# Getting started

## Installation

The Python package is hosted on the Python Package Index (PyPI).

To install the latest PyTorch version before installing TorchIO, it is recommended to use light-the-torch:

```
$ pip install light-the-torch && ltt install torch
```

The latest published version of TorchIO can be installed using Pip Installs Packages (`pip`):

```
$ pip install torchio
```

To upgrade to the latest published version, use:

```
$ pip install --upgrade torchio
```

If you would like to install Matplotlib to use the plotting features, use:

```
$ pip install torchio[plot]
```

If you are on Windows and have trouble installing TorchIO, try installing PyTorch with conda before pip-installing TorchIO.

## Hello, World!

This example shows the basic usage of TorchIO, where an instance of `SubjectsDataset` is passed to a PyTorch `DataLoader` to generate training batches of 3D images that are loaded, preprocessed and augmented on the fly, in parallel:

```
import torch
import torchio as tio
from torch.utils.data import DataLoader

# Each instance of tio.Subject is passed arbitrary keyword arguments.
# Typically, these arguments will be instances of tio.Image
subject_a = tio.Subject(
    t1=tio.ScalarImage('subject_a.nii.gz'),
    label=tio.LabelMap('subject_a.nii'),
    diagnosis='positive',
)

# Image files can be in any format supported by SimpleITK or NiBabel, including DICOM
subject_b = tio.Subject(
    t1=tio.ScalarImage('subject_b_dicom_folder'),
    label=tio.LabelMap('subject_b_seg.nrrd'),
    diagnosis='negative',
)

# Images may also be created using PyTorch tensors or NumPy arrays
tensor_4d = torch.rand(4, 100, 100, 100)
subject_c = tio.Subject(
    t1=tio.ScalarImage(tensor=tensor_4d),
    label=tio.LabelMap(tensor=(tensor_4d > 0.5)),
    diagnosis='negative',
)

subjects_list = [subject_a, subject_b, subject_c]

# Let's use one preprocessing transform and one augmentation transform
# This transform will be applied only to scalar images:
rescale = tio.RescaleIntensity(out_min_max=(0, 1))

# As RandomAffine is faster then RandomElasticDeformation, we choose to
# apply RandomAffine 80% of the times and RandomElasticDeformation the rest
# Also, there is a 25% chance that none of them will be applied
spatial = tio.OneOf({
        tio.RandomAffine(): 0.8,
        tio.RandomElasticDeformation(): 0.2,
    },
    p=0.75,
)

# Transforms can be composed as in torchvision.transforms
transforms = [rescale, spatial]
transform = tio.Compose(transforms)

# SubjectsDataset is a subclass of torch.data.utils.Dataset
subjects_dataset = tio.SubjectsDataset(subjects_list, transform=transform)

# Images are processed in parallel thanks to a PyTorch DataLoader
training_loader = DataLoader(subjects_dataset, batch_size=4, num_workers=4)

# Training epoch
for subjects_batch in training_loader:
    inputs = subjects_batch['t1'][tio.DATA]
    target = subjects_batch['label'][tio.DATA]
```

## Tutorials

Open in Colab

The best way to quickly understand and try the library is the Jupyter Notebooks hosted on Google Colab.

They include multiple examples and visualization of most of the classes, including training of a 3D U-Net for brain segmentation on $T_1$-weighted MRI with full volumes and with subvolumes (aka patches or windows).