GETTING STARTED

1 Features 3
2 Contributors 5
3 Contact 7

3.1 Installation ................................................................. 7
3.2 Quick Start ................................................................. 10
3.3 Image Registration with Deep Learning ............................... 11
3.4 Design Experiments ..................................................... 13
3.5 Running DeepReg Remotely (on the Cluster) ......................... 15
3.6 Introduction to DeepReg Demos ..................................... 16
3.7 Paired Images ............................................................. 16
3.8 Unpaired Images .......................................................... 25
3.9 Grouped Images ........................................................... 37
3.10 Classical Registration .................................................. 44
3.11 Command Line Tools ................................................... 50
3.12 Configuration File ....................................................... 56
3.13 Dataset Loader ............................................................ 66
3.14 Registry ................................................................. 76
3.15 Experimental Features ................................................. 76
3.16 Entry Point ............................................................... 77
3.17 Dataset Loader ............................................................ 80
3.18 File Loader ............................................................... 83
3.19 Network ................................................................. 86
3.20 Network Backbone ..................................................... 86
3.21 Layer ................................................................. 89
3.22 Loss ................................................................. 101
3.23 Optimizer .............................................................. 108
3.24 Guidelines .............................................................. 109
3.25 Unit Test .............................................................. 110
3.26 DeepReg demo ......................................................... 111
3.27 Documentation ......................................................... 113
3.28 Release .............................................................. 113

Python Module Index 115
Index 117
DeepReg is a freely available, community-supported open-source toolkit for research and education in medical image registration using deep learning.

The current version is implemented as a TensorFlow2-based framework, and contains implementations for unsupervised- and weakly-supervised algorithms with their combinations and variants. DeepReg has a practical focus on growing and diverse clinical applications, as seen in the provided examples - DeepReg Demos.

Get involved and help make DeepReg better!
DeepReg extends and simplifies workflows for medical imaging researchers working in TensorFlow 2, and can be easily installed and used for efficient training and rapid deployment of deep-learning registration algorithms. DeepReg is designed to be used with minimal programming or scripting, owing to its built-in command line tools. Our development and all related work involved in the project is public, and released under the Apache 2.0 license.
DeepReg is maintained and led by a team of developers and researchers. People with significant contributions to DeepReg are listed below (in alphabetical order).

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation (at time of contribution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adrià Casamitjana</td>
<td>University College London</td>
</tr>
<tr>
<td>Alexander Grimwood</td>
<td>University College London</td>
</tr>
<tr>
<td>Daniel C. Alexander</td>
<td>University College London</td>
</tr>
<tr>
<td>Dean C. Barratt</td>
<td>University College London</td>
</tr>
<tr>
<td>Ester Bonmati</td>
<td>University College London</td>
</tr>
<tr>
<td>Juan Eugenio Iglesias</td>
<td>University College London / Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>Matt J. Clarkson</td>
<td>University College London</td>
</tr>
<tr>
<td>Nina Montaña Brown</td>
<td>University College London</td>
</tr>
<tr>
<td>Qianye Yang</td>
<td>University College London</td>
</tr>
<tr>
<td>Rémi Delaunay</td>
<td>University College London / King’s College London</td>
</tr>
<tr>
<td>Shaheer U. Saeed</td>
<td>University College London</td>
</tr>
<tr>
<td>Stefano B. Blumberg</td>
<td>University College London</td>
</tr>
<tr>
<td>Tom Vercauteren</td>
<td>King’s College London</td>
</tr>
<tr>
<td>Yipeng Hu</td>
<td>University College London</td>
</tr>
<tr>
<td>Yunguan Fu</td>
<td>University College London / InstaDeep</td>
</tr>
<tr>
<td>Zachary M. C. Baum</td>
<td>University College London</td>
</tr>
<tr>
<td>Zhe Min</td>
<td>University College London</td>
</tr>
</tbody>
</table>

This open-source initiative started within University College London, with support from the Wellcome/EPSRC Centre for Interventional and Surgical Sciences (WEISS), and partial support from the Wellcome/EPSRC Centre for Medical Engineering (CME).
Chapter 2. Contributors
For development matters, please raise an issue.

For matters regarding the Code of Conduct, such as a complaint, please email the DeepReg Development Team: DeepRegNet@gmail.com.

Alternatively, please contact one or more members of the CoC Committee as appropriate: Nina Montana Brown (nina.brown.15@ucl.ac.uk), Ester Bonmati (e.bonmati@ucl.ac.uk), Matt Clarkson (m.clarkson@ucl.ac.uk).

3.1 Installation

DeepReg uses in Python 3.7 and external python dependencies are defined in requirements. DeepReg primarily supports and is regularly tested with Ubuntu and Debian Linux distributions.

There are multiple different methods to install DeepReg:

1. Clone DeepReg and create a virtual environment using Anaconda / Miniconda (recommended).
2. Clone DeepReg and build a docker image using the provided docker file.
3. Install directly from PyPI release without cloning DeepReg.

3.1.1 Install via Conda

The recommended method is to install DeepReg in a dedicated virtual environment using Anaconda / Miniconda.

Please clone DeepReg first and change current directory to the DeepReg root directory:

```
$ cd DeepReg
```

Then, install or update the conda environment following the instructions below. Please see the official conda documentation for more details.

Linux

Mac OS
DeepReg

Windows

Install DeepReg without GPU support.

```bash
conda env create -f environment_cpu.yml
conda activate deepreg
```

Install DeepReg with GPU support.

```bash
conda env create -f environment.yml
conda activate deepreg
```

Update DeepReg without GPU support.

```bash
conda env update -f environment_cpu.yml
```

Update DeepReg with GPU support.

```bash
conda env update -f environment.yml
```

Install DeepReg without GPU support.

```bash
conda env create -f environment_cpu.yml
conda activate deepreg
```

Update DeepReg without GPU support.

```bash
conda env update -f environment_cpu.yml
```

Install/update DeepReg with GPU support.

**Warning:** Not supported or tested.

Install/update DeepReg without GPU support.

**Warning:** DeepReg on Windows is not fully supported. However, you can use the Windows Subsystem for Linux. Set up WSL and follow the DeepReg setup instructions for Linux.

Install/update DeepReg with GPU support.

**Warning:** Not supported or tested.

After activating the conda environment, please install DeepReg locally:

```bash
pip install -e .
```
3.1.2 Install via docker

We also provide the docker file for building the docker image. Please clone DeepReg repository first:

```
```

Then, install DeepReg following the instructions below.

**Install docker**

Docker can be installed following the official documentation.

For Linux based OS, there are some additional setup after the installation. Otherwise you might have permission errors.

**Build docker image**

```
docker build . -t deepreg -f Dockerfile
```

where

- `-t` names the built image as `deepreg`.
- `-f` provides the docker file for configuration.

**Create a container**

```
docker run --name <container_name> --privileged=true -ti deepreg bash
```

where `-name <container_name>` names the created container. `-privileged=true` is required to solve the permission issue linked to TensorFlow profiler. `-it` allows interaction with container and enters the container directly, check more info on stackoverflow.

**Remove a container**

```
docker rm -v <container_name>
```

which removes a created container and its volumes, check more info on docker documentation.

3.1.3 Install via PyPI

Please use the following command to install DeepReg directly from the PyPI release:

```
pip install deepreg
```

The PyPI release currently does not ship with test data and demos. Running examples, such as those in Quick Start and DeepReg Demo, in this documentation may require downloading additional test data.

Once you have installed DeepReg via `pip`, you can run the following command to download the necessary files to run all examples by:
deepreg_download

The above will download the files to the current working directory. If you need to download to a specific directory, use the `--output_dir` or `-d` flag to specify this.

**Note**

1. All dependencies, APIs and command-line tools will be installed automatically via each installation method.
2. Only released versions of DeepReg are available via PyPI release. Therefore it is different from the latest (unstable) version on GitHub.

### 3.2 Quick Start

This is a set of simple tests to use DeepReg command line tools. More details and other options can be found in Command Line Tools.

First, install DeepReg and change current directory to the root directory of DeepReg.

#### 3.2.1 Train a registration network

Train a registration network using unpaired and labeled example data with a predefined configuration:

```
depreg_train --gpu "" --config_path config/unpaired_labeled_ddf.yaml --log_dir test
```

where:

- `--gpu ""` indicates using CPU. Change to `--gpu "0"` to use the GPU at index 0.
- `--config_path <filepath>` specifies the configuration file path.
- `--log_dir test` specifies the output folder. In this case, the output is saved in `logs/test`.

#### 3.2.2 Evaluate a trained network

Once trained, evaluate the network using a test dataset:

```
depreg_predict --gpu "" --ckpt_path logs/test/save/ckpt-2 --mode test
```

where:

- `--ckpt_path <filepath>` specifies the checkpoint file path.
- `--mode test` specifies prediction on the test dataset.
3.2.3 Warp an image

DeepReg provides a command line interface (CLI) tool to warp an image/label with a dense displacement field (DDF):

```
deepreg_warp --image data/test/nifti/unit_test/moving_image.nii.gz --ddf data/test/nifti/unit_test/ddf.nii.gz --out logs/test_warp/out.nii.gz
```

where:

- `--image <filepath>` specifies the image/label file path.
- `--ddf <filepath>` specifies the ddf file path.
- `--out <filepath>` specifies the output file path.

3.3 Image Registration with Deep Learning

A series of scientific tutorials on deep learning for registration can be found at the learn2reg tutorial, held in conjunction with MICCAI 2019.

This document provides a practical overview for a number of algorithms supported by DeepReg.

3.3.1 Registration

Image registration is the process of mapping the coordinate system of one image into another image. A registration method takes a pair of images as input, denoted as moving and fixed images. In this tutorial, we register the moving image into the fixed image, i.e. mapping the coordinates of the moving image onto the fixed image.

3.3.2 Network

Predict a dense displacement field

With deep learning, given a pair of moving and fixed images, the registration network outputs a dense displacement field (DDF) with the same shape as the moving image. Each value can be considered as the placement of the corresponding pixel / voxel of the moving image. Therefore, the DDF defines a mapping from the moving image’s coordinates to the fixed image.

In this tutorial, we mainly focus on DDF-based methods.

Predict a dense velocity field

Another option is to predict a dense (static) velocity field (DVF), such that a diffeomorphic DDF can be numerically integrated. Read “A fast diffeomorphic image registration algorithm” and “Diffeomorphic demons: Efficient non-parametric image registration” for more details.
**Predict an affine transformation**

A more constrained option is to predict an affine transformation, parameterised by the affine transformation matrix of 12 degrees of freedom. The DDF can then be computed to resample the moving images in fixed image space.

**Predict a region of interest**

Instead of outputting the transformation between coordinates, given moving image, fixed image, and a region of interest (ROI) in the moving image, the network can predict the ROI in the fixed image directly. Interested readers are referred to the MICCAI 2019 paper: Conditional segmentation in lieu of image registration

### 3.3.3 Loss

A loss function has to be defined to train a deep neural network. There are mainly three types of losses:

**Intensity based (image based) loss**

The common loss functions are normalized cross correlation (NCC), sum of squared distance (SSD), and normalized mutual information (MI).

Intensity based losses measure the dissimilarity between a fixed image and a warped moving image, which is an adaptation from classical image registration methods. These losses can perform poorly on multi-modality registrations (e.g. SSD loss in CT-MRI registration), although certain multi-modality tasks, such as registration between different MRI sequences are handled well by MI. Generally, intensity based losses work best when there is an inherent consistency in appearance between moving and fixed images, which is more common in single-modality registration.

**Feature based (label based) loss**

This type of loss measures the dissimilarity of the fixed image labels and warped moving image labels. The label is often an ROI in the image, like the segmentation of an organ in a CT image.

The common loss function is Dice loss, Jacard and average cross-entropy over all voxels.

**Deformation loss**

This type of loss measures the amount of deformation in an image and penalises non-smooth deformations. Penalising sudden or discontinuous deformations helps to regularise the transformation between fixed and moving images.

For DDF, the common loss functions are bending energy, L1 or L2 norm of the displacement gradient.

### 3.3.4 Learning

Depending on the availability of the data labels, registration networks can be trained with different approaches:
**Unsupervised**

When the data label is unavailable, the training can be driven by the unsupervised loss. The loss function often consists of the intensity based loss and deformation loss. The following is an illustration of an unsupervised DDF-based registration network.

**Weakly-supervised**

When there is no intensity based loss that is appropriate for the image pair one would like to register, the training can take a pair of corresponding moving and fixed labels (in addition to the image pair), represented by binary masks, to compute a label dissimilarity (feature based loss) to drive the registration.

Combined with the regularisation on the predicted displacement field, this forms a weakly-supervised training. An illustration of an weakly-supervised DDF-based registration network is provided below.

When multiple labels are available for each image, the labels can be sampled during the training iteration, such that only one label per image is used in each iteration of the data set (epoch).

**Combined**

When the data label is available, combining intensity based, feature based, and deformation based losses together has shown superior registration accuracy, compared to unsupervised and weakly supervised methods. Following is an illustration of a combined DDF-based registration network.

### 3.4 Design Experiments

DeepReg dataset loaders use a folder/directory-based file storing approach, with which the user will be responsible for organising image and label files in required file formats and folders. This design was primarily motivated by the need to minimise the risk of data leakage (or information leakage), both in code development and subsequent applications.

#### 3.4.1 Random-split

Every call of the `deepreg_train` or `deepreg_predict` function uses a dataset “physically” separated by folders, including ‘train’, ‘val’ and ‘test’ sets used in a random-split experiment. In this case, the user needs to randomly assign available experiment image and label files into the three folders. Again, for more details see the Dataset loader.

#### 3.4.2 Cross-validation

Experiments such as cross-validation can be readily implemented by using the “multi-folder support” in the dataset section of the yaml configuration files. See details in configuration.

For example, in a 3-fold cross-validation, the user may randomly partition available experiment data files into four folders, ‘fold0’, ‘fold1’, ‘fold2’ and ‘test’. The ‘test’ is a hold-out testing set. Each run of the 3-fold cross-validation then can be specified in a different yaml file as follows.

“cv_run1.yaml”:
DeepReg

dataset:

dir:

train: # training data set
- "data/test/h5/paired/fold0"
- "data/test/h5/paired/fold1"
valid: "data/test/h5/paired/fold2" # validation data set
test: ""

“cv_run2.yaml”:

dataset:

dir:

train: # training data set
- "data/test/h5/paired/fold0"
- "data/test/h5/paired/fold2"
valid: "data/test/h5/paired/fold1" # validation data set
test: ""

“cv_run3.yaml”:

dataset:

dir:

train: # training data set
- "data/test/h5/paired/fold1"
- "data/test/h5/paired/fold2"
valid: "data/test/h5/paired/fold0" # validation data set
test: ""

To further facilitate flexible uses of these dataset loaders, the deepreg_train and deepreg_predict functions also accept multiple yaml files - therefore the same train section does not have to be repeated multiple times for the multiple cross-validation folds or for the test. An example dataset section for configuring testing when using deepreg_predict is given below.

“test.yaml”:

dataset:

dir:

train: ""
valid: ""
test: "data/test/h5/paired/test" # validation data set
3.5 Running DeepReg Remotely (on the Cluster)

This tutorial gives an example of how to run DeepReg remotely (e.g. on a cluster). Our example is specific to the UCL cluster, which has operating system CentOS 7 (similar to Ubuntu), with job scheduler Sun Grid Engine (SGE). More information on the specific configuration at UCL is available here.

3.5.1 Installing the Environment

Install the environment in the cluster, as described here. In the case you do not have root access (as the case of the UCL cluster), you might need to use pip -u option to install the requirements.txt file.

3.5.2 Example Script

Below is the submission script for running quick start example here. Change <DeepReg_dir> in the script to the remote DeepReg repo location and save the below code in a <your_name>.qsub. Submit the job with qsub <your_name>.qsub and check the status of the job with qstat, the saved stdout and stderr is in <DeepReg_dir>/logs/.

```
#!/bin/bash
#$ -S /bin/bash # bash for job
#$ -l gpu=true # use gpu
#$ -l tmem=10G # virtual mem used
#$ -l h_rt=36:0:0 # max job runtime hour:min:sec
#$ -R y
#$ -N DeepReg_tst # job name
#$ -wd <DeepReg_dir>/logs # output, error log dir

hostname
date
cd <DeepReg_dir>
conda activate deepreg # activate conda env
export PATH=/share/apps/cuda-10.1/bin:/share/apps/gcc-8.3/bin:$PATH # path for cuda, gcc
export LD_LIBRARY_PATH=/share/apps/cuda-10.1/lib64:/share/apps/gcc-8.3/lib64:$LD_LIBRARY_PATH # path for cuda, gcc
deeprg_train 
  --gpu "0" 
  --config_path config/unpaired_labeled_ddf.yaml 
  --log_dir test
```

3.5.3 Contact and Version

Please contact stefano.blumberg.17@ucl.ac.uk, for information about the cluster. The information here is likely to change as UCL updates the software on the cluster.
3.6 Introduction to DeepReg Demos

DeepReg offers multiple built-in dataset loaders to support real-world clinical scenarios, in which images may be paired, unpaired or grouped. Images may also be labeled with segmented regions of interest to assist registration.

A typical workflow to develop a registration network using DeepReg includes:

- Select a dataset loader, among the unpaired, paired and grouped, and prepare data into folders as required;
- Configure the network training in the configuration yaml file(s), as specified in supported configuration details;
- Train and tune the registration network with the command line tool deepreg_train;
- Test or use the final trained registration network with the command line tool deepreg_predict.

Besides the tutorials, a series of DeepReg Demos are provided to showcase a wide range of applications with real clinical image and label data. These applications range from ultrasound, CT and MR images, covering many clinical specialties such as neurology, urology, gastroenterology, oncology, respiratory and cardiovascular diseases.

Each DeepReg Demo provides a step-by-step instruction to explain how different scenarios can be implemented with DeepReg. All data sets used are open-accessible. Pre-trained models with numerical and graphical inference results are also available.

Note: DeepReg Demos are provided to demonstrate functionalities in DeepReg. Although effort has been made to ensure these demos are representative of real-world applications, the implementations and the results are not peer-reviewed or tested for clinical efficacy. Substantial further adaptation and development may be required for any potential clinical adoption.

