intensity-normalization Documentation

Release 2.2.4

Jacob Reinhold

May 31, 2023

CONTENTS:

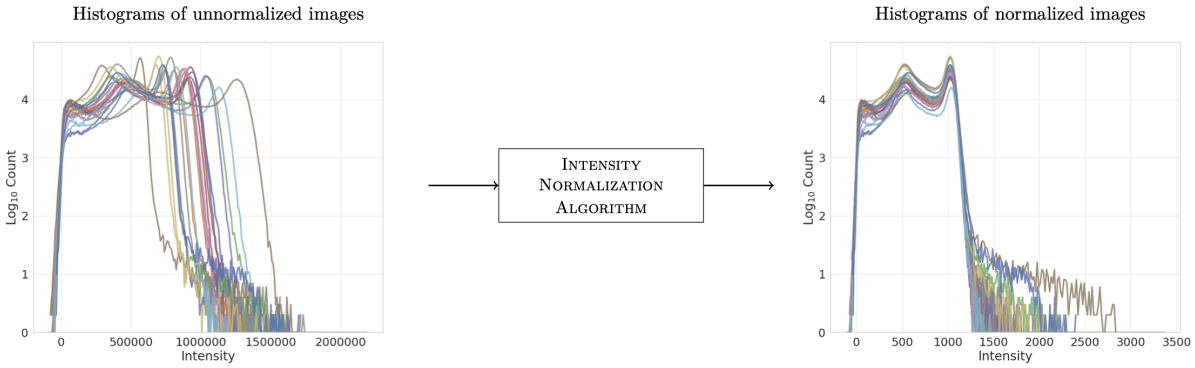
CHAPTER

ONE

INTENSITY-NORMALIZATION

This package contains various methods to normalize the intensity of various modalities of magnetic resonance (MR) images, e.g., T1-weighted (T1-w), T2-weighted (T2-w), FLuid-Attenuated Inversion Recovery (FLAIR), and Proton Density-weighted (PD-w).

The basic functionality of this package can be summarized in the following image:



where the left-hand side are the histograms of the intensities for a set of *unnormalized* images (from the same scanner with the same protocol!) and the right-hand side are the histograms after (FCM) normalization.

We used this package to explore the impact of intensity normalization on a synthesis task (**pre-print** available here).

Note that while this release was carefully inspected, there may be bugs. Please submit an issue if you encounter a problem.

1.1 Methods

We implement the following normalization methods (the names of the corresponding command-line interfaces are to the right in parentheses):

1.1.1 Individual time-point normalization methods

- Z-score normalization (`zscore-normalize`)
- Fuzzy C-means (FCM)-based tissue-based mean normalization (`fcm-normalize`)
- Kernel Density Estimate (KDE) WM mode normalization (`kde-normalize`)
- WhiteStripe[1] (`ws-normalize`)

1.1.2 Sample-based normalization methods

- Least squares (LSQ) tissue mean normalization (`lsq-normalize`)
- Piecewise Linear Histogram Matching (Nyúl & Udupa)[2][3] (`nyul-normalize`)
- RAVEL[4] (`ravel-normalize`)

Individual image-based methods normalize images based on one time-point of one subject.

Sample-based methods normalize images based on a *set* of images of (usually) multiple subjects of the same modality.

Recommendation on where to start: If you are unsure which one to choose for your application, try FCM-based WM-based normalization (assuming you have access to a T1-w image for all the time-points). If you are getting odd results in non-WM tissues, try least squares tissue normalization (which minimizes the least squares distance between CSF, GM, and WM tissue means within a set).

Read about the methods and how they work. If you have a non-standard modality, e.g., a contrast-enhanced image, read about how the methods work and determine which method would work for your use case. Make sure you plot the foreground intensities (with the `-p` option in the CLI or the `HistogramPlotter` in the Python API) to validate the normalization results.

All algorithms except Z-score (`zscore-normalize`) *and the Piecewise Linear Histogram Matching* (`nyul-normalize`) *are specific to images of the brain.*

[1] R. T. Shinohara, E. M. Sweeney, J. Goldsmith, N. Shiee, F. J. Mateen, P. A. Calabresi, S. Jarso, D. L. Pham, D. S. Reich, and C. M. Crainiceanu, "Statistical normalization techniques for magnetic resonance imaging," NeuroImage Clin., vol. 6, pp. 9–19, 2014.

[2] N. Laszlo G and J. K. Udupa, "On Standardizing the MR Image Intensity Scale," Magn. Reson. Med., vol. 42, pp. 1072–1081, 1999.

[3] M. Shah, Y. Xiao, N. Subbanna, S. Francis, D. L. Arnold, D. L. Collins, and T. Arbel, "Evaluating intensity normalization on MRIs of human brain with multiple sclerosis," Med. Image Anal., vol. 15, no. 2, pp. 267–282, 2011.

[4] J. P. Fortin, E. M. Sweeney, J. Muschelli, C. M. Crainiceanu, and R. T. Shinohara, "Removing inter-subject technical variability in magnetic resonance imaging studies," NeuroImage, vol. 132, pp. 198–212, 2016.

1.2 Motivation

Intensity normalization is an important pre-processing step in many image processing applications regarding MR images since MR images have an inconsistent intensity scale across (and within) sites and scanners due to, e.g.,:

1) the use of different equipment,
2) different pulse sequences and scan parameters,
3) and a different environment in which the machine is located.

Importantly, the inconsistency in intensities *isn't a feature* of the data (unless you want to classify the scanner/site from which an image came)—it's an artifact of the acquisition process. The inconsistency causes a problem with machine learning-based image processing methods, which usually assume the data was gathered iid from some distribution.

1.3 Install

The easiest way to install the package is through the following command:

```
pip install intensity-normalization
```

To install from the source directory, clone the repo and run:

```
python setup.py install
```

Note the package antspy is required for the RAVEL normalization routine, the preprocessing tool as well as the co-registration tool, but all other normalization and processing tools work without it. To install the antspy package along with the RAVEL, preprocessing, and co-registration CLI, install with:

```
pip install "intensity-normalization[ants]"
```

1.4 Basic Usage

See the 5 minute overview for a more detailed tutorial.

In addition to the above small tutorial, here is consolidated documentation.

Call any executable script with the `-h` flag to see more detailed instructions about the proper call.

Note that **brain masks** (or already skull-stripped images) are required for most of the normalization methods. The brain masks do not need to be perfect, but each mask needs to remove most of the tissue outside the brain. Assuming you have T1-w images for each subject, an easy and robust method for skull-stripping is ROBEX[5].

If the images are already skull-stripped, you don't need to provide a brain mask. The foreground will be automatically estimated and used.

You can install ROBEX—and get python bindings for it at the same time–with the package pyrobex (installable via `pip install pyrobex`).

[5] Iglesias, Juan Eugenio, Cheng-Yi Liu, Paul M. Thompson, and Zhuowen Tu. "Robust brain extraction across datasets and comparison with publicly available methods." IEEE transactions on medical imaging 30, no. 9 (2011): 1617-1634.

1.4.1 Individual time-point normalization methods

Example call to a individual time-point normalization CLI:

```
fcm-normalize t1w_image.nii -m brain_mask.nii
```

1.4.2 Sample-based normalization methods

Example call to a sample-based normalization CLI:

```
nyul-normalize images/ -m masks/ -o nyul_normalized/ -v
```

where `images/` is a directory full of N MR images and `masks/` is a directory full of N corresponding brain masks, `nyul_normalized` is the output directory for the normalized images, and `-v` controls the verbosity of the output.

