DeepReg

DeepReg

Jul 21, 2021

GETTING STARTED

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!

CHAPTER

ONE

FEATURES

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.

CHAPTER

TWO

CONTACT

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).

CHAPTER

THREE

CONTRIBUTORS

DeepReg is maintained by a team of developers and researchers. People with significant contributions to DeepReg are acknowledged in the Contributor List.

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).

3.1 Installation

DeepReg can be installed in Python 3.7 and external python dependencies are mainly defined in requirements. DeepReg primarily supports and is regularly tested with Ubuntu and Mac OS.

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:

```
git clone https://github.com/DeepRegNet/DeepReg.git
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

Windows

Install prerequisites (Optional).

```
sudo apt-get update
sudo apt-get install graphviz
```

Install DeepReg without GPU support.

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

Install DeepReg with GPU support.

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

Install prerequisites (Optional).

```
brew install graphviz
```

Install DeepReg without GPU support.

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

Install DeepReg with GPU support.

> **Warning:** Not supported or tested.

Install 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 DeepReg with GPU support.

> **Warning:** Not supported or tested.

After activating the conda environment, please install DeepReg locally:

```
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:

```
git clone https://github.com/DeepRegNet/DeepReg.git
```

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` 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:

```
deepreg_train --gpu "" --config_path config/unpaired_labeled_ddf.yaml --exp_name 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:

```
deepreg_predict --gpu "" --ckpt_path logs/test/save/ckpt-2 --split test
```

where:

- `--ckpt_path <filepath>` specifies the checkpoint file path.
- `--split 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":

```yaml
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":

```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":

```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":

```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 the 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

Below is the script to install the environment for DeepReg in the cluster. If you want to switch to the current working branch. Call `git branch origin/<branch_name>` after downloading DeepReg.

```
git clone https://github.com/<personal_acount_id>/DeepReg.git
module load default/python/3.8.5

cd <DeepReg_Dir>

export PATH=/share/apps/anaconda3-5/bin:$PATH
conda env create -f environment.yml   #set up environment
source /share/apps/source_files/cuda/cuda-10.1.source    # set up cuda for GPU
source activate deepreg   # activate conda env

export CONDA_PIP="/home/<cs_account_id>/.conda/envs/deepreg/bin/pip"
$CONDA_PIP install -e .
```

`module` is a command to find packages and set up them in your own storge. You can get more information by `module help`. Another way to install packages is

```
export PATH=/share/apps/<module_name>/bin:$PATH
```

You can find any available packages in cluster nodes by

```
ls /share/apps/ | grep '<package_name>*'
```

Tip: For now, all the packages are stored in `share/apps/`. If the path does not exist, try `module load default/python/3.8.5`. Then, call `$PATH` to find the new location of packages.

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 home/<cs_account_id>/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
#$ -N DeepReg_tst   # job name
#$ -wd home/<cs_account_id>/logs   # output, error log dir.
#Please call `mkdir logs` before using the script.

hostname
date

cd ../<DeepReg_dir>
export PATH=/share/apps/anaconda3-5/bin:$PATH
source activate deepreg   # activate conda env
export PATH=/share/apps/cuda-10.1/bin:/share/apps/gcc-8.3/bin:$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

deepreg_train \
--gpu \
--config_path config/unpaired_labeled_ddf.yaml \
--log_dir test
```

You also can directly access one of four cluster nodes reserved for development purposes, by the command below. You can then run your code via the command line. More information on the specific configuration at UCL is available here.

```
qrsh -l tmem=14G,h_vmem=14G
```

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.

Source Code

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 `--full` to use the original training configuration, such as