3.7 Paired Images

The following DeepReg Demos provide examples of using paired images.

- **Paired lung CT registration**
  This demo registers paired CT lung images, with optional weak supervision.

- **Paired brain MR-ultrasound registration**
  This demo registers paired preoperative MR images and 3D tracked ultrasound images for locating brain tumours during neurosurgery, with optional weak supervision.

- **Paired prostate MR-ultrasound registration**
  This demo registers paired MR-to-ultrasound prostate images, an example of weakly-supervised multimodal image registration.

3.7.1 Paired lung CT registration

Note: Please read the DeepReg Demo Disclaimer.
Author

DeepReg Development Team (Shaheer Saeed)

Application

This is a registration between CT images acquired at different time points for a single patient. The images being registered are taken at inspiration and expiration for each subject. This is an intra subject registration. This type of intra subject registration is useful when there is a need to track certain features on a medical image such as tumor location when conducting invasive procedures.

Data

The dataset for this demo comes from Lean2Reg Challenge: CT Lung Registration - Training Data [1].

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model. Image intensities are rescaled during pre-processing.

```
python demos/paired_ct_lung/demo_data.py
```

Launch demo training

Please execute the following command to launch a demo training. The training logs and model checkpoints will be saved under demos/paired_ct_lung/logs_train.

```
python demos/paired_ct_lung/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/paired_ct_lung/demo_train.py --no-test
```

3.7. Paired Images
Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under demos/paired_ct_lung/logs_predict. Check the CLI documentation for more details about prediction output.

```
python demos/paired_ct_lung/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside demo_predict.py. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
Contact

Please raise an issue for any questions.
3.7.2 Paired brain MR-ultrasound registration

Note: Please read the DeepReg Demo Disclaimer.

Warning: This demo ought to be improved in the future.

Source Code

Author

DeepReg Development Team (Shaheer Saeed)

Application

This demo aims to register pairs of brain MR and ultrasound scans. The dataset consists of 22 subjects with low-grade brain gliomas who underwent brain tumour resection [1]. The main application for this type of registration is to better delineate brain tumour boundaries during surgery and correct tissue shift induced by the craniotomy.

Data

The dataset for this demo comes from Xiao et al. [1] and can be downloaded from: https://archive.sigma2.no/pages/public/datasetDetail.jsf?id=10.11582/2020.00025.

Instruction

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model. By default, the downloaded data is only a partial of the original one. However the access to the original data is temporarily unavailable.

```
python demos/paired_mrus_brain/demo_data.py
```
Launch demo training

Please execute the following command to launch a demo training. The training logs and model checkpoints will be saved under `demos/paired_mr/us_brain/logs_train`.

```bash
python demos/paired_mr/us_brain/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```bash
python demos/paired_mr/us_brain/demo_train.py --no-test
```

Note: The number of epochs and reduced dataset size for training will result in a loss in test accuracy so please train with the full dataset and for a greater number of epochs for improved results.

Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/paired_mr/us_brain/logs_predict`. Check the CLI documentation for more details about prediction output.

```bash
python demos/paired_mr/us_brain/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```bash
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
Contact

Please raise an issue for any questions.
Reference


3.7.3 Paired prostate MR-ultrasound registration

Note: Please read the DeepReg Demo Disclaimer.

Warning: This demo ought to be improved in the future..

Source Code

This demo uses DeepReg to re-implement the algorithms described in Weakly-supervised convolutional neural networks for multimodal image registration. A standalone demo was hosted at https://github.com/yipenghu/label-reg.

Author

DeepReg Development Team

Application

Registering preoperative MR images to intraoperative transrectal ultrasound images has been an active research area for more than a decade. The multimodal image registration task assist a number of ultrasound-guided interventions and surgical procedures, such as targeted biopsy and focal therapy for prostate cancer patients. One of the key challenges in this registration task is the lack of robust and effective similarity measures between the two image types. This demo implements a weakly-supervised learning approach to learn voxel correspondence between intensity patterns between the multimodal data, driven by expert-defined anatomical landmarks, such as the prostate gland segmentation.

Data

This is a demo without real clinical data due to regulatory restrictions. The MR and ultrasound images used are simulated dummy images.

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.
DeepReg

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model.

```
python demos/paired_mrus_prostate/demo_data.py
```

Launch demo training

Please execute the following command to launch a demo training (the first of the ten runs of a 9-fold cross-validation). The training logs and model checkpoints will be saved under `demos/paired_mrus_prostate/logs_train`.

```
python demos/paired_mrus_prostate/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/paired_mrus_prostate/demo_train.py --no-test
```

Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/paired_mrus_prostate/logs_predict`. Check the CLI documentation for more details about prediction output.

```
python demos/paired_mrus_prostate/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
3.8 Unpaired Images

The following DeepReg Demos provide examples of using unpaired images.

- **Unpaired abdominal CT registration**

  This demo compares three training strategies, using unsupervised, weakly-supervised and combined losses, to register inter-subject abdominal CT images.

- **Unpaired lung CT registration**

  This demo registers unpaired CT lung images, with optional weak supervision.
DeepReg

- Unpaired hippocampus MR registration
  This demo aligns hippocampus on MR images between different patients, with optional weak supervision.
- Unpaired prostate ultrasound registration
  This demo registers 3D ultrasound images with a 9-fold cross-validation. This strategy is applicable for any of the available dataset loaders.

3.8.1 Unpaired abdomen CT registration

Note: Please read the DeepReg Demo Disclaimer.

Warning: This demo ought to be improved in the future.

Source Code

Author

DeepReg Development Team (Ester Bonmati)

Application

This demo shows how to register unpaired abdominal CT data from different patients using DeepReg. In addition, the demo demonstrates the difference between the unsupervised, weakly-supervised and their combination, using a U-Net.

Data

The data set is from the MICCAI Learn2Reg grand challenge (https://learn2reg.grand-challenge.org/) task 3 [1], and can be downloaded directly from https://learn2reg.grand-challenge.org/Datasets/.

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model.

```python
demos/unpaired_ct_abdomen/demo_data.py
```
Launch demo training

In this demo, three different training methods are provided: unsupervised, weakly supervised and the combined method. Please execute one of the following commands to launch a demo training. The training logs and model checkpoints will be saved under `demos/unpaired_ct_abdomen/logs_train/method` with `method` be `unsup`, `weakly` or `comb`.

```
python demos/unpaired_ct_abdomen/demo_train.py --method unsup
python demos/unpaired_ct_abdomen/demo_train.py --method weakly
python demos/unpaired_ct_abdomen/demo_train.py --method comb
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/unpaired_ct_abdomen/demo_train.py --method unsup --no-test
```

Predict

Please execute one of the following commands to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/unpaired_ct_abdomen/logs_predict/method` with `method` be `unsup`, `weakly` or `comb`. Check the CLI documentation for more details about prediction output.

```
python demos/unpaired_ct_abdomen/demo_predict.py --method unsup
python demos/unpaired_ct_abdomen/demo_predict.py --method weakly
python demos/unpaired_ct_abdomen/demo_predict.py --method comb
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
Contact

Please raise an issue for any questions.

Reference

3.8.2 Unpaired lung CT registration

Note: Please read the DeepReg Demo Disclaimer.

Source Code

Author

DeepReg Development Team (Shaheer Saeed)

Application

This is a registration between CT images from different patients. The images are all from acquired at the same time-point in the breathing cycle. This is an inter subject registration. This kind of registration is useful for determining how one stimulus affects multiple patients. If a drug or invasive procedure is administered to multiple patients, registering the images from different patients can give medical professionals a sense of how each patient is responding in comparison to others. An example of such an application can be seen in [2].

Data

The dataset for this demo comes from [1] and can be downloaded from: https://zenodo.org/record/3835682#.XsUWXsBpPhE

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model. Image intensities are rescaled during pre-processing.

```bash
python demos/unpaired_ct_lung/demo_data.py
```

Launch demo training

Please execute the following command to launch a demo training. The training logs and model checkpoints will be saved under demos/unpaired_ct_lung/logs_train.

```bash
python demos/unpaired_ct_lung/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```bash
python demos/unpaired_ct_lung/demo_train.py --no-test
```
Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under demos/unpaired_ct_lung/logs_predict. Check the CLI documentation for more details about prediction output.

```
python demos/unpaired_ct_lung/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside demo_predict.py. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
Contact

Please raise an issue for any questions.

Reference


3.8.3 Unpaired hippocampus MR registration

Note: Please read the DeepReg Demo Disclaimer.
Warning: This demo ought to be improved in the future.

Source Code

Author

DeepReg Development Team (Adrià Casamitjana)

Application

This is a demo targeting the alignment of hippocampal substructures (head and body) using mono-modal MR images between different patients. The images are cropped around those areas and manually annotated. This is a 3D intra-modal registration using a composite loss of image and label similarity.

Data

The dataset for this demo comes from the Learn2Reg MICCAI Challenge (Task 4) [1] and can be downloaded from: https://drive.google.com/uc?export=download&id=1RvJiG2IoU8uGkWzUuGjqVcGQW2RzNYA

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model.

```
python demos/unpaired_mr_brain/demo_data.py
```

Pre-processing includes:

- Rescaling all images’ intensity to 0-255.
- Creating and applying a binary mask to mask-out the padded values in images.
- Transforming label volumes using one-hot encoding (only for foreground classes)
Launch demo training

Please execute the following command to launch a demo training. The training logs and model checkpoints will be saved under demos/unpaired_mr_brain/logs_train.

```
python demos/unpaired_mr_brain/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/unpaired_mr_brain/demo_train.py --no-test
```

Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under demos/unpaired_mr_brain/logs_predict. Check the CLI documentation for more details about prediction output.

```
python demos/unpaired_mr_brain/demo_predict.py
```

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
Contact

Please raise an issue for any questions.
3.8.4 Unpaired prostate ultrasound registration

**Note:** Please read the DeepReg Demo Disclaimer.

**Source Code**

This DeepReg Demo is also an example of cross validation.

**Author**

DeepReg Development Team

**Application**

Transrectal ultrasound (TRUS) images are acquired from prostate cancer patients during image-guided procedures. Pairwise registration between these 3D images may be useful for intraoperative motion modelling and group-wise registration for population studies.

**Data**

The 3D ultrasound images used in this demo were derived from the Prostate-MRI-US-Biopsy dataset, hosted at the Cancer Imaging Archive (TCIA).

**Instruction**

**Installation**

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

**Download data**

Please execute the following command to download/pre-process the data and download the pre-trained model. Data are split into 10 folds for cross-validation.

```
python demos/unpaired_us_prostate_cv/demo_data.py
```
**Launch demo training**

Please execute the following command to launch a demo training (the first of the ten runs of a 9-fold cross-validation). The training logs and model checkpoints will be saved under `demos/unpaired_us_prostate_cv/logs_train`.

```
python demos/unpaired_us_prostate_cv/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/unpaired_us_prostate_cv/demo_train.py --no-test
```

**Predict**

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/unpaired_us_prostate_cv/logs_predict`. Check the CLI documentation for more details about prediction output.

```
python demos/unpaired_us_prostate_cv/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

**Visualise**

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs [here](#) for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
3.9 Grouped Images

The following DeepReg Demos provide examples of using grouped images.

- Pairwise registration for grouped prostate segmentation masks
  
  This demo registers grouped masks (as input images) of prostate glands from MR images, an example of feature-based registration.

- Pairwise registration for grouped cardiac MR images

Contact

Please raise an issue for any questions.
DeepReg

This demo registers grouped CMR images, where each group has multi-sequence CMR images from a single patient.

3.9.1 Pairwise registration for grouped prostate segmentation masks

Note: Please read the DeepReg Demo Disclaimer.

Source Code

This demo uses DeepReg to demonstrate a number of features:

- For grouped data in h5 files, e.g. “group-1-2” indicates the 2th visit from Subject 1;
- Use masks as the images for feature-based registration - aligning the prostate gland segmentation in this case - with deep learning;
- Register intra-patient longitudinal data.

Author

DeepReg Development Team

Application

Longitudinal registration detects the temporal changes and normalises the spatial difference between images acquired at different time-points. For prostate cancer patients under active surveillance programmes, quantifying these changes is useful for detecting and monitoring potential cancerous regions.

Data

This is a demo without real clinical data due to regulatory restrictions. The MR and ultrasound images used are simulated dummy 3D Ultrasound images. Data are organized into 10 separate folds.

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download/pre-process the data and download the pre-trained model.

```
python demos/grouped_mask_prostate_longitudinal/demo_data.py
```
Launch demo training

Please execute the following command to launch a demo training (the first of the ten runs of a 9-fold cross-validation). The training logs and model checkpoints will be saved under `demos/grouped_mask_prostate_longitudinal/logs_train`.

```
python demos/grouped_mask_prostate_longitudinal/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/grouped_mask_prostate_longitudinal/demo_train.py --no-test
```

Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/grouped_mask_prostate_longitudinal/logs_predict`. Check the CLI documentation for more details about prediction output.

```
python demos/grouped_mask_prostate_longitudinal/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs [here](#) for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
3.9.2 Pairwise registration for grouped cardiac MR images

Note: Please read the DeepReg Demo Disclaimer.

Source Code
This demo uses the grouped dataset loader to register intra-subject multi-sequence cardiac magnetic resonance (CMR) images.
Author
DeepReg Development Team

Application
Computer-assisted management for patients suffering from myocardial infraction (MI) often requires quantifying the difference and comprising the multiple sequences, such as the late gadolinium enhancement (LGE) CMR sequence MI, the T2-weighted CMR. They collectively provide radiological information otherwise unavailable during clinical practice.

Data
This demo uses CMR images from 45 patients, acquired from the MyoPS2020 challenge held in conjunction with MICCAI 2020.

Instruction
Installation
Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data
Please execute the following command to download/pre-process the data and download the pre-trained model. Images are re-sampled to an isotropic voxel size.

```
python demos/grouped_mr_heart/demo_data.py
```

Launch demo training
Please execute the following command to launch a demo training (the first of the ten runs of a 9-fold cross-validation). The training logs and model checkpoints will be saved under demos/grouped_mr_heart/logs_train.

```
python demos/grouped_mr_heart/demo_train.py
```

Here the training is launched using the GPU of index 0 with a limited number of steps and reduced size. Please add flag `--no-test` to use the original training configuration, such as

```
python demos/grouped_mr_heart/demo_train.py --no-test
```
Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/grouped_mr_heart/logs_predict`. Check the CLI documentation for more details about prediction output.

```
python demos/grouped_mr_heart/demo_predict.py
```

Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.
3.9. Grouped Images

Reference


Contact

Please raise an issue for any questions.
3.10 Classical Registration

The following DeepReg Demos provide examples of using classical registration methods.

- Classical affine registration for head-and-neck CT images
  This demo registers head-and-neck CT images using iterative affine registration.
- Classical nonrigid registration for prostate MR images
  This demo registers prostate MR images using iterative nonrigid registration.

3.10.1 Classical affine registration for head-and-neck CT images

Note: Please read the DeepReg Demo Disclaimer.

Source Code

This is a special demo that uses the DeepReg package for classical affine image registration, which iteratively solves an optimisation problem. Gradient descent is used to minimise the image dissimilarity function of a given pair of moving and fixed images.

Author

DeepReg Development Team

Application

Although in this demo the moving images are simulated using a randomly generated transformation. The registration technique can be used in radiotherapy to compensate the difference between CT acquired at different time points, such as pre-treatment and intra-/post-treatment.

Data

Data is an example CT volume with two labels.

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.
Download data

Please execute the following command to download and pre-process the data.

```python
deprecated demos/classical_ct_headneck_affine/demo_data.py
```

Launch registration

Please execute the following command to register two images. The fixed image will be the downloaded data and the moving image will be simulated by applying a random affine transformation, such that the ground-truth is available for. The optimised transformation will be applied to the moving images, as well as the moving labels. The results, saved in a timestamped folder under the project directory, will compare the warped image/labels with the ground-truth image/labels.

```python
deprecated demos/classical_ct_headneck_affine/demo_register.py
```

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```bash
deprecated deepreg_vis -m 2 -i 'demos/classical_ct_headneck_affine/logs_reg/moving_image.nii.gz,'
  -> demos/classical_ct_headneck_affine/logs_reg/warped_moving_image.nii.gz, demos/
  --classical_ct_headneck_affine/logs_reg/fixed_image.nii.gz' --slice-inds '4,8,12' -s,
  --> demos/classical_ct_headneck_affine/logs_reg
```

Note: The registration script must be run before running the command to generate the visualisation.
Contact

Please raise an issue for any questions.
Reference


3.10.2 Classical nonrigid registration for prostate MR images

Note: Please read the DeepReg Demo Disclaimer.

Source Code

This is a special demo that uses the DeepReg package for classical nonrigid image registration, which iteratively solves an optimisation problem. Gradient descent is used to minimise the image dissimilarity function of a given pair of moving and fixed images, often regularised by a deformation smoothness function.

Author

DeepReg Development Team

Application

Registering inter-subject prostate MR images may be useful to align different glands in a common space for investigating the spatial distribution of cancer.

Data

Data is an example MR volumes with the prostate gland segmentation from MICCAI Grand Challenge: Prostate MR Image Segmentation 2012.

Instruction

Installation

Please install DeepReg following the instructions and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

Download data

Please execute the following command to download and pre-process the data.

```
python demos/classical_mr_prostate_nonrigid/demo_data.py
```
Launch registration

Please execute the following command to register two images. The optimised transformation will be applied to the moving images, as well as the moving labels. The results, saved in a timestamped folder under the project directory, will compare the warped image/labels with the ground-truth image/labels.

```
python demos/classical_mr_prostate_nonrigid/demo_register.py
```

Visualise

The following command can be executed to generate a plot of three image slices from the the moving image, warped image and fixed image (left to right) to visualise the registration. Please see the visualisation tool docs here for more visualisation options such as animated gifs.

```
```

Note: The registration script must be run before running the command to generate the visualisation.
Contact

Please raise an issue for any questions.
3.11 Command Line Tools

With DeepReg installed, multiple command line tools are available, currently including:

- `deepreg_train`, for training a registration network.
- `deepreg_predict`, for evaluating a trained network.
- `deepreg_warp`, for warping an image with a dense displacement field.

3.11.1 Train

`deepreg_train` accepts the following arguments via command line tools. More configuration can be specified in the configuration file. Please see configuration file for further details.