The command line interface is standard across all sampled-based normalization routines (i.e., you should be able to run all sample-based normalization routines with the same call as in the above example); however, each has unique method-specific options.

1.5 Potential Pitfalls

1) This package was developed to process **adult human** MR images; neonatal, pediatric, and animal MR images *should* also work but—if the data has different proportions of tissues or differences in relative intensity among tissue types compared with adults—the normalization may fail. The `nyul-normalize` method, in particular, will fail hard if you train it on adult data and test it on non-adult data (or vice versa). Please open an issue if you encounter a problem with the package when normalizing non-adult human data.

2) When we refer to any specific modality, it is referring to a **non-contrast** version unless otherwise stated. Using a contrast image as input to a method that assumes non-contrast will produce suboptimal results. One potential way to normalize contrast images with this package is to 1) find a tissue that is not affected by the contrast (e.g., grey matter) and normalize based on some summary statistic of that (where the tissue mask was found on a non-contrast image); 2) use a simplistic (but non-robust) method like Z-score normalization.

   Read about the methods and how they work to decide which method would work best for your contrast-enhanced images.

1.6 Contributing

Help wanted! See CONTRIBUTING.rst for details and/or reach out to me if you'd like to contribute. Credit will be given! If you want to add a method, I'll be happy to add your reference to the citation section below.

1.7 Test Package

Unit tests can be run from the main directory as follows:

```
pytest tests
```

1.8 Citation

If you use the `intensity-normalization` package in an academic paper, please cite the corresponding paper:

```
@inproceedings{reinhold2019evaluating,
  title={Evaluating the impact of intensity normalization on {MR} image synthesis},
  author={Reinhold, Jacob C and Dewey, Blake E and Carass, Aaron and Prince, Jerry L},
  booktitle={Medical Imaging 2019: Image Processing},
  volume={10949},
  pages={109493H},
  year={2019},
  organization={International Society for Optics and Photonics}}
```

1.9 References

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install intensity-normalization, run this command in your terminal:

```
$ pip install intensity_normalization
```

This is the preferred method to install intensity-normalization, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for intensity-normalization can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/jcreinhold/intensity_normalization
```

Or download the tarball:

```
$ curl -OJL https://github.com/jcreinhold/intensity_normalization/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

EXAMPLE USAGE

3.1 Individual timepoint-based normalization

The below section focuses on Fuzzy C-means (FCM)-based normalization, but can be used as a reference for all individual timepoint-based normalization methods.

Once the package is installed, if you just want to do some sort of normalization and not think too much about it, a reasonable choice is Fuzzy C-means (FCM)-based normalization. Note that FCM requires access to a (*non-gadolinium-enhanced*) T1-w image, if this is not possible then I would recommend doing either z-score or KDE normalization for simple normalization tasks. The FCM method also requires a brain mask for the image, although the brain mask doesn't need to be perfect (ROBEX works fine for this purpose).

Note that FCM-based normalization acts on the image by calculating the specified tissue mean, e.g., white matter (WM) mean and setting that to a specified value (the default is 1 in the code base although that is a tunable parameter). Our FCM-based normalization method requires that a timepoint contains a T1-w image. We use the T1-w image and the brain mask to create a tissue mask over which we calculate the tissue mean. We then normalize as previously stated (see here for more detail). This tissue mask can then be used to normalize the remaining contrasts in the set of images for a specific patient assuming that the remaining contrast images are registered to the T1-w image.

Since most of the command line interfaces (CLIs) are installed along with the package, we can run `fcm-normalize` in the terminal to normalize a T1-w image and create a WM mask by running the following command (replacing paths as necessary):

```
fcm-normalize t1w_image.nii.gz -m brain_mask.nii.gz -o t1w_norm.nii.gz -v -mo t1 -tt wm
```

This will output the normalized T1-w image to `t1w_norm.nii.gz` and will additionally create a tissue mask for the WM in the same directory with `wm_membership` appended to the filename. You can then input the WM membership back in to the program to normalize an image of a different contrast, e.g. for T2:

```
fcm-normalize t2w_image.nii.gz -tm t1w_image_wm_membership.nii.gz -o t2w_norm.nii.gz -v -
↪mo t2
```

You can run `fcm-normalize -h` to see more options, but the above covers most of the details necessary to run FCM normalization on a single image.

You can process a directory of images like this:

```
find -L t1w_image_dir -type f -name '*.nii*' -exec fcm-normalize "{}" \;
```

and it will FCM normalize all the images in the directory `t1w_image_dir/` assuming all the images are skull-stripped.

If you want to quickly inspect the normalization results on a directory (as in the last command), you can append the `-p` flag which will create a plot of the histograms inside the brain mask of the normalized image (or images if you're using a sample-based method).

For the above case, you should expect to see alignment around the intensity level of 1 (or whatever the `--norm-value` is set to). You can also use the `plot-histograms` CLI which is also installed (see here for documentation). A use case of the `plot-histograms` command would be to plot the histograms of all individual timepoint-based normalized images, or to inspect the histograms of a set of images before and after normalization.

3.2 Example usage on a directory for sample-based methods

The sample-based normalization CLIs (RAVEL, Nyul, and LSQ) operate on a directory of images (either 2D or 3D). That is, suppose you have a directory of images `img_dir` that contains images, like so:

```
├── img_dir
│   ├── img1.ext
│   ├── img2.ext
│   ├── img3.ext
│   ├── ...
│   ├── imgN.ext
```

In addition to the images, the normalization CLIs also can take brain masks as input; the masks (or absence of masks) can affect normalization quality. If you have brain masks for the corresponding images (for example, the images in `img_dir`), they should be setup like so:

```
├── mask_dir
│   ├── mask1.ext
│   ├── mask2.ext
│   ├── mask3.ext
│   ├── ...
│   ├── maskN.ext
```

Note that when both `img_dir` and `mask_dir` are sorted alphabetically, each mask should correspond to the correct image. Other than that, the name of the image or mask is not important.

If you have a setup as shown above (with `img_dir` and `mask_dir`), you can call any sample-based normalization CLI on `img_dir` to normalize all images in that directory. For example, with `nyul-normalize` (assuming that `img_dir` contains T1-w images, in this example) the call would be something like:

```
nyul-normalize img_dir -m mask_dir -o out_dir -v -mo t1 -tt wm
```

3.3 Additional Provided Routines

There a variety of other routines provided for analysis and preprocessing. The CLI names are:

1) `plot-histograms` - plot the histograms of a directory of images on one figure for comparison
2) `tissue-membership` - find and output tissue membership of an input image

The following (along with `ravel-normalize`) are available only if you install `intensity-normalization` with `pip install "intensity-normalization[ants]"`.