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

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
```

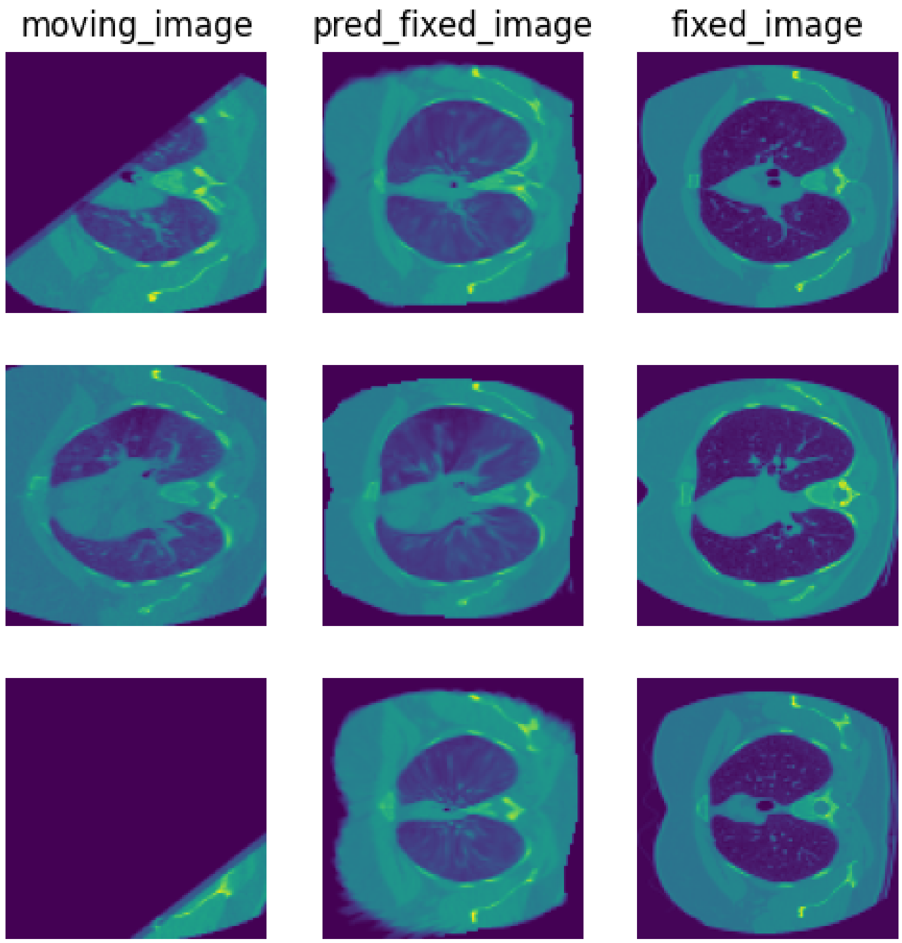
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/paired_ct_lung/logs_predict/<time-stamp>/test/<pair-number>
↪/moving_image.nii.gz, demos/paired_ct_lung/logs_predict/<time-stamp>/test/<pair-
↪number>/pred_fixed_image.nii.gz, demos/paired_ct_lung/logs_predict/<time-stamp>/
↪test/<pair-number>/fixed_image.nii.gz' --slice-inds '64,50,72' -s demos/paired_ct_
↪lung/logs_predict/
```

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

[1] Hering, Alessa, Murphy,Keelin, and van Ginneken, Bram. (2020). Lean2Reg Challenge: CT Lung Registration : CT Lung Registration - Training Data

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_mrus_brain/logs_train.

```
python demos/paired_mrus_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 --full to use the original training configuration, such as

```
python demos/paired_mrus_brain/demo_train.py --full
```

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_mrus_brain/logs_predict. Check the CLI documentation for more details about prediction output.