Required arguments

- **GPU:**
  --gpu or -g, specifies the index or indices of GPUs for training.
  Example usage:
  - `--gpu ""` for CPU only
  - `--gpu "0"` for using only GPU 0
  - `--gpu "0,1"` for using GPU 0 and 1.

- **Configuration:**
  --config_path or -c, specifies the configuration file for training.
  The path must end with .yaml.
  Optionally, multiple paths can be specified, and the configuration will be merged. In case of conflicts, values are overwritten by the last config file defining them.
  Example usage:
  - `--config_path config1.yaml` for using one single configuration file.
  - `--config_path config1.yaml config2.yaml` for using multiple configuration files.
Optional arguments

- **GPU memory allocation:**
  
  `--gpu_allow_growth` or `-gr`, if given, TensorFlow will only grow the memory usage as is needed. By default it allocates all available GPU memory.
  
  Example usage:
  
  `- --gpu_allow_growth`, no extra argument is needed.

- **Load checkpoint:**
  
  `--ckpt_path` or `-k`, specifies the path of the saved model checkpoint, so that the training will be resumed from the given checkpoint.
  
  The path must end with `.ckpt`.
  
  By default it starts training from a random initialization.
  
  Example usage:
  
  `- --ckpt_path weights-epoch2.ckpt` for reloading the given checkpoint.

- **Output root:**
  
  `--log_root`, specifies the directory for saving all logs.
  
  By default it is `logs` under the root of package.
  
  Example usage:
  
  `- --log_root /logs` for saving all logs under `/logs`.

- **Output directory:**
  
  `--log_dir` or `-l`, specifies the directory name to save logs.
  
  The directory will be under the `log_root` which is `logs` by default.
  
  By default it creates a timestamp-named directory, e.g. `logs/20200810-194042/`.
  
  Example usage:
  
  `- --log_dir test` for saving under `logs/test/`.

- **Maximum number of epochs:**
  
  `--max_epochs`, specifies the maximum number of epochs for training and overwrites the value defined in the configuration.
  
  By default, the value is -1, meaning the number of epochs will be defined by configuration.
  
  Example usage:
  
  `- --max_epochs 2` for run training only for two epochs.
Output

During the training, multiple output files will be saved in the log directory `logs/log_dir`, where `log_dir` is specified in the arguments, otherwise a timestamped folder name will be used. The output files are:

- **config.yaml** is a backup of the used configuration. It can be used for prediction. In case of multiple configuration files, a merged configuration file will be saved.
- **train/** and **validation/** are the directories that save tensorboard logs on metrics.
- **save/** is the directory containing saved checkpoints of the trained network.

3.11.2 Predict

depreg_predict accepts the following arguments via command line tools. More configuration can be specified in the configuration file. Please see configuration file for further details.

Required arguments

- **GPU:**
  
  --gpu or -g, specifies the index or indices of GPUs for training.
  
  Example usage:
  
  - --gpu "" for CPU only
  - --gpu "0" for using only GPU 0
  - --gpu "0,1" for using GPU 0 and 1.

- **Model checkpoint:**
  
  --ckpt_path or -k, specifies the path of the saved model checkpoint, so that the trained model will be loaded for evaluation.
  
  The path must end with .ckpt.
  
  Example usage:
  
  - --ckpt_path weights-epoch2.ckpt for reloading the given checkpoint.

- **Evaluation data:**
  
  --mode or -m, specifies in which data set the prediction is performed.
  
  It must be one of train/valid/test.
  
  Example usage:
  
  - --mode test for evaluating the model on test data.
Optional arguments

- **GPU memory allocation:**
  --gpu_allow_growth or -gr, if given, TensorFlow will only grow the memory usage as is needed.
  By default it allocates all availables in the GPU memory.
  Example usage:
  - --gpu_allow_growth, no extra argument is needed.

- **Output root:**
  --log_root, specifies the directory for saving all logs.
  By default it is logs under the root of package.
  Example usage:
  - --log_root /logs for saving all logs under /logs.

- **Output directory:**
  --log_dir or -l, specifies the directory name to save logs.
  The directory will be under the log_root which is logs by default.
  By default it creates a timestamp-named directory, e.g. logs/20200810-194042/.
  Example usage:
  - --log_dir test for saving under logs/test/.

- **Batch size:**
  --batch_size or -b, specifies the mini-batch size (per GPU) for prediction.
  The default value is 1.
  Example usage:
  - --batch_size 2 for using a mini-batch size of 2.

- **Save outputs in Nifti format:**
  The predicted 3D tensors can be saved in Nifti format for further calculation.
  By default it saves outputs in Nifti format.
  Example usage:
  - --save_nifti, for saving the outputs in Nifti format.
  - --no_nifti, for not saving the outputs in Nifti format.

- **Save outputs in png format:**
  The predicted 3D tensors can be saved as a slice of 2D images for quick visualization.
  As values have to be normalized between 0~255 (or 0~1) for png files (Nifti files are not impacted), all images (moving_image, fixed_image and pred_fixed_image) and displacement/velocity fields (ddf and dvf) will be normalized before being saved. Labels (moving_label, fixed_label and pred_fixed_label) are not affected as they are already within 0~1.
  By default it saves the outputs in png format.
  Example usage:
  - --save_png, for saving the outputs in png format.
DeepReg

- `--no_png`, for not saving the outputs in png format.

- **Configuration:**
  - `--config_path` or `-c`, specifies the configuration file for prediction. The path must end with `.yaml`. By default it uses the configuration file saved in the directory of the given checkpoint. Example usage:
    - `--config_path config1.yaml` for using one single configuration file.

**Output**

During the evaluation, multiple output files will be saved in the log directory `logs/log_dir/mode` where

- `log_dir` is defined in arguments, or a timestamped folder name will be used;
- `mode` is `train` or `valid` or `test`, specified by the argument.

The saved files include:

- **Metrics to evaluate the registration performance**
  - `metrics.csv` saves the metrics on all samples. Each line corresponds to a data sample.
  - `metrics_stats_per_label.csv` saves the mean, median and std of each metrics on all samples with the same label index.
  - `metrics_stats_overall.csv` saves a set of commonly used statistics (such as mean and std) on the metrics over all samples.

- **Inputs and predictions for each pair of image.**
  Each pair has its own directory and the followings tensors are saved inside if available. Tensors can be saved in Nifti format (one single file) or in png format (one folder contains all image slices, ordered by depth) or both.
  - `ddf`, `dvf`, `affine`
    - DDF stands for dense displacement field; DVF stands for dense (static) velocity field.
    - The 12 parameters of affine transformation are saved in `affine.txt`.
  - `moving_image`, `fixed_image` and `pred_fixed_image`
    - `pred_fixed_image` is the warped moving image if the network predicts a DDF or a DVF or an affine transformation.
  - `moving_label`, `fixed_label` and `pred_fixed_label` under directory `label_i` if the sample is labeled and `i` is the label index.
    - `pred_fixed_label` is the predicted label in the fixed image space. In many cases, this is equivalent to the warped moving label, if the network predicts a DDF or a DVF or an affine transformation.
3.11.3 Warp

deeprg_warp accepts the following arguments:

**Required arguments**

- **Image file:**
  
  --image or -i, specifies the file path of the image/label.
  
  The image/label should be saved in a Nifiti file with suffix .nii or .nii.gz. The image/label should be a 3D / 4D tensor, where the first three dimensions correspond to the moving image shape and the fourth can be a channel of features.
  
  Example usage:
  
  --image input_image.nii.gz

- **DDF file:**

  --ddf or -d, specifies the file path of the DDF.

  The DDF should be saved in a Nifiti file with suffix .nii or .nii.gz. The DDF should be a 4D tensor, where the first three dimensions correspond to the fixed image shape and the fourth dimension has 3 channels corresponding to x, y, z axes.

  Example usage:
  
  --image input_DDF.nii.gz

**Optional arguments**

- **Output directory:**

  --out or -o, specifies the file path for the output.

  The path should end with .nii or .nii.gz, otherwise the output path will be corrected automatically based on the given path.

  By default it saves the output as warped.nii.gz in the current directory.

  Example usage:
  
  --out output_image.nii.gz

**Output**

The warped image is saved in the given output file path, otherwise the default file path warped.nii.gz will be used.
3.11.4 Visualise

In addition to the images in the output, DeepReg provides a set of tools with the command `deepreg_vis`. See more details in its usage documentation.

3.12 Configuration File

In addition to the arguments provided to the command line tools, detailed training and prediction configuration is specified in a YAML file. The configuration file contains two sections, `dataset` and `train`. Within `dataset` one specifies the data file formats, sizes, as well as the data loader to use. The `train` section specifies parameters related to the neural network.

3.12.1 Dataset section

The `dataset` section specifies the path to the data to be used during training, the data loader to use as well as the specific arguments to configure the data loader.

**Dir key - Required**

The paths to the training, validation and testing data are specified under a `dir` dictionary key like this:

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train"  # folder containing training data
    valid: "data/test/h5/paired/valid"  # folder containing validation data
    test: "data/test/h5/paired/test"  # folder containing test data
```

Multiple dataset directories can be specified, such that data are sampled across several folders:

```yaml
dataset:
  dir:
    train:  # folders containing training data
      - "data/test/h5/paired/train1"
      - "data/test/h5/paired/train2"
    valid: "data/test/h5/paired/valid"  # folder containing validation data
    test: "data/test/h5/paired/test"  # folder containing test data
```

**Format key - Required**

The data file format we supply the data loaders will influence the behavior, so we must specify the data file format using the `format` key. Currently, DeepReg data loaders support Nifti and H5 file types - alternate file formats will raise errors in the data loaders. To indicate which format to use, pass a string to this field as either “nifti” or “h5”:

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train"  # folder containing training data
    valid: "data/test/h5/paired/valid"  # folder containing validation data
    test: "data/test/h5/paired/test"  # folder containing test data
  format: "nifti"
```

Depending on the data file format, DeepReg expects the images and labels to be stored in specific structures: check the data loader configuration for more details.
Labeled key - Required

The labeled key indicates whether segmentation labels should be used during training. A Boolean is used to indicate the usage of labels:

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train" # folder containing training data
    valid: "data/test/h5/paired/valid" # folder containing validation data
    test: "data/test/h5/paired/test" # folder containing test data
  format: "nifti"
  labeled: true
```

If the value passed is false, the labels will not be used in training even when they are available in the associated directories.

Type key - Required

The type of data loader used will depend on how one wants to train the network. Currently, DeepReg data loaders support the paired, unpaired, and grouped training strategies. Passing a string that doesn’t match any of the above would raise an error. The data loader type would be specified using the type key:

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train" # folder containing training data
    valid: "data/test/h5/paired/valid" # folder containing validation data
    test: "data/test/h5/paired/test" # folder containing test data
  format: "nifti"
  type: "paired" # one of "paired", "unpaired" or "grouped"
```

Data loader dependent keys

Depending on which string is passed to the type key, DeepReg will initialize a different data loader instance with different sampling strategies. These are described in depth in the dataset loader configuration documentation. Here we outline the arguments necessary to configure the different data loaders.

Sample_label - Required

In the case that we have more than one label per image, we need to inform the loader which one to use. We can use the sample_label argument to indicate which method to use during training.

- **all**: for one image that has x number of labels, the loader yields x image-label pairs with the same image. Occurs over all images, over one epoch.
- **sample**: for one image that has x number of labels, the loader yields 1 image-label pair randomly sampled from all the labels. Occurs for all images in one epoch.

During validation and testing (ie for valid and test directories), data loaders will be built to sample all the data-label pairs, regardless of the argument passed to sample_label.
In the case the labeled argument is false, the sample_label is unused, but still must be passed. Additionally, if the tensors in the files only have one label, regardless of the sample_label argument, the data loader will only pass the one label to the network.

For more details please refer to Read The Docs.

**Paired**

- moving_image_shape: (list, tuple) of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D moving image.
- fixed_image_shape: (list, tuple) of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D fixed image.

**Unpaired**

- image_shape: (list, tuple) of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D image.
Grouped

- **intra_group_prob**: float, between 0 and 1. Passing 0 would only generate inter-group samples, and passing 1 would only generate intra-group samples.
- **sample_label**: method for sampling the labels “sample”, “all”.
- **intra_group_option**: str, “forward”, “backward”, or “unconstrained”
- **sample_image_in_group**: bool, if true, only one image pair will be yielded for each group, so one epoch has num_groups pairs of data, if false, iterate through this loader will generate all possible pairs.
- **image_shape**: (list, tuple) len 3, corresponding to (dim1, dim2, dim3) of the 3D image.

```python
dataset:
  dir:
    train: "data/test/h5/paired/train" # folder containing training data
    valid: "data/test/h5/paired/valid" # folder containing validation data
    test: "data/test/h5/paired/test" # folder containing test data
  format: "nifti"
  type: "grouped" # one of "paired", "unpaired" or "grouped"
  labeled: true
  sample_label: "sample" # one of "sample", "all" or None
  image_shape: [16, 16, 3]
  sample_image_in_group: true
  intra_group_prob: 0.7
  intra_group_option: "forward"
```

See the [dataset loader configuration](#) for more details.

### 3.12.2 Train section

The **train** section defines the neural network training hyper-parameters, by specifying subsections, **method**, backbone, loss, optimizer, preprocess and other training hyper-parameters, including epochs and save_period.

**Method - required**

The **method** argument defines the registration type. It must be a string. Feasible values are: **ddf**, **dvf**, and **conditional**, corresponding to the dense displacement field (DDF) based model, dense velocity field (DDF) based model, and conditional model presented in the [registration tutorial](#).

```python
train:
  method: "ddf" # One of ddf, dvf, conditional
```
**Backbone - required**

The backbone subsection is used to define the network, with all the network-specific arguments under the same indent. The first argument should be the argument `name`, which should be string type, one of “unet”, “local” or “global”, to define a UNet, LocalNet or GlobalNet backbone, respectively. With Registry functionalities, you can also define your own networks to pass to DeepReg train via config.

The `num_channel_initial` is used to define the number of initial channels for the network, and should be int type.

```yaml
train:
  method: "ddf"  # One of ddf, dvf, conditional
  backbone:
    name: "unet"  # One of unet, local, global
    num_channel_initial: 16  # Int type, number of initial channels in the network.

  → Controls the network size.
```

**UNet**

The UNet model requires several additional arguments to define its structure:

- `depth`: int, defines the depth of the UNet from first to bottom, bottleneck layer.
- `pooling`: Boolean, pooling method used for down-sampling. True: non-parametrized pooling will be used, False: conv3d will be used.
- `concat_skip`: Boolean, concatenation method for skip layers in UNet. True: concatenation of layers, False: addition is used instead.

```yaml
train:
  method: "ddf"  # One of ddf, dvf, conditional
  backbone:
    name: "unet"  # One of unet, local, global
    num_channel_initial: 16  # Int type, number of initial channels in the network.
    depth: 3
    pooling: false
    concat_skip: true

  → Controls the network size.
```

**Local and GlobalNet**

The LocalNet has an encoder-decoder structure and extracts information from tensors at one or multiple resolution levels. We can define which levels to extract info from with the `extract_levels` argument.

The GlobalNet encodes the image and uses the bottleneck layer to output an affine transformation using a CNN.

- `extract_levels`: list of positive ints (ie, the min value in `extract_levels` should be >=0). WARNING: this argument will be deprecated in a future release as it is not used by the network.

```yaml
train:
  method: "ddf"  # One of ddf, dvf, conditional
  backbone:
    name: "local"  # One of unet, local, global
    num_channel_initial: 16  # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
```

```yaml
 extract_levels: [0, 1, 2]
  → Controls the network size.
```
Loss - required

This section defines the loss in training.

There are three different categories of losses in DeepReg:

- **image loss**: loss between the fixed image and predicted fixed image (warped moving image).
- **label loss**: loss between the fixed label and predicted fixed label (warped moving label).
- **regularization loss**: loss on predicted dense displacement field (DDF).

Not all losses are applicable for all models, the details are in the following table.

<table>
<thead>
<tr>
<th>Loss Type</th>
<th>DDF / DVF</th>
<th>Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Loss</td>
<td>Applicable</td>
<td>Non-applicable</td>
</tr>
<tr>
<td>Label Loss</td>
<td>Applicable if data are labeled</td>
<td>Applicable</td>
</tr>
<tr>
<td>Regularization Loss</td>
<td>Applicable</td>
<td>Non-applicable</td>
</tr>
</tbody>
</table>

The configuration for non-applicable losses will be ignored without errors. The loss will also be ignored if the weight is zero. However, each model must define at least one loss, otherwise error will be raised by TensorFlow.

For each loss, there are multiple existing loss functions to choose. The registry mechanism can also be used to use custom loss functions. Please read the registry documentation for more details.

Image

The image loss calculates dissimilarity between warped image tensors and fixed image tensors.

- **weight**: float type, the weight of individual loss element in the total loss function.
- **name**: string type, one of “lncc”, “ssd” or “gmi”.

```yaml
train:
  method: "ddf"  # One of ddf, dvf, conditional
  backbone:
    name: "local"  # One of unet, local, global
    num_channel_initial: 16  # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    image:
      name: "lncc"  # other options include "lncc", "ssd" and "gmi", for local
      weight: 0.1
```

The following are the DeepReg image losses. Additional arguments should be added at the same indent level:

- **lncc**: Calls a local normalized cross-correlation type loss. Requires the following arguments:
  - kernel_size: int, optional, default=9. Kernel size or kernel sigma for kernel_type=”gaussian”.
  - kernel_type: str, optional, default=”rectangular”. One of “rectangular”, “triangular” or “gaussian”
- **ssd**: Calls a sum of squared differences loss. No additional arguments required.
- **gmi**: Calls a global mutual information loss. Requires the following arguments:
  - num_bins: int, optional, default=23. Number of bins for intensity.
– **sigma_ratio**: float, optional, default=0.5. A hyperparameter for the Gaussian kernel density estimation.

**Label**

The label loss calculates dissimilarity between labels.

All default DeepReg losses can be used as multi-scale or single scale losses. Multi-scale losses require a kernel. Additionally, all losses can be weighted, so the following two arguments are global to all provided losses:

• **weight**: float type, weight of individual loss element in total loss function.
• **scales**: list of ints, or None. Optional argument. If you do not pass this argument (or pass the list [0], the value null or an empty value pair), the loss is calculated at a single scale. If you pass a list of length > 1, a multi-scale loss will be used. WARNING: an empty list ([]) will raise an error.