1) `coregister` - coregister via a rigid and affine transformation
2) `preprocess` - resample, N4-correct, and reorient the image and mask

3.4 Python API for normalization methods

While in this tutorial we discussed interfacing with the package through command line interfaces (CLIs), it is worth noting that the normalization routines (and other utilities) are available as via a Python API which you can import into your project or script, e.g.,

```
import nibabel as nib
from intensity_normalization.typing import Modality, TissueType
from intensity_normalization.normalize.fcm import FCMNormalize

image = nib.load("test_t1w_image.nii").get_fdata()  # assume skull-stripped otherwise␣
↪load mask too

fcm_norm = FCMNormalize(tissue_type=TissueType.WM)
normalized = fcm_norm(image)

# now normalize the co-registered, corresponding T2-w image
t2w_image = nib.load("test_t2w_image.nii").get_fdata()
t2w_normalized = fcm_norm(t2w_image, modality=Modality.T2)

# to use a brain mask instead of a skull-stripped image do this:
mask = nib.load("brain_mask.nii").get_fdata()
normalized_t1w = fcm_norm(image, mask)
# the WM mask is an attribute in the class, so normalize the t2 with:
normalized_t2w = fcm_norm(t2w_image, modality=Modality.T2)

# make a new instance of the normalizer to normalize a new image, i.e.:
new_image = nib.load("test_t1w_image_2.nii")
fcm_norm = FCMNormalize(tissue_type=TissueType.WM)
normalized = fcm_norm(new_image.get_fdata())

# you can save the normalized image with nibabel as follows:
nib.Nifti1Image(normalized, new_image.affine).to_filename("normalized.nii")
```

Generally, the normalization methods have a similar interface, although some methods (RAVEL, Nyul, and LSQ) require a list of images (and, optionally, corresponding masks), like so:

```
normalizer = NormalizerClass(**init_args)
normalizer(image, mask, modality)
```

where `init_args` is a dictionary of method dependent keyword arguments, `image` is something like a a numpy array (i.e., the pixel data of an image, see pymedio for a flexible package to open various types of medical image; it returns them as a subclass of np.ndarray); `mask` is one of `None` (or not provided), or something like a numpy array (like `image`); `modality` is a string representing the modality.

3.4.1 Opening and normalizing images with pymedio

There are many medical image readers available in Python, and so long as you can convert the pixel/voxel data to a numpy array, you use them. All that intensity-normalization requires is an array-like data type.

The above shows an example with nibabel and for NIfTI images. If you have DICOM images or other formats, a flexible image reader that requires minimal storage and dependencies is pymedio. Assuming you install pymedio like pip install "pymedio[all]", an example opening an image and normalizing it is shown below.

Assume test_t1w_image is a directory of DICOM images:

```
import pymedio.image as mioi
from intensity_normalization.typing import Modality, TissueType
from intensity_normalization.normalize.fcm import FCMNormalize

image = mioi.Image.from_path("test_t1w_image/")  # assume skull-stripped otherwise load␣
↪mask too

fcm_norm = FCMNormalize(tissue_type=TissueType.WM)
normalized = fcm_norm(image)

# now normalize the co-registered, corresponding T2-w image
t2w_image = mioi.Image.from_path("test_t2w_image.nii")  # or some other extension/
↪directory of DICOM
t2w_normalized = fcm_norm(t2w_image, modality=Modality.T2)

# to use a brain mask instead of a skull-stripped image do this:
mask = mioi("brain_mask.nii")
normalized_t1w = fcm_norm(image, mask)
# the WM mask is an attribute in the class, so normalize the t2 with:
normalized_t2w = fcm_norm(t2w_image, modality=Modality.T2)

# you can save the normalized image with pymedio as follows:
normalized_t2w.to_filename("normalized.nii")
```

pymedio images can be used everywhere in intensity-normalization that a numpy array can be used. It will hold the affine transformation matrix as an attribute (at .affine) and can be operated on like a numpy array without losing the affine transformation matrix.

3.4.2 Validating normalization results

You should validate the results of normalization by plotting the histograms of the foreground image intensities before and after normalization, e.g.,

```
from intensity_normalization.plot.histogram import HistogramPlotter, plot_histogram

import matplotlib.pyplot as plt
import nibabel as nib
from intensity_normalization.typing import Modality, TissueType
from intensity_normalization.normalize.fcm import FCMNormalize

image = nib.load("test_t1w_image.nii").get_fdata()
mask = nib.load("test_t1w_brain_mask.nii").get_fdata()
```

(continues on next page)

(continued from previous page)

```
fcm_norm = FCMNormalize(tissue_type=TissueType.WM)
normalized = fcm_norm(image, mask, modality=Modality.T1)

plot_histogram(image, mask)
plt.title("Unnormalized")
plt.show()

plot_histogram(norm, mask)
plt.title("FCM Normalized")
plt.show()

# or if you have a set of images
images = [nib.load(fn).get_fdata() for fn in filenames]
masks = [nib.load(fn).get_fdata() for fn in mask_filenames]
normed = [fcm_norm(img, msk) for img, msk in zip(images, masks)]
hp = HistogramPlotter(title="FCM Normalized")
_ = hp(images, masks)
plt.show()
```

3.4.3 Another Python API example (co-registration)

`intensity-normalization` relies on ANTsPy to do registration, so, for this example, you'll need to install ANTsPy first. You'll likely need to let it compile from source (~40 minutes) which requires CMake*[0].

Once you have ANTsPy installed, you can co-register an image like:

```
# load the images
import nibabel as nib
image = nib.load("path/to/image.nii")
target = nib.load("path/to/target.nii")

# setup up registration
from intensity_normalization.util.coregister import register
transformation = "Affine"
interpolator = "bSpline"
initial_rigid = True  # do initial rigid transformation before transformation

# verify this is a supported transformation, interpolator
from intensity_normalization.typing import (
    allowed_transformations, allowed_interpolators
)
assert transformation in allowed_transformations
assert interpolator in allowed_interpolators

# register the image to the target
registered = register(
    image,
    target,
    type_of_transform=transformation,
```

(continues on next page)

[0] If you're on a Mac, `brew install cmake` and then `pip install antspyx` in the environment you want to run `intensity-normalization` from or install `intensity-normalization` with `pip install "intensity-normalization[ants]"`

(continued from previous page)

```
    interpolator=interpolator,
    initial_rigid=initial_rigid
)

# save the image or get the registered image out
registered.to_filename("registered.nii")
registered_data = registered.get_fdata()
```

Alternatively, if you want to co-register many images to the same target, you can do:

```
# setup up registration
from intensity_normalization.util.coregister import Registrator
transformation = "Affine"
interpolator = "bSpline"
initial_rigid = True

registrator = Registrator(
    target,
    type_of_transform=transformation,
    interpolator=interpolator,
    initial_rigid=initial_rigid
)

registered = registrator(image)
registered.to_filename("registered.nii")
registered_data = registered.get_fdata()

# or if you have many images
images = [nib.load(path_to_image) for path_to_image in image_paths]
registered_images = registrator.register_images(images)
```

3.4.4 Saving fit information for sample-based methods

Fitting and using the resultant fit for new images is supported in the Python API. For example, you can run:

```
# load images
import nibabel as nib
image_paths = ["path/to/image1.nii", "path/to/image2.nii", ...]
images = [nib.load(image_path).get_fdata() for image_path in image_paths]

# normalize the images and save the standard histogram
from intensity_normalization.normalize.nyul import NyulNormalize
nyul_normalizer = NyulNormalize()
nyul_normalizer.fit(images)
normalized = [nyul_normalizer(image) for image in images]
nyul_normalizer.save_standard_histogram("standard_histogram.npy")

# load new images (of the same modality) and normalize those
new_image_paths = ["path/to/another/image1.nii", "path/to/another/image2.nii", ...]
new_images = [nib.load(image_path).get_fdata() for image_path in new_image_paths]
normalized = [nyul_normalizer(image) for image in images]
```

(continues on next page)

(continued from previous page)

```
# load the standard histogram
new_nyul_normalizer = NyulNormalize()
new_nyul_normalizer.load_standard_histogram("standard_histogram.npy")
normalized = [new_nyul_normalizer(image) for image in images]
```

For LSQ:

```
from intensity_normalization.normalize.lsq import LSQNormalize
lsq_normalizer = LSQNormalize()
lsq_normalizer.fit(images)
normalized = [lsq_normalizer(image) for image in images]
lsq_normalizer.save_standard_tissue_means("tissue_means.npy")

# reload the tissue means and use
lsq_normalizer = LSQNormalize()
lsq_normalizer.load_standard_tissue_means("tissue_means.npy")
normalized = [lsq_normalizer(image) for image in images]
```

RAVEL is only meant to work on a particular batch, so you need to refit it if you add new data to your batch or want to use it to normalize new data.

