmentation, in: M. Nixon, S. Mahmoodi, R. Zwiggelaar (Eds.), Medical Image Understanding and Analysis, Communications in Computer and Information Science, Springer International Publishing, Cham, 2018, pp. 277–286, doi:10.1007/978-3-319-95921-4_26.

[60] M. McCormick, D. Zukić, S.A. on, ITK 5.2 Release Candidate 3 available for testing, 2021, https://blog.kitware.com/itk-5-2-release-candidate-3-available-for-testing/.

[61] M.C.H. Lee, O. Oktay, A. Schuh, M. Schaap, B. Glocker, Image-and-spatial transformer networks for structure-guided image registration, in: D. Shen, T. Liu, T.M. Peters, L.H. Staib, C. Essert, S. Zhou, P.-T. Yap, A. Khan (Eds.), Medical Image Computing and Computer Assisted Intervention MICCAI 2019, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2019, pp. 337–345, doi:10.1007/978-3-030-32245-8_38.