• **kernel**: str, “gaussian” or “cauchy”, default “gaussian”. Optional argument. Defines the kernel to use for multi-scale losses.

**EG.**

```python
train:
    method: "ddf" # One of ddf, dvf, conditional
backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network. Controls the network size.
    extract_levels: [0, 1, 2]
loss:
    label:
        weight: 1.0
        name: "dice" # options include "dice", "cross-entropy", "mean-squared", "generalised_dice" and "jaccard"
        scales: [1, 2]
```

The default losses require the following arguments. Additional arguments should be added at the same indent level:

• **dice**: Calls a Dice loss on the labels, requires the following arguments:
  – **binary**: bool, default is false. If true, the tensors are thresholded at 0.5.
  – **neg_weight**: float, default=0.0. neg_weight weights the foreground and background classes by replacing the labels of 1s and 0s with (1-neg_weight) and neg_weight, respectively.

• **cross-entropy**: Calls a cross-entropy loss between labels, requires the following arguments:
  – **binary**: bool, default is false. If true, the tensors are thresholded at 0.5.
  – **neg_weight**: float, default=0.0. neg_weight weights the foreground and background classes by replacing the labels of 1s and 0s with (1-neg_weight) and neg_weight, respectively.

• **jaccard**: - **binary**: bool, default is false. If true, the tensors are thresholded at 0.5.
Regularization

The regularization section configures the losses for the DDF. To instantiate this part of the loss, pass “regularization” into the config file as a field.

- **weight**: float type, the weight of the regularization loss.
- **name**: string type, the type of deformation energy to compute. Options include “bending”, “gradient”

If the gradient loss is used, another argument must be passed at the same indent level: - l1: bool. Indicates whether to calculate the L1-norm (true) or L2-norm (false) gradient loss of the ddf.

**EG.**

```python
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    regularization:
      weight: 0.5 # weight of regularization loss
      name: "bending" # options include "bending", "gradient"

or

train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    regularization:
      weight: 0.5 # weight of regularization loss
      name: "gradient" # options include "bending", "gradient"
    l1: false
```

**Multiple Losses**

Add multiple losses by putting all the fields in the same file.

```python
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    image:
      name: "gmi"
      weight: 1.0
    label:

(continues on next page)
```
Optimizer - required

The optimizer can be defined by using a name and then passing optimizer specific arguments with the same name. All optimizers can use the learning_rate argument.

- **name**: string type, is used to define the optimizer during training. One of “adam”, “sgd”, “rms”.
  - **adam**: If adam is passed into name, the adam field must be passed. The dictionary can be empty, which initializes a default Keras Adam optimizer. Alternatively, fields with names equivalent to those specified in the optimizer documentation can be used.
  - **sgd**: If sgd is passed into name, the sgd field must be passed. The dictionary can be empty, which initializes a default Keras SGD optimizer. Alternatively, fields with names equivalent to those specified in the optimizer documentation can be used instead.
  - **rms**: If rms is passed into name, the rms field must be passed. The dictionary can be empty, which initializes a default Keras RMSprop optimizer. Alternatively, fields with names equivalent to those specified in the optimizer documentation can be used instead.

```json
weight: 1.0
name: "dice"
```

```
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    regularization:
      weight: 0.5 # weight of regularization loss
      name: "bending" # options include "bending", "gradient"
    optimizer:
      name: "adam"
      adam:
```

or

```json
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.
    extract_levels: [0, 1, 2]
  loss:
    regularization:
      weight: 0.5 # weight of regularization loss
      name: "bending" # options include "bending", "gradient"
    optimizer:
      name: "sgd"
      sgd:
        learning_rate: 1.0e-5
        momentum: 0.9
        nesterov: false
```
**Preprocess - required**

The `preprocess` field defines how the data loader feeds data into the model.

- **batch_size**: int, the batch size to pass to the network on each training step.
- **shuffle_buffer_num_batch**: int, helps define how much data should be pre-loaded into memory to buffer training, such that `shuffle_buffer_size = batch_size * shuffle_buffer_num_batch`.

```python
train:
    method: "ddf" # One of ddf, dvf, conditional
    backbone:
        name: "local" # One of unet, local, global
        num_channel_initial: 16 # Int type, number of initial channels in the network.
        extract_levels: [0, 1, 2]
    loss:
        regularization:
            weight: 0.5 # weight of regularization loss
            name: "bending" # options include "bending", "gradient"
    optimizer:
        name: "sgd"
        sgd:
            learning_rate: 1.0e-5
            momentum: 0.9
            nesterov: false
    preprocess:
        batch_size: 32
        shuffle_buffer_num_batch: 1
```

**Epochs - required**

The `epochs` field defines the number of epochs to train the network for.

```python
train:
    method: "ddf" # One of ddf, dvf, conditional
    backbone:
        name: "local" # One of unet, local, global
        num_channel_initial: 16 # Int type, number of initial channels in the network.
        extract_levels: [0, 1, 2]
    loss:
        regularization:
            weight: 0.5 # weight of regularization loss
            name: "bending" # options include "bending", "gradient"
    optimizer:
        name: "sgd"
        sgd:
            learning_rate: 1.0e-5
            momentum: 0.9
            nesterov: false
    preprocess:
        batch_size: 32
        shuffle_buffer_num_batch: 1
    epochs: 1000
```
The `save_period` field defines the save frequency - the model will be saved every `save_period` epochs.

```
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "local" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network. Controls the network size.
    extract_levels: [0, 1, 2]
  loss:
    regularization:
      weight: 0.5 # weight of regularization loss
      name: "bending" # options include "bending", "gradient"
    optimizer:
      name: "sgd"
      sgd:
        learning_rate: 1.0e-5
        momentum: 0.9
        nesterov: false
  preprocess:
    batch_size: 32
    shuffle_buffer_num_batch: 1
  epochs: 1000
  save_period: 5
```

### 3.13 Dataset Loader

#### 3.13.1 Dataset type

DeepReg provides six dataset loaders to support the following three different types of datasets:

- **Paired images**
  
  Images are organized into moving and fixed image pairs.
  
  An example case is two-modalities intra-subject registration, such as registering one subject’s MR image to the corresponding ultrasound image.

- **Unpaired images**
  
  Images may be considered independent samples.
  
  An example case is single-modality inter-subject registration, such as registering one CT image to another from different subjects.

- **Grouped images**
  
  Images are organized into multiple groups.
  
  An example case is single-modality intra-subject registration, such as registering time-series images within individual subjects, a group is one subject in this case.

For all three above cases, the images can be either unlabeled or labeled. A label is represented by a boolean mask on the image, such as a segmentation of an anatomical structure or landmark.
3.13.2 Dataset requirements

To use the provided dataset loaders, other detailed images and labels requirements are described in individual dataset loader sections. General requirements are described as follows.

- **Image**
  - DeepReg currently supports 3D images. But images do not have to be of the same shape, and it will be resized to the required shape using linear interpolation.
  - Currently, DeepReg only supports images stored in Nifti files or H5 files. Check Nifti_loader and h5_loader for more details.
  - **Images are automatically normalized** at per-image level: the intensity values $x$ equals to $(x - \text{min}(x) + \text{EPS}) / (\text{max}(x) - \text{min}(x) + \text{EPS})$ so that its values are between [0,1]. Check GeneratorDataLoader.data_generator in loader interface for more details.

- **Label**
  - If an image is labeled, the label shape is recommended to be the same as the image shape. Otherwise, the resize might give unexpected behaviours. But each image can have more than one labels.
  
    For instance, an image of shape $(\text{dim1}, \text{dim2}, \text{dim3})$, its label shape can be $(\text{dim1}, \text{dim2}, \text{dim3})$ (single label) or $(\text{dim1}, \text{dim2}, \text{dim3}, \text{num_labels})$ (multiple labels).
  
  - **All labels are assumed to have values between [0, 1]**. So DeepReg accepts binary segmentation masks or soft labels with float values between [0,1]. This is to prevent accidental use of non-one-hot encoding to represent multiple class labels. In case of multi labels, please use one-hot encoding to transform them into multiple channels such that each class has its own binary label.
  
  - When the images are paired, the moving and fixed images must have the same number of labels.
  
  - When there are multiple labels, it is assumed that the labels are ordered, such that the channel of index $\text{label_idx}$ is the same anatomical or pathological structure.
  
  - Currently, if the data are labeled, each data sample must have at least one label. For missing labels, consider using all zeros as a workaround.

3.13.3 Paired images

For paired images, each pair contains a moving image and a fixed image. Optionally, corresponding moving label(s) and fixed label(s).

Specifically, given a pair of images

- When the image is unlabeled,
  
  - moving image of shape $(\text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})$
  
  - fixed image of shape $(\text{f\_dim1}, \text{f\_dim2}, \text{f\_dim3})$

- When the image is labeled and there is only one label,
  
  - moving image of shape $(\text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})$
  
  - fixed image of shape $(\text{f\_dim1}, \text{f\_dim2}, \text{f\_dim3})$
  
  - moving label of shape $(\text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})$
  
  - fixed label of shape $(\text{f\_dim1}, \text{f\_dim2}, \text{f\_dim3})$

- When the image is labeled and there are multiple labels,
  
  - moving image of shape $(\text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})$

3.13. Dataset Loader
DeepReg

– fixed image of shape \((f_{\text{dim1}}, f_{\text{dim2}}, f_{\text{dim3}})\)
– moving label of shape \((m_{\text{dim1}}, m_{\text{dim2}}, m_{\text{dim3}}, \text{num_labels})\)
– fixed label of shape \((f_{\text{dim1}}, f_{\text{dim2}}, f_{\text{dim3}}, \text{num_labels})\)

Sampling

For paired images, one epoch of the dataset iterates all the image pairs sequentially with random orders. So each image pair is sampled once in each epoch with equal chance. For validation or testing, the random seed is fixed to ensure consistency.

When an image has multiple labels, e.g., the segmentation of different organs in a CT image, only one label will be sampled during training. In particular, only corresponding labels will be sampled between a pair of moving and fixed images. In case of validation or testing, instead of sampling one label per image, all labels will be iterated.

Configuration

An example configuration for paired dataset is provided as follows.

```
dataset:
  dir:
    train: "data/test/h5/paired/train" # folder contains training data
    valid: "data/test/h5/paired/test" # folder contains validation data
    test: "data/test/h5/paired/test" # folder contains test data
  format: "nifti" # value should be nifti / h5
  type: "paired" # value should be paired / unpaired / grouped
  labeled: true # value should be true / false
  moving_image_shape: [16, 16, 16] # value should be like [dim1, dim2, dim3]
  fixed_image_shape: [8, 8, 8] # value should be like [dim1, dim2, dim3]
```

where, the configuration can be split into common configurations that shared by all dataset types and specific configurations for paired images:

- **Common configurations**
  - `dir/train` gives the directory containing training data. Same for `dir/valid` and `dir/test`.
  - `format` can only be Nifti or h5 currently.
  - `type` can be paired, unpaired or grouped, corresponding to the dataset type described above.
  - `labeled` is a boolean indicating if the data is labeled or not.

- **Paired images configurations**
  - `moving_image_shape` is the shape of moving images, a list of three integers.
  - `fixed_image_shape` is the shape of fixed images, a list of three integers.

Optionally, multiple dataset directories can be specified, such that the data will be sampled from several directories, for instance:

```
dataset:
  dir:
    train: # folder contains training data
    - "data/test/h5/paired/train1"
    - "data/test/h5/paired/train2"
    valid: "data/test/h5/paired/test" # folder contains validation data
    test: "data/test/h5/paired/test" # folder contains test data
```
This is particularly useful when performing an experiment such as cross-validation.

**File loader**

For paired data, the specific requirements for data stored in Nifti and h5 files are described as follows.

**Nifti**

Nifti data are stored in files with suffix `.nii.gz`. Each file should contain only one 3D or 4D tensor, corresponding to an image or a label.

`obs` is short for one observation of a data sample - a 3D image volume or a 3D/4D label volume - and the name can be any string.

All image data should be placed under `moving_images/`, `fixed_images/` with respect to the provided directory. The label data should be placed under `moving_labels/`, and `fixed_labels/`, if available. These are top directories.

File names should be consistent between top directories, e.g.:

- `moving_images/`
  - `obs1.nii.gz`
  - `obs2.nii.gz`
  - ...
- `fixed_images/`
  - `obs1.nii.gz`
  - `obs2.nii.gz`
  - ...
- `moving_labels/`
  - `obs1.nii.gz`
  - `obs2.nii.gz`
  - ...
- `fixed_labels/`
  - `obs1.nii.gz`
  - `obs2.nii.gz`
  - ...

Check test paired Nifti data as an example.

Optionally, the data may not be all saved directly under the top directory. They can be further grouped in subdirectories as long as the data paths are consistent.
H5 data are stored in files with suffix .h5. Hierarchical multi-level indexing is not used. Each file should contain multiple key-value pairs and values are 3D or 4D tensors. Each file is equivalent to a top folder in Nifti cases.

All image data should be stored in moving_images.h5, fixed_images.h5. The label data should be stored in moving_labels.h5, and fixed_labels.h5, if available.

The keys should be consistent between files, e.g.:

- **moving_images.h5** has keys:
  - “obs1”
  - “obs2”
  - ...
- **fixed_images.h5** has keys:
  - “obs1”
  - “obs2”
  - ...
- **moving_labels.h5** has keys:
  - “obs1”
  - “obs2”
  - ...
- **fixed_labels.h5** has keys:
  - “obs1”
  - “obs2”
  - ...

Check [test paired H5 data](#) as an example.

### 3.13.4 Unpaired images

For unpaired images, all images are considered as independent and they must have the same shape. Optionally, there are corresponding labels for the images.

Specifically,

- When the image is unlabeled,
  - image of shape (dim1, dim2, dim3)
- When the image is labeled and there is only one label,
  - image of shape (dim1, dim2, dim3)
  - label of shape (dim1, dim2, dim3)
- When the image is labeled and there are multiple labels,
  - image of shape (dim1, dim2, dim3)
  - label of shape (dim1, dim2, dim3, num_labels)
**Sampling**

During each epoch, image pairs will be sampled without replacement. Therefore, given N images, one epoch will thereby have floor(N / 2) image pairs. For validation or testing, the random seed is fixed to ensure consistency.

In case of multiple labels, the sampling method is the same as in *paired data*. In particular, the only corresponding label pairs will be sampled between the two sampled images.

**Configuration**

An example configuration for unpaired dataset is provided as follows.

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train" # folder contains training data
    valid: "data/test/h5/paired/test" # folder contains validation data
    test: "data/test/h5/paired/test" # folder contains test data
  format: "nifti" # value should be nifti / h5
  type: "unpaired" # value should be paired / unpaired / grouped
  labeled: true # value should be true / false
  image_shape: [16, 16, 16] # value should be like [dim1, dim2, dim3]
```

where

- Common configurations
  - Same as *paired images*.

- Unpaired images configurations
  - `image_shape` is the shape of images, a list of three integers.

**File loader**

For unpaired data, the specific requirements for data stored in nifti and h5 files are described as follows.

**Nifti**

Nifti data are stored in files with suffix `.nii.gz` or `.nii`. Each file must contain only one 3D or 4D tensor, corresponding to an image or a label.

The `obs` is short for one observation of a data sample - a 3D image volume or a 3D/4D label volume - and the name can be any string.

All image data should be placed under `images/`. The label data should be placed under `labels/`, if available. These are *top* directories.

File names should be consistent between top directories, e.g.:

- images/
  - obs1.nii.gz
  - obs2.nii.gz
  - ...
- labels/
DeepReg

- obs1.nii.gz
- obs2.nii.gz
- ...

Check **test unpaired Nifti data** as an example.

**H5**

F5 data are stored in files with suffix `.h5`. Hierarchical multi-level indexing is not used. Each file should contain multiple key-value pairs and values are 3D or 4D tensors. Each file is equivalent to a top folder in Nifti cases.

All image data should be placed under `images.h5`. The label data should be placed under `labels.h5`, if available. The keys should be consistent between files, e.g.:

- `images.h5` has keys:
  - “obs1”
  - “obs2”
  - ...
- `labels.h5` has keys:
  - “obs1”
  - “obs2”
  - ...

Check **test unpaired H5 data** as an example.

### 3.13.5 Grouped images

For grouped images, images may not be paired but organized into multiple groups. Each group must have at least two images.

The requirements are the same as unpaired images. Specifically,

- When the image is unlabeled,
  - image of shape (dim1, dim2, dim3)
- When the image is labeled and there is only one label,
  - image of shape (dim1, dim2, dim3)
  - label of shape (dim1, dim2, dim3)
- When the image is labeled and there are multiple labels,
  - image of shape (dim1, dim2, dim3)
  - label of shape (dim1, dim2, dim3, num_labels)
Sampling

For sampling image pairs, DeepReg provides the following options:

- **inter-group sampling**, where the moving image and fixed image come from different groups.
- **intra-group sampling**, where the moving image and fixed image come from the same group.
- **mixed sampling**, where the image pairs are mixed from inter-group sampling and intra-group sampling.

For validation or testing, the random seed is fixed to ensure consistency.

In case of multiple labels, the sampling method is the same as paired data. In particular, only the corresponding label pairs will be sampled between the two sampled images.

Intra-group

To form image pairs, the group and image are sampled sequentially at two stages,

1. Sample a group from which the moving and fixed images will be sampled.
2. Sample two different images from the group as moving and fixed images. When sampling images from the same group, there are multiple options, denoted by `intra_group_option`:
   - **forward**: the moving image always has a smaller image index than fixed image.
   - **backward**: the moving image always has a larger image index than fixed image.
   - **unconstrained**: no constraint on the image index as long as the two images are different.

Therefore, each epoch generates the same number of image pairs as the number of groups, where all groups will be first shuffled and iterated. The `intra_group_option` is useful in implementing temporal-order sensitive sampling strategy.

Inter-group

To form image pairs, the group and image are sampled sequentially at two stages,

1. Sample the first group, from which the moving image will be sampled.
2. Sample the second group, from which the fixed image will be sampled.
3. Sample an image from the first group as moving image.
4. Sample an image from the second group as fixed image.

Therefore, each epoch generates the same number of image pairs as the number of groups, where all groups will be first shuffled and iterated.

Mixed

Optionally, it is possible to mix inter-group and intra-group sampling by specifying the intra-group image sampling probability `intra_group_prob=[0,1]`. The value 0 means entirely inter-group sampling and 1 means entirely intra-group sampling.

Given 0<p<1, when generating intra-group pairs, there is (1-p)*100% chance to sample the fixed images from a different group, after sampling the moving image from the current intra-group images.
Iterated

Optionally, it is possible to generate all combinations of inter-/intra-group image pairs, with `sample_image_in_group` set to false. This is originally designed for evaluation. Mixing inter-/intra-group sampling is not supported with `sample_image_in_group` set to false.

Configuration

An example configuration for grouped dataset is provided as follows.