Similar options are added to the CLI. For `nyul-normalize` the relevant new options are `--save-standard-histogram` and `--load-standard-histogram`. For LSQ, `--save-standard-tissue-means` and `--load-standard-tissue-means`.

CHAPTER

FOUR

PYTHON API

4.1 intensity_normalization package

4.1.1 Subpackages

intensity_normalization.cli package

Submodules

intensity_normalization.cli.coregister module

Co-registration CLI Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.coregister.**coregister_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.fcm module

CLI for fuzzy c-means tissue mean normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.fcm.**fcm_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.histogram module

CLI for plotting histograms of a set of images Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.histogram.**histogram_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.kde module

Kernel density estimate tissue mode normalization CLI Author: Jacob Reinhold (jcreinhold@gmail.com) Created on: 13 Oct 2021

intensity_normalization.cli.kde.**kde_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.lsq module

Least-squares fit tissue mean normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.lsq.**lsq_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.nyul module

CLI for Nyul & Udupa normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.nyul.**nyul_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.preprocess module

CLI for MR image preprocessor Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.preprocess.**preprocess_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.ravel module

CLI for RAVEL normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.ravel.**ravel_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.tissue_membership module

CLI for calculating tissue membership image Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.tissue_membership.**tissue_membership_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.whitestripe module

CLI for WhiteStripe normalization method Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.whitestripe.**whitestripe_main**(*args: Namespace | list[str] | None = None*) → int

intensity_normalization.cli.zscore module

CLI for Z-Score normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 13 Oct 2021

intensity_normalization.cli.zscore.**zscore_main**(*args: Namespace | list[str] | None = None*) → int

Module contents

intensity_normalization.normalize package

Submodules

intensity_normalization.normalize.base module

Base class for normalization methods Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.base.**DirectoryNormalizeCLI**

Bases: SampleNormalizeCLIMixin, *DirectoryCLI*

before_fit(*images: collections.abc.Sequence[intnormt.ImageLike]*, */*, *masks: collections.abc.Sequence[intnormt.ImageLike] | None = None*, ***, *modality: intnormt.Modality = Modality.T1*, ***kwargs: Any*) → tuple[ImageSeq, MaskSeqOrNone]

fit(*images: Sequence[ImageLike]*, */*, *masks: Sequence[ImageLike] | None = None*, ***, *modality: Modality = Modality.T1*, ***kwargs: Any*) → None

fit_from_directories(*image_dir: intnormt.PathLike*, */*, *mask_dir: intnormt.PathLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*, *ext: str = 'nii*'*, *return_normalized_and_masks: bool = False*, ***kwargs: Any*) → tuple[ImageSeq, MaskSeqOrNone] | None

class intensity_normalization.normalize.base.**LocationScaleCLIMixin**(***, *norm_value: float = 1.0*, ***kwargs: Any*)

Bases: LocationScaleMixin, NormalizeCLIMixin

classmethod from_argparse_args(*args: Namespace*, */*) → T

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

class intensity_normalization.normalize.base.**SingleImageNormalizeCLI**

Bases: NormalizeCLIMixin, *SingleImageCLI*

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

plot_histogram_from_args(*args: argparse.Namespace*, */*, *normalized: intnormt.ImageLike*, *mask: intnormt.ImageLike | None = None*) → None

intensity_normalization.normalize.fcm module

Fuzzy C-Means-based tissue mean normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.fcm.**FCMNormalize**(*, *norm_value: float = 1.0*, *tissue_type: TissueType = TissueType.WM*, ***kwargs: Any*)

Bases: *LocationScaleCLIMixin*, *SingleImageNormalizeCLI*

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

calculate_location(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

calculate_scale(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

static description() → str

classmethod from_argparse_args(*args: Namespace*, */*) → *FCMNormalize*

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

property is_fit: bool

static name() → str

save_additional_info(*args: Namespace*, ***kwargs: Any*) → None

intensity_normalization.normalize.kde module

Kernel density estimation-based tissue mode normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.kde.**KDENormalize**(*norm_value: float = 1.0*, ***kwargs: Any*)

Bases: *LocationScaleCLIMixin*, *SingleImageNormalizeCLI*

calculate_location(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

calculate_scale(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

static description() → str

static fullname() → str

static name() → str

intensity_normalization.normalize.lsq module

Least-squares fit tissue means of a set of images Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.lsq.**LeastSquaresNormalize**(*, *norm_value: float = 1.0*, ***kwargs: Any*)

Bases: *LocationScaleCLIMixin*, *DirectoryNormalizeCLI*

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

calculate_location(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

calculate_scale(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

static description() → str

classmethod from_argparse_args(*args: Namespace*, */*) → *LeastSquaresNormalize*

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

load_standard_tissue_means(*filename: str | PathLike*, */*) → None

static name() → str

save_additional_info(*args: Namespace*, ***kwargs: Any*) → None

save_standard_tissue_means(*filename: str | PathLike*, */*) → None

scaling_factor(*tissue_means: ndarray[Any, dtype[ScalarType]]*) → float

static tissue_means(*image:* ImageLike, */*, *tissue_membership:* ImageLike) → ndarray[Any, dtype[ScalarType]]

intensity_normalization.normalize.nyul module

Nyul & Udupa piecewise linear histogram matching normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 02 Jun 2021

class intensity_normalization.normalize.nyul.**NyulNormalize**(*, *output_min_value: float = 1.0*, *output_max_value: float = 100.0*, *min_percentile: float = 1.0*, *max_percentile: float = 99.0*, *percentile_after_min: float = 10.0*, *percentile_before_max: float = 90.0*, *percentile_step: float = 10.0*)

Bases: *DirectoryNormalizeCLI*

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

static description() → str

classmethod from_argparse_args(*args: Namespace*, */*) → *NyulNormalize*

static fullname() → str

get_landmarks(*image:* ImageLike, */*) → ndarray[Any, dtype[ScalarType]]

load_standard_histogram(*filename: str | PathLike*) → None

static name() → str

normalize_image(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → intnormt.ImageLike

property percentiles: ndarray[Any, dtype[ScalarType]]

save_additional_info(*args: Namespace*, ***kwargs: Any*) → None

save_standard_histogram(*filename: str | PathLike*) → None

intensity_normalization.normalize.ravel module

RAVEL normalization (WhiteStripe then CSF correction) Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: Jun 02, 2021

class intensity_normalization.normalize.ravel.**RavelNormalize**(***, *membership_threshold: float = 0.99*, *register: bool = True*, *num_unwanted_factors: int = 1*, *sparse_svd: bool = False*, *whitestripe_kwargs: dict[str, Any] | None = None*, *quantile_to_label_csf: float = 1.0*, *masks_are_csf: bool = False*)

Bases: *DirectoryNormalizeCLI*

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

create_image_matrix_and_control_voxels(*images: collections.abc.Sequence[intnormt.ImageLike]*, */*, *masks: collections.abc.Sequence[intnormt.ImageLike] | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → tuple[npt.NDArray, npt.NDArray]

creates a matrix of images; rows correspond to voxels, columns are images

Parameters

- **images** – list of MR images of interest
- **masks** – list of corresponding brain masks
- **modality** – modality of the set of images (e.g., t1)