```
python demos/paired_mrus_brain/demo_predict.py
```

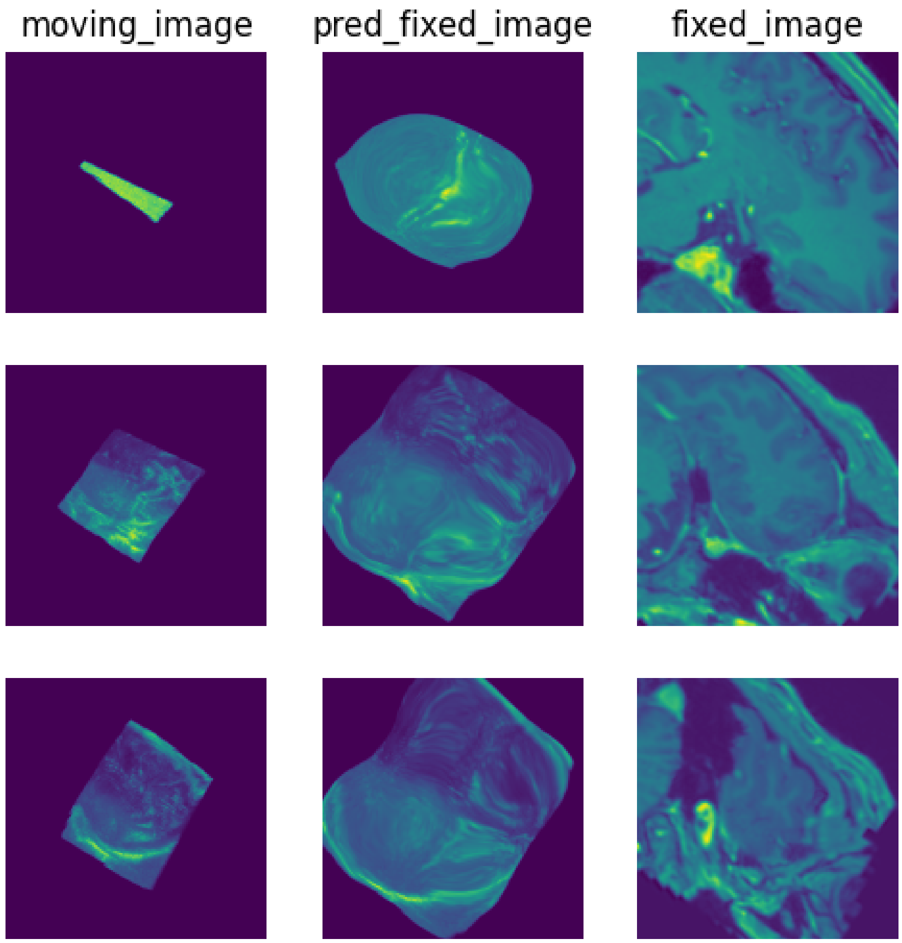
Optionally, the user-trained model can be used by changing the ckpt_path variable inside demo_predict.py. Note that the path should end with .ckpt and checkpoints are saved under logs_train as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/paired_mrus_brain/logs_predict/<time-stamp>/test/<pair-
↪number>/moving_image.nii.gz, demos/paired_mrus_brain/logs_predict/<time-stamp>/test/
↪<pair-number>/pred_fixed_image.nii.gz, demos/paired_mrus_brain/logs_predict/<time-
↪stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '190,128,96' -s demos/
↪paired_mrus_brain/logs_predict
```

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

[1] Y. Xiao, M. Fortin, G. Unsgård , H. Rivaz, and I. Reinertsen, "REtroSpective Evaluation of Cerebral Tumors (RESECT): a clinical database of pre-operative MRI and intra-operative ultrasound in low-grade glioma surgeries". Medical Physics, Vol. 44(7), pp. 3875-3882, 2017.

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 segmentaion.

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/`.

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 `--full` to use the original training configuration, such as

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

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
```

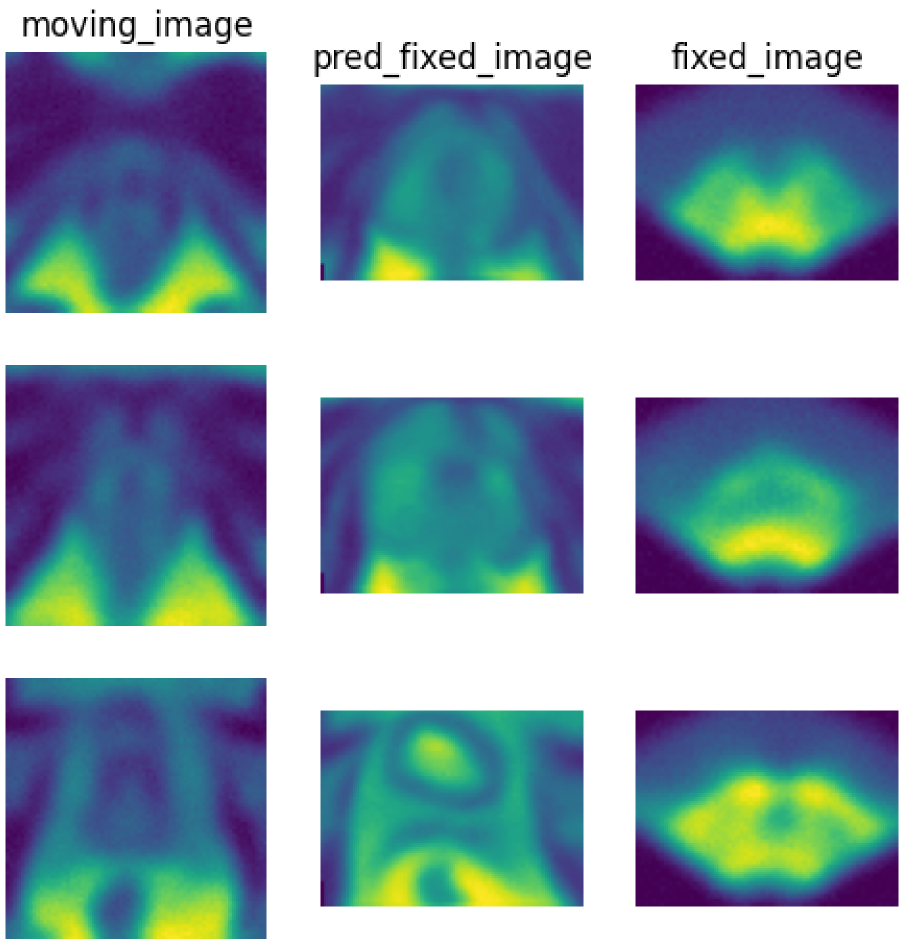
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/paired_mrus_prostate/logs_predict/<time-stamp>/test/<pair-
↪number>/moving_image.nii.gz, demos/paired_mrus_prostate/logs_predict/<time-stamp>/
↪test/<pair-number>/pred_fixed_image.nii.gz, demos/paired_mrus_prostate/logs_predict/
↪<time-stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '12,20,36' -s
↪demos/paired_mrus_prostate/logs_predict
```