```yaml
dataset:
  dir:
    train: "data/test/h5/paired/train"  # folder contains training data
    valid: "data/test/h5/paired/test"   # folder contains validation data
    test: "data/test/h5/paired/test"   # folder contains test data
  format: "nifti"  # value should be nifti / h5
  type: "unpaired" # value should be paired / unpaired / grouped
  labeled: true  # value should be true / false
  intra_group_prob: 1 # probability of intra-group sampling, value should be between 0 and 1
  intra_group_option: "forward"  # option for intra-group sampling, value should be forward / backward / unconstrained
  sample_image_in_group: true  # true if sampling one image per group, value should be true / false
  image_shape: [16, 16, 16] # value should be like [dim1, dim2, dim3]
```

where

- **Common configurations**
  
  Same as *paired images*.

- **Grouped images configurations**

  - `intra_group_prob`, a value between 0 and 1, 0 is for inter-group only and 1 is for intra-group only.
  
  - `intra_group_option`, forward or backward or unconstrained, as described above.
  
  - `sample_image_in_group`, true if sampling one image at a time per group, false if generating all possible pairs.

File loader

For grouped data, the specific requirements for data stored in Nifti and h5 files are described as follows.

**Nifti**

Nifti data are stored in files with suffix `.nii.gz`. Each file should contain only one 3D or 4D tensor, corresponding to an image or a label.

`obs` is short for one observation of a data sample - a 3D image volume or a 3D/4D label volume - and the name can be any string.

All image data should be placed under `images/`. The label data should be placed under `labels/`, if available. These are *top* directories.
The leaf directories will be considered as different groups, and file names should be consistent between top directories, e.g.:

- **images**
  - group1
    - obs1.nii.gz
    - obs2.nii.gz
    - ...
  - ...

- **labels**
  - group1
    - obs1.nii.gz
    - obs2.nii.gz
    - ...
  - ...

Check test grouped Nifti data as an example.

### H5

H5 data are stored in files with suffix `.h5`. Hierarchical multi-level indexing is not used. Each file should contain multiple key-value pairs and values are 3D or 4D tensors. Each file is equivalent to a top folder in Nifti cases.

All image data should be placed under `images.h5`. The label data should be placed under `labels.h5`, if available.

The keys must satisfy a specific format, `group-%d-%d`, where `%d` represents an integer number. The first number corresponds to the group index, and the second number corresponds to the observation index. For example, `group-3-2` corresponds to the second observation from the third group.

The keys should be consistent between files, e.g.:

- **images.h5** has keys:
  - “group-1-1”
  - “group-1-2”
  - ...
  - “group-2-1”
  - ...

- **labels.h5** has keys:
  - “group-1-1”
  - “group-1-2”
  - ...
  - “group-2-1”
  - ...

Check test grouped H5 data as an example.
3.14 Registry

This is a functionality under active development. A full documentation will be released later. DeepReg is adopting the usage of the registry system recently, to facilitate the usage of custom functionalities including

- neural network architecture
- loss
- optimizer
- data pre-processing
- data loader

The registry is defined in deepreg/registry.py, where the class Registry maintains a dictionary mapping (category, key) to value. It also provides the build_from_config functionality, which allows creating a class instance from a config directly. This allows for the simplification of the config file such that each configuration only needs to provide the name of the class and the arguments to build the class for use.

With the registry, when developing new classes inside DeepReg, we should use the corresponding decorator, so that the class is registered. Please check deepreg/model/backbone/u_net.py as an example. Moreover, the corresponding __init__.py (maybe multiple ones) should be modified so that these classes will be automatically registered when executing import deepreg. For defining custom classes outside DeepReg, more detailed tutorial will be released later.

For now, we only support custom classes of

- backbone
- loss

3.15 Experimental Features

DeepReg provides some experimental features. These are still in development with variable levels of readiness.

The following tutorials provide an overview of these features. To submit feedback, open a new issue.

- Label sampling

3.15.1 Label sampling

Images may have multiple labels, such as with segmentation of different organs in CT scans. In this case, for each sampled image pair, one label pair is randomly chosen by default.

**Corresponding label pairs**

When using multiple labels, ensure the labels are ordered correctly. label_idx in [width, height, depth, label_idx] must be the same anatomical or pathological structure; a corresponding label pair between the moving and fixed labels.
**Consistent label pairs**

Consistent label pairs between a pair of moving and fixed labels requires:

1. The two images have the same number of labels, and  
2. The labels have the same order  

When a pair of moving and fixed images have inconsistent label pairs, label dissimilarity cannot be defined. The following applies:

- When using the unpaired-labeled-image loader, consistent label pairs are required;  
- When using the grouped-labeled-image loader, consistent label pairs are required between intra-group image pairs;  
- When mixing intra-inter-group images in the grouped-labeled-image loader, consistent label pairs are required between all intra-group and inter-group image pairs.  

However,

- When using the paired-labeled-image loader, consistent label pairs are not required between different image pairs;  
- When using the grouped-labeled-image loader without mixing intra-group and inter-group images, consistent label pairs are not required between different image groups.  

**Partially labeled image data**

When one of the label dissimilarity measures prevents accidentally missing labels. When appropriate, enable training with missing labels with a placeholder all-zero mask for the imaging data.  

**Option for iterating all available label pairs**

This option is default for testing. All the label pairs will be sampled once for each sampled image pair. This option is not supported when mixing intra-group and inter-group image pairs.  

### 3.16 Entry Point

#### 3.16.1 Train

Module to train a network using init files and a CLI.

```python
deeprg.train.build_config(config_path: (str, list), log_root: str, log_dir: str, ckpt_path: str, max_epochs: int = -1) -> (dict, str)
```

Function to initialise log directories, assert that checkpointed model is the right type and to parse the configuration for training.

**Parameters**

- `config_path` – list of str, path to config file  
- `log_root` – root of logs  
- `log_dir` – path to where training logs to be stored.  
- `ckpt_path` – path where model is stored.  
- `max_epochs` – if max_epochs > 0, use it to overwrite the configuration
Returns
- config: a dictionary saving configuration
- log_dir: the path of directory to save logs

deepreg.train.main(args=None)
Entry point for train script.

Parameters args –
Function to train a model.

Parameters
- gpu – which local gpu to use to train
- config_path – path to configuration set up
- gpu_allow_growth – whether to allocate whole GPU memory for training
- ckpt_path – where to store training checkpoints
- log_root – root of logs
- log_dir – where to store logs in training
- max_epochs – if max_epochs > 0, will use it to overwrite the configuration
- registry – registry to construct class objects

3.16.2 Predict
Module to perform predictions on data using command line interface.
depreg.predict.build_config(config_path: (class 'str'), class 'list'), log_root: str, log_dir: str, ckpt_path: str) -> [class 'dict'], class 'str']
Function to create new directory to log directory to store results.

Parameters
- config_path – string or list of strings, path of configuration files
- log_root – str, root of logs
- log_dir – string, path to store logs.
- ckpt_path – str, path where model is stored.

Returns
- config, configuration dictionary
- log_dir, path of the directory for saving outputs

deepreg.predict.build_pair_output_path(indices: list, save_dir: str) -> (class 'str'), class 'str')
Create directory for saving the paired data

Parameters
- indices – indices of the pair, the last one is for label
- save_dir – directory of output
Returns

• save_dir, str, directory for saving the moving/fixed image
• label_dir, str, directory for saving the rest outputs

depreg.predict.main(args=None)
Entry point for predict script.

Parameters args –

Function to predict some metrics from the saved model and logging results.

Parameters

• gpu – which env gpu to use.
• gpu_allow_growth – whether to allow gpu growth or not
• ckpt_path – where model is stored, should be like log_folder/save/ckpt-x
• mode – train / valid / test, to define which split of dataset to be evaluated
• batch_size – int, batch size to perform predictions in
• log_dir – path to store logs
• log_root – folder name to store logs
• sample_label – sample/all, not used
• save_nifti – if true, outputs will be saved in nifti format
• save_png – if true, outputs will be saved in png format
• config_path – to overwrite the default config
• registry – registry to construct class objects

Function to predict results from a dataset from some model

Parameters

• dataset – where data is stored
• fixed_grid_ref – shape=(1, f_dim1, f_dim2, f_dim3, 3)
• model – model to be used for prediction
• model_method – ddf / dvf / affine / conditional
• save_dir – path to store dir
• save_nifti – if true, outputs will be saved in nifti format
• save_png – if true, outputs will be saved in png format
3.16.3 Warp

Module to warp a image with given ddf. A CLI tool is provided.

deeprg.warp.main(args=None)
   Entry point for warp script.

Parameters: args

deeprg.warp.shape_sanity_check(image: numpy.ndarray, ddf: numpy.ndarray)
   Verify image and ddf shapes are consistent and correct.

Parameters:
  - image: a numpy array of shape (m_dim1, m_dim2, m_dim3) or (m_dim1, m_dim2, m_dim3, ch)
  - ddf: a numpy array of shape (f_dim1, f_dim2, f_dim3, 3)

deepreg.warp.warp(image_path: str, ddf_path: str, out_path: str)

Parameters:
  - image_path: file path of the image file
  - ddf_path: file path of the ddf file
  - out_path: file path of the output

3.17 Dataset Loader

3.17.1 Paired Loader

Load paired image data. Supported formats: h5 and Nifti. Image data can be labeled or unlabeled.

class deeprg.dataset.loader.paired_loader.PairedDataLoader(file_loader, data_dir_paths: List[str], labeled: bool, sample_label: str, seed, moving_image_shape: <class 'list'>, fixed_image_shape: <class 'tuple'>)

Load paired data using given file loader. The function sample_index_generator needs to be defined for the GeneratorDataLoader class.

Parameters:
  - file_loader
  - data_dir_paths: path of the directories storing data, the data has to be saved under four different sub-directories: moving_images, fixed_images, moving_labels, fixed_labels
  - labeled: true if the data are labeled
  - sample_label:
  - seed:
• **moving_image_shape** – (width, height, depth)
• **fixed_image_shape** – (width, height, depth)

`sample_index_generator()`
Generate indexes in order to load data using the `GeneratorDataLoader` class.

`validate_data_files()`
Verify all loaders have the same files.

### 3.17.2 Unpaired Loader

Load unpaired data. Supported formats: h5 and Nifti. Image data can be labeled or unlabeled.

```python
class deepreg.dataset.loader.unpaired_loader.UnpairedDataLoader
    (file_loader,
     data_dir_paths: List[str], labeled: bool,
     sample_label: str, seed: int,
     image_shape: (class 'list',<class 'tuple'>))
```

Load unpaired data using given file loader. Handles both labeled and unlabeled cases. The function `sample_index_generator` needs to be defined for the `GeneratorDataLoader` class.

Load data which are unpaired, labeled or unlabeled.

**Parameters**

- **file_loader** –
- **data_dir_paths** – paths of the directories storing data, the data are saved under four different sub-directories: images, labels
- **labeled** – whether the data is labeled.
- **sample_label** –
- **seed** –
- **image_shape** – (width, height, depth)

`close()`
Close the moving files opened by the `file_loaders`.

`sample_index_generator()`
Generates sample indexes to load data using the `GeneratorDataLoader` class.

`validate_data_files()`
Verify all loader have the same files. Since fixed and moving loaders come from the same `file_loader`, there is no need to check both (avoid duplicate).
3.17.3 Grouped Loader


class deepreg.dataset.loader.grouped_loader.GroupedDataLoader (file_loader, data_dir_paths: List[str], labeled: bool, sample_label: (<class 'str'>, None), intra_group_prob: float, intra_group_option: str, sample_image_in_group: bool, seed: (<class 'int'>, None), image_shape: (<class 'list'>, <class 'tuple'>))

Load grouped data.

Yield indexes of images to load using sample_index_generator from GeneratorDataLoader. AbstractUnpairedLoader handles different file formats

Parameters

- **file_loader** – a subclass of FileLoader
- **data_dir_paths** – paths of the directory storing data, the data has to be saved under two different sub-directories:
  - images
  - labels
- **labeled** – bool, true if the data is labeled, false if unlabeled
- **sample_label** – “sample” or “all”, read get_label_indices in deepreg/dataset/util.py for more details.
- **intra_group_prob** – float between 0 and 1,
  - 0 means generating only inter-group samples,
  - 1 means generating only intra-group samples
- **intra_group_option** – str, “forward”, “backward, or “unconstrained”
- **sample_image_in_group** – bool,
  - if true, only one image pair will be yielded for each group, so one epoch has num_groups pairs of data,
  - if false, iterate through this loader will generate all possible pairs
- **seed** – controls the randomness in sampling, if seed=None, then the randomness is not fixed
• **image_shape** – list or tuple of length 3, corresponding to (dim1, dim2, dim3) of the 3D image

close()
Close file loaders

**get_inter_sample_indices() → list**
Calculate the sample indices for inter-group sampling The index to identify a sample is (group1, image1, group2, image2), means

- image1 of group1 is moving image
- image2 of group2 is fixed image

All pairs of images in the dataset are registered. Assuming group i has ni images and that N=[n1, n2, ..., nI], then in total the number of samples are: \(\text{sum(N) * (sum(N)-1) - sum( N * (N-1) )}\)

**Returns** a list of sample indices

**get_intra_sample_indices() → list**
Calculate the sample indices for intra-group sampling The index to identify a sample is (group1, image1, group2, image2), means - image1 of group1 is moving image - image2 of group2 is fixed image

Assuming group i has ni images, then in total the number of samples are - \(\text{sum( ni * (ni-1) / 2 ) for forward/backward}\) - \(\text{sum( ni * (ni-1) ) for unconstrained}\)

**Returns** a list of sample indices

**sample_index_generator()**
Yield (moving_index, fixed_index, image_indices) sequentially, where

- moving_index = (group1, image1)
- fixed_index = (group2, image2)
- image_indices = [group1, image1, group2, image2]

**validate_data_files()**
If the data are labeled, verify image loader and label loader have the same files.

### 3.18 File Loader

#### 3.18.1 Interface

```python
class deepreg.dataset.loader.interface.FileLoader(dir_paths: list, name: str, grouped: bool)
```

Interface / abstract class to load data from multiple directories.

**Parameters**

- **dir_paths** – path to the directory of the data set
- **name** – name is used to identify the subdirectories or file names
- **grouped** – true if the data is grouped

**close()**
Close opened file handles if exist.

**get_data(index: (int, tuple))**
Get one data array by specifying an index.

**Parameters** index – the data index which is required
• for paired or unpaired, the index is one single int, data_index
• for grouped, the index is a tuple of two ints, (group_index, in_group_data_index)

Returns the data array at the specified index

get_data_ids () → List[str]
Return the unique IDs of the data in this data set. This function is used to verify the consistency between moving and fixed images and label.

get_num_groups () → int
Return the number of groups in grouped data set.

Returns int, number of groups in this data set, if grouped

get_num_images () → int
Return the number of image in this data set.

Returns int, number of images in this data set

get_num_images_per_group () → List[int]
Return the number of images in each group. Each group must have at least one image.

Returns a list of integers, representing the number of images in each group.

set_data_structure ()
Store the data structure in memory to retrieve data using data_index.

set_group_structure ()
In addition to set_data_structure, store the group structure in the group_struct so that group_struct[group_index] = list of data_index and data can be retrieved data by data_index = group_struct[group_index][in_group_data_index]

3.18.2 Nifti Loader

class deepreg.dataset.loader.nifti_loader.NiftiFileLoader (dir_paths: List[str],
name: str, grouped: bool)

Generalized loader for nifti files.

Init.

Parameters

• dir_paths – path of directories having nifti files.
• name – name is used to identify the subdirectories.
• grouped – whether the data is grouped.

close ()
Close opened files.

get_data (index: (‘int’, ‘tuple’)) → numpy.ndarray
Get one data array by specifying an index

Parameters index – the data index which is required

• for paired or unpaired, the index is one single int, data_index
• for grouped, the index is a tuple of two ints, (group_index, in_group_data_index)

Returns arr the data array at the specified index
get_data_ids() \rightarrow \text{List}[\text{str}]

Return the unique IDs of the data in this data set this function is used to verify the consistency between images and label, moving and fixed.

Returns data_path_splits but without suffix

get_num_images() \rightarrow \text{int}

Returns \text{int}, number of images in this data set

set_data_structure()

Store the data structure in the memory so that we can retrieve data using data_index this function sets data_path_splits, a list of string tuples to identify path of data

• if grouped, a split is (dir_path, group_path, file_name, suffix) data is stored in dir_path/name/group_path/file_name.suffix
• if not grouped, a split is (dir_path, file_name, suffix) data is stored in dir_path/name/file_name.suffix

set_group_structure()

In addition to set_data_structure store the group structure in the group_struct so that group_struct[group_index] = list of data_index we can retrieve data using (group_index, in_group_data_index) data_index = group_struct[group_index][in_group_data_index]

deeperg.dataset.loader.nifti_loader.load_nifti_file(file_path: str) \rightarrow \text{numpy.ndarray}

Parameters file_path – path of a Nifti file with suffix .nii or .nii.gz

Returns return the numpy array

3.18.3 H5 Loader

Load h5 files and associated information.

class deeperg.dataset.loader.h5_loader.H5FileLoader(dir_paths: \text{List}[\text{str}], name: \text{str}, grouped: \text{bool})

Generalized loader for h5 files.

Init.

Parameters

• dir_paths – path of h5 files.
• name – name is used to identify the file names.
• grouped – whether the data is grouped.

close()

Close opened h5 file handles.

get_data(index: (\text{<class 'int'>}, \text{<class 'tuple'>})) \rightarrow \text{numpy.ndarray}

Get one data array by specifying an index

Parameters index – the data index which is required

• for paired or unpaired, the index is one single int, data_index
• for grouped, the index is a tuple of two ints, (group_index, in_group_data_index)

Returns arr the data array at the specified index

generate_data_ids() \rightarrow \text{List}[\text{str}]

Get the unique IDs of data in this data set to verify consistency between images and label, moving and fixed.
**DeepReg**

**Returns** data_path_splits as the data can be identified using dir_path and data_key