Returns

rows are voxels, columns are images control_voxels: rows are csf intersection voxels, columns are images

Return type
image_matrix

static description() → str

estimate_unwanted_factors(*control_voxels: ndarray[Any, dtype[ScalarType]]*) → ndarray[Any, dtype[ScalarType]]

classmethod from_argparse_args(*args: Namespace*, */*) → *RavelNormalize*

static fullname() → str

static name() → str

normalize_image(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → intnormt.ImageLike

process_directories(*image_dir: intnormt.PathLike*, */*, *mask_dir: intnormt.PathLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*, *ext: str = 'nii*'*, *return_normalized_and_masks: bool = False*, ***kwargs: Any*) → tuple[list[mioi.Image], list[mioi.Image] | None] | None

save_additional_info(*args: Namespace*, ***kwargs: Any*) → None

set_template(*template: intnormt.ImageLike | ants.ANTsImage*) → None

set_template_mask(*template_mask: intnormt.ImageLike | ants.ANTsImage | None*) → None

teardown() → None

property template: ants.ANTsImage | None

property template_mask: ants.ANTsImage | None

use_mni_as_template() → None

intensity_normalization.normalize.whitestripe module

WhiteStripe (normal-appearing white matter mean & standard deviation) normalization Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.whitestripe.**WhiteStripeNormalize**(***, *norm_value: float = 1.0*, *width: float = 0.05*, *width_l: float | None = None*, *width_u: float | None = None*, ***kwargs: Any*)

Bases: *LocationScaleCLIMixin*, *SingleImageNormalizeCLI*

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

calculate_location(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

calculate_scale(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

static description() → str

classmethod from_argparse_args(*args: Namespace*, */*) → *WhiteStripeNormalize*

static fullname() → str

static name() → str

plot_histogram_from_args(*args: argparse.Namespace*, */*, *normalized: intnormt.ImageLike*, *mask: intnormt.ImageLike | None = None*) → None

setup(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → None

teardown() → None

intensity_normalization.normalize.zscore module

Z-score normalize image (voxel-wise subtract mean, divide by standard deviation) Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.normalize.zscore.**ZScoreNormalize**(***, *norm_value: float = 1.0*, ***kwargs: Any*)

Bases: *LocationScaleCLIMixin*, *SingleImageNormalizeCLI*

calculate_location(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

calculate_scale(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → float

static description() → str

static fullname() → str

static name() → str

plot_histogram_from_args(*args: argparse.Namespace*, */*, *normalized: intnormt.ImageLike*, *mask: intnormt.ImageLike | None = None*) → None

setup(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *modality: intnormt.Modality = Modality.T1*) → None

teardown() → None

Module contents

intensity_normalization.plot package

Submodules

intensity_normalization.plot.histogram module

Plot histogram of the intensities of a set of images Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 02 Jun 2021

class intensity_normalization.plot.histogram.**HistogramPlotter**(*, *figsize: tuple[int, int] = (12, 10)*, *alpha: float = 0.8*, *title: str | None = None*)

Bases: *DirectoryCLI*

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

static description() → str

classmethod from_argparse_args(*args: Namespace*) → *HistogramPlotter*

from_directories(*image_dir: intnormt.PathLike*, */*, *mask_dir: intnormt.PathLike | None = None*, ***, *ext: str = 'nii*'*, *exclude: collections.abc.Sequence[str] = ('membership',)*, ***kwargs: Any*) → plt.Axes

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

static name() → str

plot_all_histograms(*images: collections.abc.Sequence[intnormt.ImageLike]*, */*, *masks: collections.abc.Sequence[intnormt.ImageLike] | None*, ***kwargs: Any*) → plt.Axes

intensity_normalization.plot.histogram.**plot_histogram**(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *ax: plt.Axes | None = None*, *n_bins: int = 200*, *log: bool = True*, *alpha: float = 0.8*, *linewidth: float = 3.0*, ***kwargs: Any*) → plt.Axes

plots the histogram of the intensities of a numpy array within a given brain mask or estimated foreground mask (the estimate is just all intensities above the mean)

Parameters

- **image** – image/array of interest
- **mask** – mask of the foreground, if none then assume skull-stripped
- **ax** – matplotlib ax to plot on, if none then create new ax
- **n_bins** – number of bins to use in histogram
- **log** – use log scale on the y-axis
- **alpha** – value in [0,1], controls opacity of line plot
- **linewidth** – width of line in histogram plot
- **kwargs** – arguments to the histogram function

Returns
the ax the histogram is plotted on

Return type
ax

Module contents

intensity_normalization.util package

Submodules

intensity_normalization.util.coregister module

Co-register images with ANTsPy Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 03 Jun 2021

class intensity_normalization.util.coregister.**Registrator**(*template: nib.nifti1.Nifti1Image | ants.ANTsImage = None*, ***, *type_of_transform: str = 'Affine'*, *interpolator: str = 'bSpline'*, *metric: str = 'mattes'*, *initial_rigid: bool = True*)

Bases: *SingleImageCLI*

static description() → str

classmethod from_argparse_args(*args: Namespace*) → *Registrator*

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

static load_image(*image_path: str | PathLike*) → ants.ANTsImage

static name() → str

register_images(*images: collections.abc.Sequence[nib.nifti1.Nifti1Image | ants.ANTsImage]*, */*) → collections.abc.Sequence[nib.nifti1.Nifti1Image | ants.ANTsImage]

register_images_to_templates(*images: collections.abc.Sequence[nib.nifti1.Nifti1Image | ants.ANTsImage]*, */*, ***, *templates: collections.abc.Sequence[nib.nifti1.Nifti1Image | ants.ANTsImage]*) → collections.abc.Sequence[nib.nifti1.Nifti1Image | ants.ANTsImage]

intensity_normalization.util.coregister.**register**(*image: ValidImage*, */*, *template: Optional[ValidImage] = None*, ***, *type_of_transform: str = 'Affine'*, *interpolator: str = 'bSpline'*, *metric: str = 'mattes'*, *initial_rigid: bool = True*, *template_mask: Optional[ValidImage] = None*) → nib.nifti1.Nifti1Image | ants.ANTsImage

intensity_normalization.util.histogram_tools module

Process the histograms of MR (brain) images Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

intensity_normalization.util.histogram_tools.**get_first_tissue_mode**(*image: ImageLike*, */*, ***, *remove_tail: bool = True*, *tail_percentage: float = 99.0*) → float

Mode of the lowest-intensity tissue class

Parameters

- **image** – array of image data (like an np.ndarray)
- **remove_tail** – remove tail from histogram
- **tail_percentage** – if remove_tail, use the histogram below this percentage

Returns
mode of the lowest-intensity tissue class

Return type
first_tissue_mode

intensity_normalization.util.histogram_tools.**get_largest_tissue_mode**(*image:* ImageLike, */*) → float

Mode of the largest tissue class

Parameters
image – array of image data (like an np.ndarray)

Returns
value of the largest tissue mode

Return type
largest_tissue_mode

intensity_normalization.util.histogram_tools.**get_last_tissue_mode**(*image:* ImageLike, */*, ***, *remove_tail: bool = True*, *tail_percentage: float = 96.0*) → float

Mode of the highest-intensity tissue class

Parameters

- **image** – array of image data (like an np.ndarray)
- **remove_tail** – remove tail from histogram
- **tail_percentage** – if remove_tail, use the histogram below this percentage