```
get_num_images() \rightarrow int
```

**Returns** int, number of images in this data set

```
set_data_structure()
```

Store the data structure in memory so that we can retrieve data using data_index. This function sets two attributes:

- h5_files, a dict such that h5_files[dir_path] = opened h5 file handle
- data_path_splits, a list of string tuples to identify path of data
  - if grouped, a split is (dir_path, group_name, data_key) such that data = h5_files[dir_path]["group-
    {group_name}-{data_key}"]
  - if not grouped, a split is (dir_path, data_key) such that data = h5_files[dir_path][data_key]

```
set_group_structure()
```

Similar to NiftiLoader as the first two tokens of a split forms a group_id. Store the group structure in group_struct so that group_struct[group_index] = list of data_index. Retrieve data using (group_index,
in_group_data_index). data_index = group_struct[group_index][in_group_data_index].

### 3.19 Network

#### 3.19.1 DDF / DVF Network

#### 3.19.2 Conditional Network

#### 3.19.3 Affine Network

#### 3.19.4 Utils

### 3.20 Network Backbone

#### 3.20.1 Local Net

```
class deepreg.model.backbone.local_net.LocalNet(*args: Any, **kwargs: Any)
```

Build LocalNet for image registration.

**Reference:**

  for multimodal image registration.”
  for multimodal deformable image registration,”
  https://arxiv.org/abs/1711.01666

Image is encoded gradually, i from level 0 to E, then it is decoded gradually, j from level E to D. Some of the decoded levels are used for generating extractions.

So, extract_levels are between [0, E] with E = max(extract_levels), and D = min(extract_levels).
3.20. Network Backbone

Parameters

- **image_size** – such as (dim1, dim2, dim3)
- **out_channels** – number of channels for the extractions
- **num_channel_initial** – number of initial channels.
- **extract_levels** – number of extraction levels.
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.
- **control_points** – specify the distance between control points (in voxels).
- **kwargs** – additional arguments.

**call** (inputs: tensorflow.Tensor, training=None, mask=None) ➞ tensorflow.Tensor

Build LocalNet graph based on built layers.

Parameters

- **inputs** – image batch, shape = (batch, f_dim1, f_dim2, f_dim3, ch)
- **training** – None or bool.
- **mask** – None or tf.Tensor.

**Returns** shape = (batch, f_dim1, f_dim2, f_dim3, out_channels)

3.20.2 Global Net

**class** deepreg.model.backbone.global_net.GlobalNet (*args: Any, **kwargs: Any)

Build GlobalNet for image registration.

Reference:

Image is encoded gradually, i from level 0 to E. Then, a densely-connected layer outputs an affine transformation.

Parameters

- **image_size** – tuple, such as (dim1, dim2, dim3)
- **out_channels** – int, number of channels for the output
- **num_channel_initial** – int, number of initial channels
- **extract_levels** – list, which levels from net to extract
- **out_kernel_initializer** – str, which kernel to use as initializer
- **out_activation** – str, activation at last layer
- **kwargs** – additional arguments.

**call** (inputs: tensorflow.Tensor, training=None, mask=None) ➞ tensorflow.Tensor

Build GlobalNet graph based on built layers.

Parameters

- **inputs** – image batch, shape = (batch, f_dim1, f_dim2, f_dim3, ch)
- **training** – None or bool.
• **mask** – None or tf.Tensor.

Returns shape = (batch, dim1, dim2, dim3, 3)

### 3.20.3 U-Net

**class** deepreg.model.backbone.u_net.UNet(*args: Any, **kwargs: Any)  
Class that implements an adapted 3D UNet.

Reference:


Initialise UNet.

**Parameters**

- **image_size** – (dim1, dim2, dim3), dims of input image.
- **out_channels** – number of channels for the output
- **num_channel_initial** – number of initial channels
- **depth** – input is at level 0, bottom is at level depth
- **out_kernel_initializer** – which kernel to use as initializer
- **out_activation** – activation at last layer
- **pooling** – for downsampling, use non-parameterized pooling if true, otherwise use conv3d
- **concat_skip** – when upsampling, concatenate skipped tensor if true, otherwise use addition
- **control_points** – specify the distance between control points (in voxels).
- **kwargs** – additional arguments.

**call** (inputs: tensorflow.Tensor, training=None, mask=None) → tensorflow.Tensor  
Builds graph based on built layers.

**Parameters**

- **inputs** – shape = [batch, f_dim1, f_dim2, f_dim3, in_channels]
- **training** –
- **mask** –

Returns shape = [batch, f_dim1, f_dim2, f_dim3, out_channels]
3.21 Layer

3.21.1 Layer

Activation

```python
class deepreg.model.layer.Activation(*args: Any, **kwargs: Any)
    Layer wraps tf.keras.activations.get().
    Parameters
    • identifier – e.g. “relu”
    • kwargs – additional arguments.
```

AdditiveUpSampling

```python
class deepreg.model.layer.AdditiveUpSampling(*args: Any, **kwargs: Any)
    Layer up-samples 3d tensor and reduce channels using split and sum.
    Parameters
    • output_shape – (out_dim1, out_dim2, out_dim3)
    • stride – int, 1-D Tensor or list
    • kwargs – additional arguments.
    call(inputs, **kwargs) → tensorflow.Tensor
    Parameters
    • inputs – shape = (batch, dim1, dim2, dim3, channels)
    • kwargs – additional arguments.
    Returns shape = (batch, out_dim1, out_dim2, out_dim3, channels//stride]
```

Conv3d

```python
class deepreg.model.layer.Conv3d(*args: Any, **kwargs: Any)
    Layer wraps tf.keras.layers.Conv3D.
    Parameters
    • filters – number of channels of the output
    • kernel_size – int or tuple of 3 ints, e.g. (3,3,3) or 3
    • strides – int or tuple of 3 ints, e.g. (1,1,1) or 1
    • padding – same or valid
    • activation – defines the activation function
    • use_bias – whether add bias to output
    • kernel_initializer – defines the initialization method, defines the initialization method
    • kwargs – additional arguments.
```
DeepReg

**Conv3dBlock**

```python
class deepreg.model.layer.Conv3dBlock(*args: Any, **kwargs: Any)
A conv3d block having conv3d - norm - activation.

Parameters

* filters – number of channels of the output
* kernel_size – int or tuple of 3 ints, e.g. (3,3,3) or 3
* strides – int or tuple of 3 ints, e.g. (1,1,1) or 1
* padding – str, same or valid
* kwargs – additional arguments.

call (inputs, training=None, **kwargs) → tensorflow.Tensor

Parameters

* inputs – shape = (batch, in_dim1, in_dim2, in_dim3, channels)
* training – training flag for normalization layers (default: None)
* kwargs – additional arguments.

Returns shape = (batch, in_dim1, in_dim2, in_dim3, channels)
```

**Conv3dWithResize**

```python
class deepreg.model.layer.Conv3dWithResize(*args: Any, **kwargs: Any)
A layer contains conv3d - resize3d.

Parameters

* output_shape – tuple, (out_dim1, out_dim2, out_dim3)
* filters – int, number of channels of the output
* kernel_initializer – str, defines the initialization method
* activation – str, defines the activation function
* kwargs – additional arguments.

call (inputs, **kwargs) → tensorflow.Tensor

Parameters

* inputs – shape = (batch, dim1, dim2, dim3, channels)
* kwargs – additional arguments.

Returns shape = (batch, out_dim1, out_dim2, out_dim3, channels)
```
**Deconv3d**

class deepreg.model.layer.Deconv3d(*args: Any, **kwargs: Any)
Layer wraps tf.keras.layers.Conv3DTranspose and does not requires input shape when initializing.

Parameters

- **filters** – number of channels of the output
- **output_shape** – (out_dim1, out_dim2, out_dim3)
- **kernel_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – same or valid.
- **use_bias** – use bias for Conv3DTranspose or not.
- **kwargs** – additional arguments.

**Deconv3dBlock**

class deepreg.model.layer.Deconv3dBlock(*args: Any, **kwargs: Any)
A deconv3d block having deconv3d - norm - activation.

Parameters

- **filters** – number of channels of the output
- **output_shape** – (out_dim1, out_dim2, out_dim3)
- **kernel_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – str, same or valid
- **kwargs** – additional arguments.

call(inputs, training=None, **kwargs) → tensorflow.Tensor

Parameters

- **inputs** – shape = (batch, in_dim1, in_dim2, in_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** – additional arguments.

Return output shape = (batch, in_dim1, in_dim2, in_dim3, channels)

**Dense**

class deepreg.model.layer.Dense(*args: Any, **kwargs: Any)
Layer wraps tf.keras.layers.Dense and flattens input if necessary.

Parameters

- **units** – number of hidden units
- **bias_initializer** – str, default “zeros”
- **kwargs** – additional arguments.
DeepReg

\[ \text{call}(\text{inputs}, **\text{kwargs}) \rightarrow \text{tensorflow.Tensor} \]

**Parameters**

- **inputs** – shape = (batch, *vol_dim, channels)
- **kwargs** – (not used)

**Returns**  shape = (batch, units)

**DownSampleResnetBlock**

```python
class deepreg.model.layer.DownSampleResnetBlock(*args: Any, **kwargs: Any)
A down-sampling resnet conv3d block, with max-pooling or conv3d.
1. conved = conv3d_block(inputs) # adjust channel
2. skip = residual_block(conved) # develop feature
3. pooled = pool(skip) # down-sample
```

**Parameters**

- **filters** – number of channels of the output
- **kernel_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **pooling** – if True, use max pooling to downsample, otherwise use conv.
- **kwargs** – additional arguments.

\[ \text{call}(\text{inputs}, \text{training}=\text{None}, **\text{kwargs}) \]

**Parameters**

- **inputs** – shape = (batch, in_dim1, in_dim2, in_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** – additional arguments.

**Returns**

(pooled, skip)

- downsampled, shape = (batch, in_dim1//2, in_dim2//2, in_dim3//2, channels)
- skipped, shape = (batch, in_dim1, in_dim2, in_dim3, channels)

**IntDVF**

```python
class deepreg.model.layer.IntDVF(*args: Any, **kwargs: Any)
Layer calculates DVF from DDF.
```

**Reference:**

- integrate_vec of neuron https://github.com/adalca/neurite/blob/legacy/neuron/utils.py

**Parameters**

- **fixed_image_size** – tuple, (f_dim1, f_dim2, f_dim3)
- **num_steps** – int, number of steps for integration
• **kwargs – additional arguments.

    call (inputs, **kwargs) \rightarrow \text{tensorflow.Tensor}

    Parameters

    • inputs – dvf, shape = (batch, f_dim1, f_dim2, f_dim3, 3), type = float32
    • **kwargs – additional arguments.

    Returns ddf, shape = (batch, f_dim1, f_dim2, f_dim3, 3)

**LocalNetResidual3dBlock**

    class deepreg.model.layer.LocalNetResidual3dBlock (*args: Any, **kwargs: Any)

    A resnet conv3d block, simpler than Residual3dBlock.

    1. conved = conv3d(inputs)
    2. out = act(norm(conved) + inputs)

    Parameters

    • filters – number of channels of the output
    • kernel_size – int or tuple of 3 ints, e.g. (3,3,3) or 3
    • strides – int or tuple of 3 ints, e.g. (1,1,1) or 1
    • **kwargs – additional arguments.

**LocalNetUpSampleResnetBlock**

    class deepreg.model.layer.LocalNetUpSampleResnetBlock (*args: Any, **kwargs: Any)

    Layer up-samples tensor with two inputs (skipped and down-sampled).

    Parameters

    • filters – int, number of output channels
    • use_additive_upsampling – bool to used additive upsampling
    • **kwargs – additional arguments.

    build (input_shape)

    Parameters input_shape – tuple (nonskip_tensor_shape, skip_tensor_shape)

    call (inputs, training=None, **kwargs) \rightarrow \text{tensorflow.Tensor}

    Parameters

    • inputs – list = [inputs_nonskip, inputs_skip]
    • training – training flag for normalization layers (default: None)
    • **kwargs – additional arguments.

    Returns 3.21. Layer
MaxPool3d

class deepreg.model.layer.MaxPool3d(*args: Any, **kwargs: Any)
Layer wraps tf.keras.layers.MaxPool3D

Parameters

• pool_size – int or tuple of 3 ints
• strides – int or tuple of 3 ints, if None default will be pool_size
• padding – str, same or valid
• kwargs – additional arguments.

Norm

class deepreg.model.layer.Norm(*args: Any, **kwargs: Any)
Class merges batch norm and layer norm.

Parameters

• name – str, batch_norm or layer_norm
• axis – int
• kwargs – additional arguments.

Residual3dBlock

class deepreg.model.layer.Residual3dBlock(*args: Any, **kwargs: Any)
A resnet conv3d block.

1. conved = conv3d(conv3d_block(inputs))
2. out = act(norm(conved) + inputs)

Parameters

• filters – int, number of filters in the convolutional layers
• kernel_size – int or tuple of 3 ints, e.g. (3,3,3) or 3
• strides – int or tuple of 3 ints, e.g. (1,1,1) or 1
• kwargs – additional arguments.

call(inputs, training=None, **kwargs) → tensorflow.Tensor

Parameters

• inputs – shape = (batch, in_dim1, in_dim2, in_dim3, channels)
• training – training flag for normalization layers (default: None)
• kwargs – additional arguments.

Return output shape = (batch, in_dim1, in_dim2, in_dim3, channels)
UpSampleResnetBlock

class deepreg.model.layer.UpSampleResnetBlock(*args: Any, **kwargs: Any)
An up-sampling resnet conv3d block, with deconv3d.

Parameters

- filters – number of channels of the output
- kernel_size – int or tuple of 3 ints, e.g. (3,3,3) or 3
- concat – bool, specify how to combine input and skip connection images. If True, use concatenation, otherwise use sum (default=False).
- kwargs – additional arguments.

build(input_shape)

Parameters input_shape – tuple, (downsampled_image_shape, skip_image_shape)

call(inputs, training=None, **kwargs) → tensorflow.Tensor

Parameters

- inputs – tuple
  - down-sampled
  - skipped
- training – training flag for normalization layers (default: None)
- kwargs – additional arguments.

Returns shape = (batch, *skip_connection_image_shape, filters]

Warping

class deepreg.model.layer.Warping(*args: Any, **kwargs: Any)
A layer warps an image using DDF.

Reference:

- transform of neuron https://github.com/adalca/neurite/blob/legacy/neurite/utils.py
  where vol = image, loc_shift = ddf

Parameters

- fixed_image_size – shape = (f_dim1, f_dim2, f_dim3) or (f_dim1, f_dim2, f_dim3,
  ch) with the last channel for features
- kwargs – additional arguments.

call(inputs, **kwargs) → tensorflow.Tensor

Parameters

- inputs – (ddf, image)
  - ddf, shape = (batch, f_dim1, f_dim2, f_dim3, 3), dtype = float32
  - image, shape = (batch, m_dim1, m_dim2, m_dim3), dtype = float32
- kwargs – additional arguments.
3.21.2 Util

Module containing utilities for layer inputs

deepreg.model.layer_util.gaussian_filter_3d(kernel_sigma: (list, tuple, int)) → tensorflow.Tensor

Define a gaussian filter in 3d for smoothing.

The filter size is defined 3*kernel_sigma

Parameters kernel_sigma – the deviation at each direction (list) or use an isotropic deviation (int)

Returns kernel: tf.Tensor specify a gaussian kernel of shape: [3*k for k in kernel_sigma]

deepreg.model.layer_util.gen_rand_affine_transform(batch_size: int, scale: float, seed: (int, None) = None) → tensorflow.Tensor

Function that generates a random 3D transformation parameters for a batch of data.

for 3D coordinates, affine transformation is

\[
\begin{bmatrix}
  x' & y' & z' & 1
\end{bmatrix} = \begin{bmatrix}
  x & y & z & 1
\end{bmatrix} * \begin{bmatrix}
  * & * & * & 0 \\
  * & * & * & 0 \\
  * & * & * & 0 \\
  * & * & * & 1
\end{bmatrix}
\]

where each * represents a degree of freedom, so there are in total 12 degrees of freedom the equation can be denoted as

\[
\text{new} = \text{old} * \text{T}
\]

where

• new is the transformed coordinates, of shape (1, 4)
• old is the original coordinates, of shape (1, 4)
• T is the transformation matrix, of shape (4, 4)

the equation can be simplified to

\[
\begin{bmatrix}
  x' & y' & z'
\end{bmatrix} = \begin{bmatrix}
  x & y & z
\end{bmatrix} * \begin{bmatrix}
  * & * & * \\
  * & * & * \\
  * & * & * \\
  * & * & *
\end{bmatrix}
\]

so that

\[
\text{new} = \text{old} * \text{T}
\]

where

• new is the transformed coordinates, of shape (1, 3)
• old is the original coordinates, of shape (1, 4)
• T is the transformation matrix, of shape (4, 3)

Given original and transformed coordinates, we can calculate the transformation matrix using

\[
x = \text{np.linalg.lstsq(a, b)}
\]
such that
\[ a \times = b \]
In our case,
- \( a = \text{old} \)
- \( b = \text{new} \)
- \( x = T \)

To generate random transformation, we choose to add random perturbation to corner coordinates as follows: for corner of coordinates \((x, y, z)\), the noise is
\[-(x, y, z) \ast (r1, r2, r3)\]
where \(ri\) is a random number between \((0, \text{scale})\). So
\[(x', y', z') = (x, y, z) \ast (1-r1, 1-r2, 1-r3)\]
Thus, we can directly sample between 1-scale and 1 instead

We choose to calculate the transformation based on four corners in a cube centered at \((0, 0, 0)\). A cube is shown as below, where
- \(C = (-1, -1, -1)\)
- \(G = (-1, -1, 1)\)
- \(D = (-1, 1, -1)\)
- \(A = (1, -1, -1)\)

```
G -- -- -- -- -- -- -- -- H
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
E -- -- -- -- -- -- -- -- F
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
|     |     |     |
A -- -- -- -- -- -- -- -- B
```

**Parameters**
- `batch_size` – int
- `scale` – a float number between 0 and 1
- `seed` – control the randomness

**Returns** shape = `(batch, 4, 3)`
deepreg.model.layer_util.gen_rand_ddf(batch_size: int, image_size: tuple, field_strength: (class 'tuple', <class 'list'>), low_res_size: (class 'tuple', <class 'list'>), seed: (class 'int', None) = None) → tensorflow.Tensor

Function that generates a random 3D DDF for a batch of data. :param batch_size: :param image_size: :param field_strength: maximum field strength, computed as a U[0,field_strength] :param low_res_size: low_resolution deformation field that will be upsampled to

the original size in order to get smooth and more realistic fields.

Parameters **seed** – control the randomness

Returns
deepreg.model.layer_util.get_n_bits_combinations(num_bits: int) → list

Function returning list containing all combinations of n bits. Given num_bits binary bits, each bit has value 0 or 1, there are in total 2**n_bits combinations.

Parameters **num_bits** – int, number of combinations to evaluate

Returns a list of length 2**n_bits, return[i] is the binary representation of the decimal integer.

Example

```python
>>> from deepreg.model.layer_util import get_n_bits_combinations
>>> get_n_bits_combinations(3)
[[0, 0, 0], # 0
 [0, 0, 1], # 1
 [0, 1, 0], # 2
 [0, 1, 1], # 3
 [1, 0, 0], # 4
 [1, 0, 1], # 5
 [1, 1, 0], # 6
 [1, 1, 1]] # 7
```
deepreg.model.layer_util.get_reference_grid(grid_size: (class 'tuple', <class 'list'>)) → tensorflow.Tensor

Generate a 3D grid with given size.

Reference:

• volshape_to_meshgrid of neuron [https://github.com/adalca/neurite/blob/legacy/neurite/utils.py](https://github.com/adalca/neurite/blob/legacy/neurite/utils.py)

neuron modifies meshgrid to make it faster, however local benchmark suggests tf.meshgrid is better

Note:

for tf.meshgrid, in the 3-D case with inputs of length M, N and P, outputs are of shape (N, M, P) for ‘xy’ indexing and (M, N, P) for ‘ij’ indexing.

Parameters **grid_size** – list or tuple of size 3, [dim1, dim2, dim3]

Returns shape = (dim1, dim2, dim3, 3), grid[i, j, k, :] = [i j k]
deepreg.model.layer_util.pyramid_combination(values: list, weight_floor: list, weight_ceil: list) → tensorflow.Tensor

Calculates linear interpolation (a weighted sum) using values of hypercube corners in dimension n.

For example, when num_dimension = len(loc_shape) = num_bits = 3 values correspond to values at corners of following coordinates
values[:2] correspond to the corners with last coordinate == 0

values[1::2] correspond to the corners with last coordinate == 1

The weights correspond to the floor corners. For example, when num_dimension = len(loc_shape) = num_bits = 3, weight_floor = [f1, f2, f3] (ignoring the batch dimension). weight_ceil = [c1, c2, c3] (ignoring the batch dimension).

So for corner with coords (x, y, z), x, y, z’s values are 0 or 1

- weight for x = f1 if x = 0 else c1
- weight for y = f2 if y = 0 else c2
- weight for z = f3 if z = 0 else c3

so the weight for (x, y, z) is

\[ W_{xyz} = ((1-x) \cdot f1 + x \cdot c1) \cdot ((1-y) \cdot f2 + y \cdot c2) \cdot ((1-z) \cdot f3 + z \cdot c3) \]

Let

\[ W_{xy} = ((1-x) \cdot f1 + x \cdot c1) \cdot ((1-y) \cdot f2 + y \cdot c2) \]

Then

- W_{xy0} = W_{xy} \cdot f3
- W_{xy1} = W_{xy} \cdot c3

Similar to W_{xyz}, denote V_{xyz} the value at (x, y, z), the final sum V equals

\[ \text{sum over } x,y,z \ (V_{xyz} \cdot W_{xyz}) = \text{sum over } x,y \ (V_{xy0} \cdot W_{xy0} + V_{xy1} \cdot W_{xy1}) = \text{sum over } x,y \ (V_{xy0} \cdot W_{xy} \cdot f3 + V_{xy1} \cdot W_{xy} \cdot c3) = \text{sum over } x,y \ (V_{xy0} \cdot W_{xy} \cdot f3 + \text{sum over } x,y \ (V_{xy1} \cdot W_{xy} \cdot c3) \]

That’s why we call this pyramid combination. It calculates the linear interpolation gradually, starting from the last dimension. The key is that the weight of each corner is the product of the weights along each dimension.
Parameters

- **values** – a list having values on the corner, it has $2^n$ tensors of shape (*loc_shape) or (batch, *loc_shape) or (batch, *loc_shape, ch) the order is consistent with get_n_bits_combinations loc_shape is independent from n, aka num_dim

- **weight_floor** – a list having weights of floor points, it has n tensors of shape (*loc_shape) or (batch, *loc_shape) or (batch, *loc_shape, 1)

- **weight ceil** – a list having weights of ceil points, it has n tensors of shape (*loc_shape) or (batch, *loc_shape) or (batch, *loc_shape, 1)

**Returns** one tensor of the same shape as an element in values (*loc_shape) or (batch, *loc_shape) or (batch, *loc_shape, 1)


Sample the volume at given locations.

Input has

- volume, vol, of shape = (batch, v_dim 1, ..., v_dim n), or (batch, v_dim 1, ..., v_dim n, ch), where n is the dimension of volume, ch is the extra dimension as features.

  Denote vol_shape = (v_dim 1, ..., v_dim n)

- location, loc, of shape = (batch, l_dim 1, ..., l_dim m, n), where m is the dimension of output.

  Denote loc_shape = (l_dim 1, ..., l_dim m)

Reference:

- neuron’s interpn https://github.com/adalca/neurite/blob/legacy/neuron/utils.py

Difference

1. they dont have batch size
2. they support more dimensions in vol

TODO try not using stack as neuron claims it’s slower

Parameters

- **vol** – shape = (batch, *vol_shape) or (batch, *vol_shape, ch) with the last channel for features

- **loc** – shape = (batch, *loc_shape, n) such that loc[b, l1, ..., lm, :] = [v1, ..., vn] is of shape (n), which represents a point in vol, with coordinates (v1, ..., vn)

- **interpolation** – linear only, TODO support nearest

- **zero_boundary** – if true, values on or outside boundary will be zeros

**Returns** shape = (batch, *loc_shape) or (batch, *loc_shape, ch)

deeperg.model.layer_util.resize3d(image: tensorflow.Tensor, size: (<class 'tuple'>, <class 'list'>), method: str = tensorflow.image.ResizeMethod.BILINEAR) → tensorflow.Tensor

Tensorflow does not have resize 3d, therefore the resize is performed two folds.

- resize dim2 and dim3
- resize dim1 and dim2
Parameters

- **image** – tensor of shape = (batch, dim1, dim2, dim3, channels) or (batch, dim1, dim2, dim3) or (dim1, dim2, dim3)
- **size** – tuple, (out_dim1, out_dim2, out_dim3)
- **method** – str, one of tf.image.ResizeMethod

Returns

tensor of shape = (batch, out_dim1, out_dim2, out_dim3, channels) or (batch, dim1, dim2, dim3) or (dim1, dim2, dim3)

deeprg.model.layer_util.warp_grid(grid: tensorflow.Tensor, theta: tensorflow.Tensor) \rightarrow tensorflow.Tensor

Perform transformation on the grid.

- grid_padded[i, j, k, :] = [i j k 1]
- grid_warped[b, i, j, k, p] = \sum_{q} (grid_padded[i, j, k, q] \times theta[b, q, p])

Parameters

- **grid** – shape = (dim1, dim2, dim3, 3), grid[i, j, k, :] = [i j k]
- **theta** – parameters of transformation, shape = (batch, 4, 3)

Returns

shape = (batch, dim1, dim2, dim3, 3)


Warp an image with given DDF.

Parameters

- **image** – an image to be warped, shape = (batch, m_dim1, m_dim2, m_dim3) or (batch, m_dim1, m_dim2, m_dim3, ch)
- **ddf** – shape = (batch, f_dim1, f_dim2, f_dim3, 3)
- **grid_ref** – shape = (f_dim1, f_dim2, f_dim3, 3) or (1, f_dim1, f_dim2, f_dim3, 3) if None grid_reg will be calculated based on ddf

Returns

shape = (batch, f_dim1, f_dim2, f_dim3) or (batch, f_dim1, f_dim2, f_dim3, ch)

---

3.22 Loss

3.22.1 Image Loss

Provide different loss or metrics classes for images.

class deeprg.model.loss.image.GlobalMutualInformation(*args: Any, **kwargs: Any)

Differentiable global mutual information via Parzen windowing method.

y_true and y_pred have to be at least 4d tensor, including batch axis.

Reference: [https://dspace.mit.edu/handle/1721.1/123142](https://dspace.mit.edu/handle/1721.1/123142), Section 3.1, equation 3.1-3.5, Algorithm 1

Init.

Parameters
• **num_bins** – number of bins for intensity, the default value is empirical.
• **sigma_ratio** – a hyper param for gaussian function
• **reduction** – using SUM reduction over batch axis, calling the loss like \( \text{loss}(\ y_{true}, \ y_{pred}) \) will return a scalar tensor.
• **name** – name of the loss

**call** (\( y_{true}: \text{tensorflow.Tensor}, \ y_{pred}: \text{tensorflow.Tensor} \)) \( \rightarrow \text{tensorflow.Tensor} \)
Return loss for a batch.

**Parameters**
- **y_true** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, ch)
- **y_pred** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, ch)

**Returns** shape = (batch,)

**get_config**()
Return the config dictionary for recreating this class.

**class** deepreg.model.loss.image.GlobalMutualInformationLoss (*args: Any, **kwargs: Any*)
Revert the sign of GlobalMutualInformation.

Init without required arguments.

**Parameters**
- **kwargs** – additional arguments.

**class** deepreg.model.loss.image.LocalNormalizedCrossCorrelation (*args: Any, **kwargs: Any*)
Local squared zero-normalized cross-correlation.

The loss is based on a moving kernel/window over the \( y_{true}/y_{pred} \), within the window the square of zncc is calculated. The kernel can be a rectangular / triangular / gaussian window. The final loss is the averaged loss over all windows. \( y_{true} \) and \( y_{pred} \) have to be at least 4d tensor, including batch axis.

Reference:
- **Code:** [https://github.com/voxelmorph/voxelmorph/blob/legacy/src/losses.py](https://github.com/voxelmorph/voxelmorph/blob/legacy/src/losses.py)

Init.

**Parameters**
- **kernel_size** – int. Kernel size or kernel sigma for kernel_type='gauss'.
- **kernel_type** – str, rectangular, triangular or gaussian
- **reduction** – using SUM reduction over batch axis, calling the loss like \( \text{loss}(\ y_{true}, \ y_{pred}) \) will return a scalar tensor.
- **name** – name of the loss

**call** (\( y_{true}: \text{tensorflow.Tensor}, \ y_{pred}: \text{tensorflow.Tensor} \)) \( \rightarrow \text{tensorflow.Tensor} \)
Return loss for a batch.

**Parameters**
- **y_true** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, ch)
- **y_pred** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, ch)

**Returns** shape = (batch,)
get_config()
Return the config dictionary for recreating this class.

class deepreg.model.loss.image.LocalNormalizedCrossCorrelationLoss(*args:
Any,
**kwargs:
Any)
Revert the sign of LocalNormalizedCrossCorrelation.

Init without required arguments.

Parameters kwargs – additional arguments.

class deepreg.model.loss.image.SumSquaredDifference(*args: Any, **kwargs: Any)
Sum of squared distance between y_true and y_pred.
y_true and y_pred have to be at least 1d tensor, including batch axis.

Init.

Parameters

• reduction – using SUM reduction over batch axis, calling the loss like loss(y_true,
y_pred) will return a scalar tensor.

• name – name of the loss

call (y_true: tensorflow.Tensor, y_pred: tensorflow.Tensor) → tensorflow.Tensor
Return loss for a batch.

Parameters

• y_true – shape = (batch, ...)

• y_pred – shape = (batch, ...)

Returns shape = (batch,)

depreg.model.loss.image.build_gaussian_kernel(kernel_size: int, input_channel: int)
Return a Gaussian kernel for LocalNormalizedCrossCorrelation.

Parameters

• kernel_size – size of the kernel for convolution.

• input_channel – number of channels for input

Returns

• filters, of shape (kernel_size, kernel_size, kernel_size, ch, 1)

• kernel_vol, scalar

depreg.model.loss.image.build_rectangular_kernel(kernel_size: int, input_channel: int)
Return a rectangular kernel for LocalNormalizedCrossCorrelation.

Parameters

• kernel_size – size of the kernel for convolution.

• input_channel – number of channels for input

Returns

• filters, of shape (kernel_size, kernel_size, kernel_size, ch, 1)

• kernel_vol, scalar
DeepReg

deepreg.model.loss.image.build_triangular_kernel(kernel_size: int, input_channel: int)

Return a triangular kernel for LocalNormalizedCrossCorrelation.

Parameters
- **kernel_size** – size of the kernel for convolution.
- **input_channel** – number of channels for input

Returns
- filters, of shape (kernel_size-1, kernel_size-1, kernel_size-1, ch, 1)
- kernel_vol, scalar

3.22.2 Label Loss

Provide different loss or metrics classes for labels.

class deepreg.model.loss.label.CrossEntropy(*args: Any, **kwargs: Any)

Define weighted cross-entropy.

The formulation is: \[ \text{loss} = \text{pos}_w \cdot \text{y_true} \log(\text{y_pred}) - \text{neg}_w \cdot (1\cdot\text{y_true}) \log(1\cdot\text{y_pred}) \]

Init.

Parameters
- **binary** – if True, project y_true, y_pred to 0 or 1
- **neg_weight** – weight for negative class
- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
- **reduction** – using SUM reduction over batch axis, calling the loss like \( \text{loss}(\text{y_true}, \text{y_pred}) \) will return a scalar tensor.
- **name** – str, name of the loss.

get_config()  
Return the config dictionary for recreating this class.

class deepreg.model.loss.label.DiceLoss(*args: Any, **kwargs: Any)

Revert the sign of DiceScore.

Init without required arguments.

Parameters **kwargs – additional arguments.

class deepreg.model.loss.label.DiceScore(*args: Any, **kwargs: Any)

Define dice score.

The formulation is:

0. \( \text{pos}_w + \text{neg}_w = 1 \)
1. let \( \text{y_prod} = \text{y_true} \cdot \text{y_pred} \) and \( \text{y_sum} = \text{y_true} + \text{y_pred} \)
2. \( \text{num} = 2 \cdot (\text{pos}_w \cdot \text{y_true} \cdot \text{y_pred} + \text{neg}_w \cdot (1\cdot\text{y_true}) \cdot (1\cdot\text{y_pred})) = 2 \cdot ((\text{pos}_w + \text{neg}_w) \cdot \text{y_prod} - \text{neg}_w \cdot \text{y_sum} + \text{neg}_w) = 2 \cdot (\text{y_prod} - \text{neg}_w \cdot \text{y_sum} + \text{neg}_w) \)
3. \( \text{denom} = (\text{pos}_w \cdot (\text{y_true} + \text{y_pred}) + \text{neg}_w \cdot (1\cdot\text{y_true} + 1\cdot\text{y_pred})) = (\text{pos}_w - \text{neg}_w) \cdot \text{y_sum} + 2 \cdot \text{neg}_w = (1-2\cdot\text{neg}_w) \cdot \text{y_sum} + 2 \cdot \text{neg}_w \)
4. dice score = \( \frac{\text{num}}{\text{denom}} \)
where num and denom are summed over all axes except the batch axis.

Init.

Parameters

- **binary** – if True, project y_true, y_pred to 0 or 1.
- **neg_weight** – weight for negative class.
- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
- **reduction** – using SUM reduction over batch axis, calling the loss like loss(y_true, y_pred) will return a scalar tensor.
- **name** – str, name of the loss.

get_config()

Return the config dictionary for recreating this class.

class deepreg.model.loss.label.JaccardIndex (*args: Any, **kwargs: Any)

Define Jaccard index.

The formulation is: 1. num = y_true * y_pred 2. denom = y_true + y_pred - y_true * y_pred 3. Jaccard index = num / denom

Init.

Parameters

- **binary** – if True, project y_true, y_pred to 0 or 1.
- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
- **reduction** – using SUM reduction over batch axis, calling the loss like loss(y_true, y_pred) will return a scalar tensor.
- **name** – str, name of the loss.

get_config()

Return the config dictionary for recreating this class.

class deepreg.model.loss.label.JaccardLoss (*args: Any, **kwargs: Any)

Revert the sign of JaccardIndex.

Init without required arguments.

Parameters **kwargs – additional arguments.

class deepreg.model.loss.label.MultiScaleLoss (*args: Any, **kwargs: Any)

Base class for multi-scale loss.

It applies the loss at different scales (gaussian or cauchy smoothing). It is assumed that loss values are between 0 and 1.

Init.

Parameters

- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
• **reduction** – using SUM reduction over batch axis, calling the loss like \( \text{loss}(y_{true}, y_{pred}) \) will return a scalar tensor.

• **name** – str, name of the loss.

**call** \((y_{true}: \text{tensorflow.Tensor}, y_{pred}: \text{tensorflow.Tensor}) \rightarrow \text{tensorflow.Tensor}\)

Use \_\_call\_\_ to calculate loss at different scales.

**Parameters**

• **y_true** – ground-truth tensor.

• **y_pred** – predicted tensor.

**Returns** multi-scale loss.

**get_config**()

Return the config dictionary for recreating this class.

deepreg.model.loss.label.cauchy_kernel1d\( (\text{sigma: int}) \rightarrow \text{tensorflow.Tensor}\)

Approximating cauchy kernel in 1d.

**Parameters** **sigma** – int, defining standard deviation of kernel.

**Returns** shape = (dim,) or ()

deepreg.model.loss.label.compute_centroid\( (y_{true}: \text{tensorflow.Tensor}, \text{grid}: \text{tensorflow.Tensor}) \rightarrow \text{tensorflow.Tensor}\)

Calculate the centroid of the mask.

**Parameters**

• **mask** – shape = (batch, dim1, dim2, dim3)

• **grid** – shape = (dim1, dim2, dim3, 3)

**Returns** shape = (batch, 3), batch of vectors denoting location of centroids.

deepreg.model.loss.label.compute_centroid_distance\( (y_{true}: \text{tensorflow.Tensor}, y_{pred}: \text{tensorflow.Tensor}, \text{grid}: \text{tensorflow.Tensor}) \rightarrow \text{tensorflow.Tensor}\)

Calculate the L2-distance between two tensors’ centroids.

**Parameters**

• **y_true** – tensor, shape = (batch, dim1, dim2, dim3)

• **y_pred** – tensor, shape = (batch, dim1, dim2, dim3)

• **grid** – tensor, shape = (dim1, dim2, dim3, 3)

**Returns** shape = (batch,)

deepreg.model.loss.label.foreground_proportion\( (y: \text{tensorflow.Tensor}) \rightarrow \text{tensorflow.Tensor}\)

Calculate the percentage of foreground vs background per 3d volume.

**Parameters** **y** – shape = (batch, dim1, dim2, dim3), a 3D label tensor

**Returns** shape = (batch,)

deepreg.model.loss.label.gaussian_kernel1d\( (\text{sigma: int}) \rightarrow \text{tensorflow.Tensor}\)

Calculate a gaussian kernel.

**Parameters** **sigma** – number defining standard deviation for gaussian kernel.

**Returns** shape = (dim,) or ()
Create a 3d separable filter.

Here `tf.nn.conv3d` accepts the `filters` argument of shape (filter_depth, filter_height, filter_width, in_channels, out_channels), where the first axis of `filters` is the depth not batch, and the input to `tf.nn.conv3d` is of shape (batch, in_depth, in_height, in_width, in_channels).

**Parameters**

- `tensor` – shape = (batch, dim1, dim2, dim3)
- `kernel` – shape = (dim4,)

**Returns** shape = (batch, dim1, dim2, dim3)

### 3.22.3 Deformation Loss

Provide regularization functions and classes for ddf.

**class** `deepreg.model.loss.deform.BendingEnergy(*args: Any, **kwargs: Any)`

Calculate the bending energy of ddf using central finite difference.

y_true and y_pred have to be at least 5d tensor, including batch axis.

**Init.**

**Parameters**

- `name` – name of the loss

**call** `(inputs: tensorflow.Tensor, **kwargs) → tensorflow.Tensor`

Return a scalar loss.

**Parameters**

- `inputs` – shape = (batch, m_dim1, m_dim2, m_dim3, 3)
- `kwargs` – additional arguments.

**Returns** shape = ()

**class** `deepreg.model.loss.deform.GradientNorm(*args: Any, **kwargs: Any)`

Calculate the L1/L2 norm of ddf using central finite difference.

y_true and y_pred have to be at least 5d tensor, including batch axis.

**Init.**

**Parameters**

- `l1` – bool true if calculate L1 norm, otherwise L2 norm
- `name` – name of the loss

**call** `(inputs: tensorflow.Tensor, **kwargs) → tensorflow.Tensor`

Return a scalar loss.

**Parameters**

- `inputs` – shape = (batch, m_dim1, m_dim2, m_dim3, 3)
- `kwargs` – additional arguments.

**Returns** shape = ()

**get_config**()

Return the config dictionary for recreating this class.
Calculate gradients on x-axis of a 3D tensor using central finite difference.

- **Parameters**
  - \( \mathbf{x} \) – shape = \((\text{batch}, \text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})\)
  - \( \mathbf{y} \) – function to call
- **Returns**
  - shape = \((\text{batch}, \text{m\_dim1-2}, \text{m\_dim2-2}, \text{m\_dim3-2})\)

Calculate gradients on y-axis of a 3D tensor using central finite difference.

- **Parameters**
  - \( \mathbf{y} \) – shape = \((\text{batch}, \text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})\)
- **Returns**
  - shape = \((\text{batch}, \text{m\_dim1-2}, \text{m\_dim2-2}, \text{m\_dim3-2})\)

Calculate gradients on z-axis of a 3D tensor using central finite difference.

- **Parameters**
  - \( \mathbf{z} \) – shape = \((\text{batch}, \text{m\_dim1}, \text{m\_dim2}, \text{m\_dim3})\)
- **Returns**
  - shape = \((\text{batch}, \text{m\_dim1-2}, \text{m\_dim2-2}, \text{m\_dim3-2})\)

### 3.23 Optimizer

Functions parsing the config optimizer options

- **Parameters**
  - \( \text{optimizer\_config} \) – unpacked dictionary for the optimiser returned from yaml.load, optimiser options and parameters
- **Returns**
  - \( \text{tf.keras.optimizers} \) object
3.24 Guidelines

We welcome contributions to DeepReg. Please raise an issue to report bugs, request features, or ask questions. For code contribution, please follow the guidelines below.

3.24.1 Setup

We recommend using conda environment on Linux or Mac machines for code development. The setup steps are:

1. Install git.
2. Clone or fork the repository.
3. Install and activate `deepreg` conda environment.
4. Run `pre-commit install` under the root of this repository DeepReg/ to install pre-commit hooks.

3.24.2 Resolve an issue

For resolving an issue, please

1. Create a branch.
   
The branch name should start with the issue number, followed by hyphen separated words describing the issue, e.g. `1-update-contribution-guidelines`.

2. Implement the required features or fix the reported bugs.
   
   There are several guidelines for commit, coding, testing, and documentation.
   
   1. Please create commits with meaningful commit messages.

   The commit message should start with `Issue #<issue number>:`, for instance `Issue #1: add commit requirements`.

   2. Please write or update unit-tests for the added or changed functionalities.

   Pull request will not be approved if test coverage is decreased. Check testing guidelines for further details.

   3. Please write meaningful docstring and documentations for added or changed code.

   We use Sphinx docstring format.

   4. Please update the CHANGELOG regarding the changes.

3. Create a pull request when the branch is ready.

Please resume the changes with some more details in the description and check the boxes after submitting the pull request. Optionally, you can create a pull request and add `WIP` in the name to mark as working in progress.
3.24.3 Add a DeepReg demo

Adding DeepReg demo should be done via pull request. Besides the guidelines above, adding demo has additional requirements described in demo guidelines.

3.25 Unit Test

In DeepReg, we use *pytest* (not *unittest*) for unit tests to ensure a certain code quality and to facilitate the code maintenance.

The testing is checked via GitHub workflows and *Codecov* is used to monitor the test coverage. While checking the Codecov report in file mode, generally a line highlighted by red means it is not covered by test. Please check the Codecov documentation for more details.

3.25.1 Test requirement

We would like to achieve 100% test coverage. In general, tests should be

- thorough, covering different scenarios.
- independent, different scenarios are not tested together.
- clean and compact, for instance,
  - Use parameterized test to reduce code redundancy.
  - Use *is_equal_np* and *is_equal_tf* provided in *test/unit/util.py* to compare arrays or tensors.

The detailed requirements are as follows:

- Test all functions in python classes.
- Test the trigger of all warning and errors in functions.
- Test the correctness of output values for functions.
- Test at least the correctness of output shapes for TensorFlow functions.

3.25.2 Example unit test

We provide here an example to help understanding the requirements.