Returns
mode of the highest-intensity tissue class

Return type
last_tissue_mode

intensity_normalization.util.histogram_tools.**get_tissue_mode**(*image:* ImageLike, */*, ***, *modality:* Modality) → float

Find the appropriate tissue mode given a modality

intensity_normalization.util.histogram_tools.**smooth_histogram**(*image:* ImageLike, */*) → tuple[*intensity_normalization.typing.ImageLike*, *intensity_normalization.typing.ImageLike*]

Use kernel density estimate to get smooth histogram

Parameters
image – array of image data (like an np.ndarray)

Returns
domain of the pdf pdf: kernel density estimate of the pdf of data

Return type
grid

intensity_normalization.util.io module

Input/output utilities for the project Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

intensity_normalization.util.io.**gather_images**(*dirpath: str | PathLike*, ***, *ext: str = 'nii*'*, *exclude: Sequence[str] = ()*) → list[pymedio.image.Image]

return all images of extension *ext* from a directory

intensity_normalization.util.io.**gather_images_and_masks**(*image_dir: intnormt.PathLike*, *mask_dir: intnormt.PathLike | None = None*, ***, *ext: str = 'nii*'*, *exclude: collections.abc.Sequence[str] = ()*) → tuple[PymedioImageList, PymedioMaskListOrNone]

intensity_normalization.util.io.**glob_ext**(*dirpath: str | PathLike*, ***, *ext: str = 'nii*'*, *exclude: Sequence[str] = ()*) → list[pathlib.Path]

return a sorted list of ext files for a given directory path

intensity_normalization.util.io.**split_filename**(*filepath: str | PathLike*, */*, ***, *resolve: bool = False*) → *SplitFilename*

split a filepath into the directory, base, and extension .. rubric:: Examples

```
>>> split_filename("path/base.ext")
SplitFilename(path=PosixPath('path'), base='base', ext='.ext')
```

intensity_normalization.util.io.**zip_with_nones**(**args: Sequence[Any] | None*) → Zipped

zip sequence args but if an arg is None, yield None in that argument index .. rubric:: Examples

```
>>> for x, y, z in zip_with_nones((1, 2), None, ("a", "b")):
...     print(x, y, z)
1 None a
2 None b
```

intensity_normalization.util.preprocess module

Preprocess MR images for image processing

Preprocess MR images according to a simple scheme: 1) N4 bias field correction 2) resample to X mm x Y mm x Z mm 3) reorient images to specification

Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 21 May 2018

class intensity_normalization.util.preprocess.**Preprocessor**(***, *resolution: tuple[float, ...] | None = None*, *orientation: str = 'RAI'*, *n4_convergence_options: dict[str, Any] | None = None*, *interp_type: str = 'linear'*, *second_n4_with_smoothed_mask: bool = True*)

Bases: *SingleImageCLI*

static description() → str

classmethod from_argparse_args(*args: Namespace*) → *Preprocessor*

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

static load_image(*image_path: str | PathLike*) → ants.ANTsImage

static name() → str

intensity_normalization.util.preprocess.**preprocess**(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *resolution: tuple[float, ...] | None = None*, *orientation: str = 'RAS'*, *n4_convergence_options: dict[str, Any] | None = None*, *interp_type: str = 'linear'*, *second_n4_with_smoothed_mask: bool = True*) → tuple[mioi.Image, mioi.Image]

Preprocess an MR image

Preprocess an MR image according to a simple scheme: 1) N4 bias field correction 2) resample to X mm x Y mm x ... 3) reorient images to RAI

Parameters

- **image** – image to preprocess
- **mask** – mask covering the brain of image (none if already skull-stripped)
- **resolution** – resolution for resampling. None for no resampling.
- **orientation** – reorient the image according to this. See ANTsPy for details.
- **n4_convergence_options** – n4 processing options. See ANTsPy for details.
- **interp_type** – interpolation type for resampling choices: linear, nearest_neighbor, gaussian, windowed_sinc, bspline
- **second_n4_with_smoothed_mask** – do a second N4 with a smoothed mask often improves the bias field correction in the image

Returns

preprocessed image and corresponding foreground mask

intensity_normalization.util.tissue_membership module

Find the tissue-membership of a T1-w brain image Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.util.tissue_membership.**TissueMembershipFinder**(*hard_segmentation: bool = False*)

Bases: *SingleImageCLI*

static description() → str

classmethod from_argparse_args(*args: Namespace*) → *TissueMembershipFinder*

static fullname() → str

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

static name() → str

intensity_normalization.util.tissue_membership.**find_tissue_memberships**(*image: intnormt.ImageLike*, */*, *mask: intnormt.ImageLike | None = None*, ***, *hard_segmentation: bool = False*, *n_classes: int = 3*) → mioi.Image

Tissue memberships for a T1-w brain image with fuzzy c-means

Parameters

- **image** – image to find tissue masks for (must be T1-w)
- **mask** – mask covering the brain of image (none if already skull-stripped)
- **hard_segmentation** – pick the maximum membership as the true class in output
- **n_classes** – number of classes (usually three for CSF, GM, WM)

Returns

membership values for each of three classes in the image
(or class determinations w/ hard_seg)

Return type
tissue_mask

Module contents

4.1.2 Submodules

4.1.3 intensity_normalization.base_cli module

CLI base class for normalization/preprocessing methods Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 06 Jun 2021

class intensity_normalization.base_cli.**CLIMixin**

Bases: object

static add_method_specific_arguments(*parent_parser: ArgumentParser*) → ArgumentParser

append_name_to_file(*filepath: intnormt.PathLike*, *alternate_path: intnormt.PathLike | None = None*) → pathlib.Path

abstract call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

abstract static description() → str

abstract classmethod from_argparse_args(*args: Namespace*) → T

abstract static fullname() → str

abstract classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

static load_image(*image_path: str | PathLike*) → pymedio.image.Image

classmethod main(*parser: ArgumentParser*) → Callable[[Namespace | list[str] | None], int]

abstract static name() → str

classmethod parser() → ArgumentParser

class intensity_normalization.base_cli.**DirectoryCLI**

Bases: *CLIMixin*

abstract call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

class intensity_normalization.base_cli.**SingleImageCLI**

Bases: *CLIMixin*

call_from_argparse_args(*args: Namespace*, */*, ***kwargs: Any*) → None

classmethod get_parent_parser(*desc: str*, *valid_modalities: frozenset[str] = frozenset({'flair', 'md', 'other', 'pd', 't1', 't2'})*, ***kwargs: Any*) → ArgumentParser

intensity_normalization.base_cli.**setup_log**(*verbosity: int*) → None

set logger with verbosity logging level and message

4.1.4 intensity_normalization.errors module

Project-specific errors Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 24 Apr 2018

exception intensity_normalization.errors.**NormalizationError**

Bases: RuntimeError

4.1.5 intensity_normalization.typing module

Project-specific types Author: Jacob Reinhold <jcreinhold@gmail.com> Created on: 01 Jun 2021

class intensity_normalization.typing.**ImageLike**(**args*, ***kwargs*)

Bases: Protocol[S_co, T_co, U_co]

support anything that implements the methods here

any(*axis: int | tuple[int, ...] | None = None*) → Any

flatten() → T_co

mean() → float

min() → float

property ndim: integer[NBit] | int

nonzero() → Any

reshape(**shape: SupportsIndex, order: Literal['A', 'C', 'F'] | None = Ellipsis*) → T_co

property shape: tuple[int, ...]

squeeze() → Any

std() → float

sum() → Float | Int

transpose(**axes: int*) → T_co

class intensity_normalization.typing.**Modality**(*value*)

Bases: Enum

An enumeration.

FLAIR: str = 'flair'

MD: str = 'md'

OTHER: str = 'other'

PD: str = 'pd'

T1: str = 't1'

T2: str = 't2'

classmethod from_string(*string: str* | Modality) → *Modality*

class intensity_normalization.typing.**SplitFilename**(*path*, *base*, *ext*)

Bases: NamedTuple

base: str

Alias for field number 1

ext: str

Alias for field number 2

path: Path

Alias for field number 0

class intensity_normalization.typing.**TissueType**(*value*)

Bases: Enum

An enumeration.

CSF: str = 'csf'

GM: str = 'gm'

WM: str = 'wm'

classmethod from_string(*string: str*) → *TissueType*

to_fullname() → str

to_int() → int

class intensity_normalization.typing.**dir_path**

Bases: _ParseType

class intensity_normalization.typing.**file_path**

Bases: _ParseType

intensity_normalization.typing.**new_parse_type**(*func: Callable[[Any], Any]*, *name: str*) → NewParseType

class intensity_normalization.typing.**nonnegative_float**

Bases: _ParseType

class intensity_normalization.typing.**nonnegative_int**

Bases: _ParseType

class intensity_normalization.typing.**positive_float**

Bases: _ParseType

class intensity_normalization.typing.**positive_int**

Bases: _ParseType

class intensity_normalization.typing.**positive_int_or_none**

Bases: _ParseType

class intensity_normalization.typing.**positive_odd_int_or_none**

Bases: _ParseType

class intensity_normalization.typing.**probability_float**

Bases: _ParseType

class intensity_normalization.typing.**probability_float_or_none**

Bases: _ParseType

class intensity_normalization.typing.**save_file_path**

Bases: _ParseType

4.1.6 Module contents

Top-level package for intensity-normalization.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/jcreinhold/intensity-normalization/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

intensity-normalization could always use more documentation, whether as part of the official intensity-normalization docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jcreinhold/intensity-normalization/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up `intensity-normalization` for local development.

1. Fork the `intensity-normalization` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/intensity-normalization.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv intensity-normalization
$ cd intensity-normalization/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 intensity_normalization tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7, 3.8, and 3.9.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_intensity_normalization
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Jacob Reinhold <jcreinhold@gmail.com>

6.2 Contributors

- Sarthak Pati
- Jakub Kaczmarzyk

CHAPTER

SEVEN

HISTORY

7.1 2.2.4 (2023-05-31)

- Update to allow Python >=3.9

7.2 2.2.3 (2022-03-15)

- Revert error on different image shapes from `RavelNormalize`; it is required!

7.3 2.2.2 (2022-03-15)

- Remove plural from `Modality` and `TissueType` enumerations.
- Update tutorials to use the `Modality` and `TissueType` enumerations.
- Remove error on different image shapes from `RavelNormalize` when registration enabled.

7.4 2.2.1 (2022-03-14)

- Update documentation to support modifications to Python API
- Update dependencies
- Remove incorrect warning from WhiteStripe normalization

7.5 2.2.0 (2022-02-25)

- Change backend to `pymedio` to support more medical image formats

7.6 2.1.4 (2022-01-17)

- Fix testing bugs in 2.1.3 and cleanup some interfaces

7.7 2.1.3 (2022-01-17)

- Cleanup Makefile and dependencies
- Add py.typed file

7.8 2.1.2 (2022-01-03)

- Updates for security

7.9 2.1.1 (2021-10-20)

- Fix warning about negative values when not passing in a mask
- Remove redundant word from Nyul normalize keyword arguments

7.10 2.1.0 (2021-10-13)

- Restructure CLIs for faster startup and improve handling of missing antspy
- add option to CLIs to print version

7.11 2.0.3 (2021-10-11)

- Improve Nyul docs and argument/keyword names

7.12 2.0.2 (2021-09-27)

- Fix and improve documentation
- Add an escape-hatch “other” modality
- Add peak types as modalities in KDE & WS

7.13 2.0.1 (2021-08-31)

- Save and load fit artifacts for LSQ and Nyul for both the CLIs and Python API

7.14 2.0.0 (2021-08-22)

- Major refactor to reduce redundancy, make code more usable outside of the CLIs, and generally improve code quality.

7.15 1.4.0 (2021-03-16)

- First release on PyPI.

CHAPTER

EIGHT

ALGORITHM DESCRIPTIONS

For all the algorithm descriptions, let $I(\mathbf x)$ be the MR brain image under consideration where $\mathbf x \in [0,N]\times[0,M]\times[0,L]\subset \mathbb{N}^3$ for $N,M,L \in \mathbb{N}$, the dimensions of I, and let $B \subset I$ be the corresponding brain mask (i.e., the set of indices corresponding to the location of the brain in I).

If any of the descriptions here are unclear, please see the corresponding paper for a more concise, refined description.

8.1 Z-score

Z-score normalization uses the brain mask B for the image I to determine the mean and standard deviation of the intensities inside the brain mask, that is: $$ \mu = \frac{1}{|B|} \sum_{\mathbf b \in B} I(\mathbf b) \quad \text{and} \quad \sigma = \sqrt{\frac{\sum_{\mathbf b \in B} (I(\mathbf b) - \mu)^2}{|B|-1}} $$ Then the Z-score normalized image is $$ I_{\text{z-score}}(\mathbf x) = \frac{I(\mathbf x) - \mu}{\sigma}. $$

8.2 Fuzzy C-Means

Fuzzy C-means normalization uses a segmentation of a specified tissue (i.e., CSF, GM, or WM) to normalize the entire image to the mean of the tissue. The procedure is as follows. Let $T\subset B$ be the tissue mask for the image I, i.e., T is the set of indices corresponding to the location of the tissue in the image I. Then the tissue mean is $$ \mu = \frac{1}{|T|} \sum_{\mathbf t \in T} I(\mathbf t) $$ and the segmentation-based normalized image is $$ I_{\text{seg}}(\mathbf x) = \frac{c\cdot I(\mathbf x)}{\mu} $$ where $c \in \mathbb{R}_{>0}$ is some constant. In this function, we use three-class fuzzy c-means to get a segmentation of the tissue over the brain mask B for the T1-w image and we arbitrarily set $c = 1$.

8.3 Kernel Density Estimation

KDE-based normalization estimates the empirical probability density function (pdf) of the intensities of I over the brain mask B using the method of kernel density estimation.

The KDE of the pdf for the intensity of the image is calculated as follows: $$ \hat{p}(x) = \frac{1}{N\cdot M \cdot L \cdot \delta} \sum_{i = 1}^{N\cdot M \cdot L} K\left(\frac{x - x_i}{\delta}\right)$$ where x is an intensity value, K is the kernel (a kernel is essentially a non-negative function that integrates to 1), δ is the bandwidth parameter (that is, a smoothing parameter) which scales the kernel K.

In our experiment, we use a Gaussian kernel and set $\delta = 80$, which was found to empirically determine a reasonable density estimate across various datasets.

The kernel density estimate provides a smooth version of the histogram which allows us to more robustly pick a the **mode** associated with the WM via a combinatorial optimization routine. The found WM peak p is then

used to normalize the entire image, in much the same way as the segmentation-based normalization. That is, $$ I_{\text{kde}}(\mathbf x) = \frac{c \cdot I(\mathbf x)}{p} $$ where $c \in \mathbb{R}_{>0}$ is some constant. In this package, we arbitrarily set $c = 1$.