```python
import pytest
import logging

def subtract(x: int) -> int:
    """
    A function subtracts one from a non-negative integer.
    :param x: a non-negative integer
    :return: x - 1
    """
    assert isinstance(x, int), f"input {x} is not int"
    assert x >= 0, f"input {x} is negative"
    if x == 0:
        logging.warning("input is zero")
    return x - 1
```

(continues on next page)
class TestSubtract:
    @pytest.mark.parametrize("x,expected", [{0, -1}, (1, 0)]
    def test_value(self, x, expected):
        got = subtract(x=x)
        assert got == expected

    @pytest.mark.parametrize("x,msg", [{-1, "is negative"}, (0.0, "is not int")])
    def test_err(self, x, msg):
        with pytest.raises(AssertionError) as err_info:
            subtract(x=x)
        assert msg in str(err_info.value)

    def test_warning(self, caplog):
        caplog.clear()  # clear previous log
        subtract(x=0)
        assert "input is zero" in caplog.text

where

- we group multiple test functions for subtract under the same class TestSubtract.
- we parameterize test to test different inputs.
- we catch errors using pytest.raises and check error messages.
- we check warning message using caplog.

For further usage like fixture and other functionalities, please check pytest documentation or existing tests in DeepReg. You can also raise an issue for any questions.

### 3.26 DeepReg demo

The demos folder directly under the DeepReg root directory contains demonstrations using DeepReg for different image registration applications.

Contributions are welcome! Below is a set of requirements for a demo to be included as a DeepReg Demo.

- Each demo must have an independent folder directly under demos/;
- Name the folder as [loader-type]_[image-modality]_[organ-disease]_[optional:brief-remark], e.g. unpaired_ultrasound_prostate or grouped_mr_brain_longitudinal;
- For simplicity, avoid sub-folders (other than those specified below) and separate files for additional functions/classes;
- Experiment using cross-validation or advanced data set sampling is NOT encouraged, unless the purpose of the demo is to demonstrate how to design experiments.
3.26.1 Open accessible data

• Each demo must have a demo_data.py script to automatically download and preprocess demo data;

• Data for training and test should be downloaded under the demo folder named dataset, such as dataset/train and dataset/test;

• Data should be hosted in a reliable and efficient (DeepReg repo will not store demo data or model) online storage, Kaggle, GitHub and Zendoo are all options for non-login access (avoid google drive for known accessibility issues);

• Relevant dataset folder structure to utilise the supported loaders can be either pre-arranged in data source or scripted in demo_data.py after downloading;

• Avoid slow and excessively large data set download. Consider downloading a subset as default for demonstration purpose, with options for full data set.

3.26.2 Pre-trained model

• A pre-trained model must be available for downloading, with github.com/DeepRegNet/deepreg-model-zoo being preferred for storing the models. Please contact the Development Team for access;

• The pre-trained model, e.g. ckpt files, should be downloaded and extracted under the dataset/pretrained folder. Avoid overwriting with user-trained models;

3.26.3 Training

• Each demo must have a demo_train.py script;

• This is accompanied by one or more config yaml files in the same folder. Please use the same demo folder name for the config file. Add postfix if multiple training methods are provided, e.g. unpaired_ct_abdomen_comb.yaml, unpaired_ct_abdomen_unsup.yaml.

3.26.4 Predicting

• Each demo must have a demo_predict.py script;

• By default, the pre-trained model should be used in demo_predict.py. However, the instruction should be clearly given to use the user-trained model, saved with the demo_train.py;

3.26.5 A README.md file

A markdown file must be provided under demos/<demo_name> as an entry point for each demo, which should be based on the template. Moreover, a .rst file must be provided under docs/source/demo to link the markdown file to the documentation page. The introduction.rst file should be updated properly as well.

Following is a checklist for modifying the README template:

• Update the link to source code;

• Update the author section;

• Update the application section;

• Update the data section, optionally, describe the used pre-processing methods;

• Update the name in all commands;
• Update the reference section.
• Optionally, adapt the file to custom needs.

3.27 Documentation

We use Sphinx to organize the documentation and it is hosted in ReadTheDocs.

3.27.1 Local Build

Please run the following command under docs/ directory for generating the documentation pages locally. Generated files are under docs/build/html/

```
make clean html
```

where

• clean removes the possible built files.
• Optionally we can add SPHINXOPTS="-W" to fail on any warnings, but we are currently having document isn't included in any toctree warning and no better solution has been found yet.

3.27.2 Recommendations

There are some recommendations regarding the docs.

• We prefer markdown files over reStructuredText files as its linting is covered using Prettier.

Only use reStructuredText (rst) files for some functionalities not supported by markdown, such as

– toctree
– warning/notes boxes in Installation

The conversion between markdown and rst can be done automatically using free online tool Pandoc.

• When linking to other pages, please use relative paths such as ..//getting_started/install.html instead of absolute paths https://deepreg.readthedsocs.io/en/latest/getting_started/install.html as relative paths are more robust for different version of documentations.

• To refer a markdown file outside of the source folder, create an rst file and use .. mdinclude:: <makrdown file path> to include the markdown source.

Check the source code of paired lung CT image registration page as an example.

3.28 Release

DeepReg is distributed on PyPI. To create new releases, you can follow the below instructions to automatically submit new versions to PyPI via our GitHub Actions workflow.
3.28.1 Creating a new Release

From the main DeepReg repository page, head to releases. From here, you can draft a new release.

3.28.2 Tagging & Titling

We follow semver naming conventions for tags, with vX.Y.Z where each represents major, minor, and patch release versions.

From semver.org:

- Major version: when you make incompatible API changes,
- Minor version: when you add functionality in a backwards compatible manner, and
- Patch version: when you make backwards compatible bug fixes.

Typically, most releases will be an increment of the minor or patch versions.

Enter the new version in the format vX.Y.Z into the “Tag version” and “Release title” fields of the draft a release page.

3.28.3 Publish!

Click the “Publish release” button, and our GitHub Action workflow will handle the rest.

It’s recommended that you check the output log from the GitHub Actions page to make sure everything went as planned for the release.
d

deepreg.dataset.loader.grouped_loader, 82
deepreg.dataset.loader.h5_loader, 85
deepreg.dataset.loader.nifti_loader, 84
deepreg.dataset.loader.paired_loader, 80

deepreg.dataset.loader.unpaired_loader, 81
deepreg.model.layer_util, 96
deepreg.model.loss.deform, 107
deepreg.model.loss.image, 101
deepreg.model.loss.label, 104
deepreg.model.optimizer, 108
deepreg.predict, 78
deepreg.train, 77
deepreg.warp, 80
INDEX

A

Activation (class in deepreg.model.layer), 89
AdditiveUpSampling (class in deepreg.model.layer), 89

B

BendingEnergy (class in deepreg.model.loss.deform), 107
build() (deepreg.model.layer.LocalNetUpSampleResnetBlock method), 93
build() (deepreg.model.layer.UpSampleResnetBlock method), 95
build_config() (in module deepreg.train), 77
build_pair_output_path() (in module deepreg.predict), 78
build_rectangular_kernel() (in module deepreg.model.loss.image), 103
build_triangular_kernel() (in module deepreg.model.loss.image), 103

C

call() (deepreg.model.backbone.global_net.GlobalNet method), 87
call() (deepreg.model.backbone.local_net.LocalNet method), 87
call() (deepreg.model.backbone.u_net.UNet method), 88
call() (deepreg.model.layer.AdditiveUpSampling method), 89
call() (deepreg.model.layer.Conv3dBlock method), 90
call() (deepreg.model.layer.Conv3dWithResize method), 90
call() (deepreg.model.layer.Deconv3dBlock method), 91
call() (deepreg.model.layer.Dense method), 91
call() (deepreg.model.layer.DownSampleResnetBlock method), 92
call() (deepreg.model.layer.IntDVF method), 93
call() (deepreg.model.layer.LocalNetUpSampleResnetBlock method), 93
call() (deepreg.model.layer.Residual3dBlock method), 94
call() (deepreg.model.layer.UpSampleResnetBlock method), 95
call() (deepreg.model.layer.Warping method), 95
call() (deepreg.model.loss.deform.BendingEnergy method), 107
call() (deepreg.model.loss.deform.GradientNorm method), 107
call() (deepreg.model.loss.image.GlobalMutualInformation method), 102
call() (deepreg.model.loss.image.LocalNormalizedCrossCorrelation method), 102
call() (deepreg.model.loss.image.SumSquaredDifference method), 103
call() (deepreg.model.loss.label.MultiScaleLoss method), 106
cauhky_kernel1d() (in module deepreg.model.loss.label), 106
close() (deepreg.dataset.loader.grouped_loader.GroupedDataLoader method), 83
close() (deepreg.dataset.loader.h5_loader.H5FileLoader method), 85
close() (deepreg.dataset.loader.interface.FileLoader method), 83
close() (deepreg.dataset.loader.nifti_loader.NiftiFileLoader method), 84
close() (deepreg.dataset.loader.unpaired_loader.UnpairedDataLoader method), 81
compute_centroid() (in module deepreg.model.loss.label), 106
compute_centroid_distance() (in module deepreg.model.loss.label), 106
Conv3d (class in deepreg.model.layer), 89
Conv3dBlock (class in deepreg.model.layer), 90
Conv3dWithResize (class in deepreg.model.layer), 90
CrossEntropy (class in deepreg.model.loss.label), 104
set_data_structure()  (deep-reg.dataset.loader.h5_loader.H5FileLoader method), 86
set_data_structure()  (deep-reg.dataset.loader.interface.FileLoader method), 84
set_data_structure()  (deep-reg.dataset.loader.nifti_loader.NiftiFileLoader method), 85
set_group_structure()  (deep-reg.dataset.loader.h5_loader.H5FileLoader method), 86
set_group_structure()  (deep-reg.dataset.loader.interface.FileLoader method), 84
set_group_structure()  (deep-reg.dataset.loader.nifti_loader.NiftiFileLoader method), 85
shape_sanity_check() (in module deepreg.warp), 80
SumSquaredDifference (class in deep-reg.model.loss.image), 103

T

train() (in module deepreg.train), 78

U

UNet (class in deepreg.model.backbone.u_net), 88
UnpairedDataLoader (class in deep-reg.dataset.loader.unpaired_loader), 81
UpSampleResnetBlock (class in deep-reg.model.layer), 95

V

validate_data_files() (deep-reg.dataset.loader.grouped_loader.GroupedDataLoader method), 83
validate_data_files() (deep-reg.dataset.loader.paired_loader.PairedDataLoader method), 81
validate_data_files() (deep-reg.dataset.loader.unpaired_loader.UnpairedDataLoader method), 81

W

warp() (in module deepreg.warp), 80
warp_grid() (in module deepreg.model.layer_util), 101
warp_image_ddf() (in module deepreg.model.layer_util), 101
Warping (class in deepreg.model.layer), 95