8.4 Piecewise Linear Histogram Matching (Nyul & Udupa)

Piecewise (affine) histogram-based normalization (which we will denote as HM for brevity)—proposed by Nyul and Udupa[1]—addresses the normalization problem by learning a standard histogram for a set of contrast images and mapping the intensities of each image to this standard histogram. The standard histogram is learned through the demarcation of pre-defined landmarks of interest.

For instance, Shah et al.[2] defined landmarks as intensity percentiles at $1,10,20,\ldots,90,99$ percent (where the intensity values below 1% and above 99% are discarded as outliers). We use these landmarks as the default in our implementation. The standard scale must be have a pre-defined range, i.e., $[m_{\text{min}}^s, m_{\text{max}}^s]$. In our experiment, we arbitrarily set $m_{\text{min}}^s = 0$ and $m_{\text{max}}^s = 100$.

8.4.1 Learning the standard histogram

Let $\mathbf I = \{I_1,I_2,\ldots,I_K\}$ be a set of K MR brain images of one contrast. We calculate the following set of quantities m_1^i and m_{99}^i, which are the 1% and 99% intensity values for the image $I_i \in \mathbf I$. We then map all the intensity values of I_i with the following linear map $$ \tilde I_i(\mathbf x) = \left(I_i(\mathbf x) - m_1^i + m_{\text{min}}^s\right) \left(\frac{m_{\text{max}}^s}{m_{99}^i}\right) $$ which takes the intensities of I_i to the range $[m_{\text{min}}^s, m_{\text{max}}^s]$ excluding outliers. Then we calculate the deciles for the new image $\tilde I_i$, i.e., the set $\{\tilde m_{10}^i,\tilde m_{20}^i,\ldots,\tilde m_{90}^i\}$ (note that $\tilde m_0^i = m_{\text{min}}^s$ and $\tilde m_{100}^i = m_{\text{max}}^s$). This is done over every image $I_i \in \mathbf I$ and the mean of each corresponding value is the learned landmark for the standard histogram. That is, for $n \in \{10,20,\ldots,90\}$, we have $$ m_n^s = \frac{1}{K} \sum_{i=1}^{K} \tilde m_n^i $$ and the standard scale landarks is the set $\{m_{\text{min}}^s,m_{10}^s,\ldots,m_{90}^s,m_{\text{max}}^s\}$.

8.4.2 Normalizing new images

For a test image I, the transform for the normalization is done by first calculating the set of percentiles $\{m_{1},m_{10},m_{20},\ldots,m_{90},m_{99}\}$. These values are then used to segment the image into deciles, i.e., we define 10 non-overlapping sets of indices $D_{i,j} = \{\mathbf x \mid m_i \le I(\mathbf x) < m_j\}$ where $i,j \in \{1,10,20,\ldots,90,99\}$ and restricting j to equal the next value in the set greater than i. We then piecewise linearly map the intensities associated with these deciles to the corresponding decile on the standard scale landmarks. Noting that each $D_{i,j}$ is disjoint from the other, the normalized image is then defined as $$ I_{\text{nu}} = \bigcup_{\substack{i,j \in \{1,10,20,\ldots,90,99\}\newline i\neq j, i \le j + 10}} \left(\frac{I(D_{i,j}) - m_i}{m_j - m_i}\right) \left(m_j^s - m_i^s\right) + m_i^s. $$

[1] L. G. Nyúl, J. K. Udupa, and X. Zhang, "New Variants of a Method of MRI Scale Standardization," IEEE Trans. Med. Imaging, vol. 19, no. 2, pp. 143–150, 2000.

[2] M. Shah, Y. Xiao, N. Subbanna, S. Francis, D. L. Arnold, D. L. Collins, and T. Arbel, "Evaluating intensity normalization on MRIs of human brain with multiple sclerosis," Med. Image Anal., vol. 15, no. 2, pp. 267–282, 2011.

8.5 WhiteStripe

WhiteStripe intensity normalization[3] attempts to do a Z-score normalization based on the intensity values of normal appearing white matter (NAWM). The NAWM is found by smoothing the histogram of the image (i.e., KDE) and selecting the mode of the distribution (for T1-w images). Let p be the intensity associated with the mode. The "white stripe" is then defined as the 10% segment of intensity values around μ. That is, let $F(x)$ be the cdf of the specific MR image $I(\mathbf x)$ inside its brain mask B, and define $\tau = 5\%$. Then, the white stripe Ω_τ is defined as the set $$ \Omega_\tau = \left\{I(\mathbf x) \mid F^{-1}\left(F(\mu) - \tau\right) < I(\mathbf x) < F^{-1}\left(F(\mu) + \tau\right)\right\}. $$ Let σ be the sample standard deviation associated with Ω_τ. Then the WhiteStripe normalized image is $$ I_{\text{ws}}(\mathbf x) = \frac{I(\mathbf x) - \mu}{\sigma}. $$

8.6 RAVEL

RAVEL normalization[4] attempts to improve upon the result of WhiteStripe by removing unwanted technical variation, e.g., scanner effects. RAVEL assumes the set of images can be expressed in the additive model $$ V = \alpha 1^T + \beta X^T + \gamma Z^T + R $$ where V is a population of WhiteStripe normalized images of the same contrast, $\alpha 1^T$ is the average scan, βX^T represents known clinical covariates (e.g., age, gender), γZ^T represents the unknown, unwanted factors (i.e., the technical variability), and R is the matrix of residuals.

Since this model is assumed, if we can determine voxels in the MR image where there are no clinical covariates, then we can solve for the unwanted factors βX^T through simple linear regression. The authors, Fortin et al., assume that CSF is not associated with these clinical covariates and uses the voxels associated with CSF as the control voxels. Then if the average scan is removed, the voxels associated with the CSF is of the form $$ V_c = \gamma Z^T + R $$ where V_c are the set of control (CSF) voxels.

Note that we can rewrite V_c as $$ V_c = U \Sigma W^T $$ through the SVD. If W is an $n\times n$ matrix of right singular vectors. Then we can use $b<n$ right singular vectors to form an orthogonal basis for the unwanted factors Z[5]. That is, we use W_b as the estimate of Z, where W_b are the select b right singular vectors. We then do voxel-wise linear regression to estimate the coefficients γ. Then the RAVEL normalized image is simply $$ I_{\text{ravel}}(\mathbf x) = I_{\text{ws}}(\mathbf x) - \gamma_{\mathbf x} Z^T. $$ where $\gamma_{\mathbf x}$ are the coefficients of unwanted variation associated with the voxel $\mathbf x$ found via linear regression. In our experiments, we follow the original paper[4] and set $b=1$ to be the first singular vector (the first right singular vector is highly correlated (>95%) with the mean intensity of the CSF).

[3] R. T. Shinohara, E. M. Sweeney, J. Goldsmith, N. Shiee, F. J. Mateen, P. A. Calabresi, S. Jarso, D. L. Pham, D. S. Reich, and C. M. Crainiceanu, "Statistical normalization techniques for magnetic resonance imaging," NeuroImage Clin., vol. 6, pp. 9–19, 2014.

[4] J. P. Fortin, E. M. Sweeney, J. Muschelli, C. M. Crainiceanu, and R. T. Shinohara, "Removing inter-subject technical variability in magnetic resonance imaging studies," Neuroimage, vol. 132, pp. 198–212, 2016.

[5] J. T. Leek and J. D. Storey, "Capturing heterogeneity in gene expression studies by surrogate variable analysis," PLoS Genet., vol. 3, no. 9, pp. 1724–1735, 2007.

8.7 References

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Z