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 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.

- 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 `--full` to use the original training configuration, such as

```
python demos/unpaired_ct_abdomen/demo_train.py --method unsup --full
```

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
```

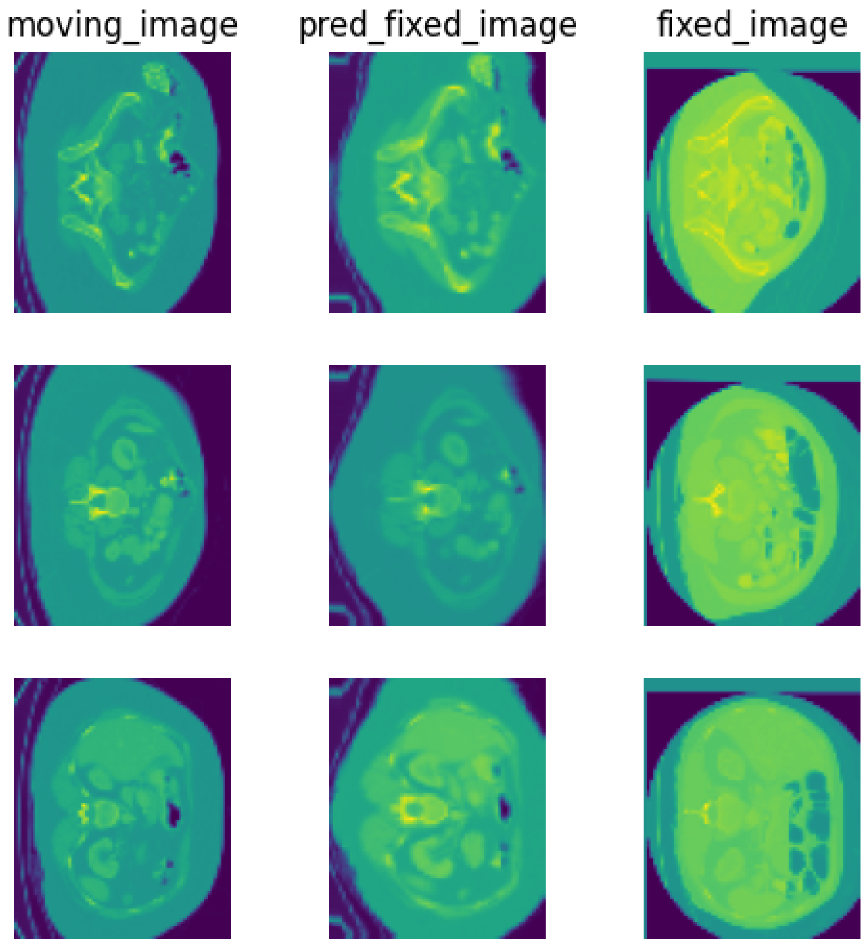
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/unpaired_ct_abdomen/logs_predict/comb/<time-stamp>/test/
↪<pair-number>/moving_image.nii.gz, demos/unpaired_ct_abdomen/logs_predict/comb/
↪<time-stamp>/test/<pair-number>/pred_fixed_image.nii.gz, demos/unpaired_ct_abdomen/
↪logs_predict/comb/<time-stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds
↪'30,50,65' -s demos/unpaired_ct_abdomen/logs_predict
```

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

[1] Adrian Dalca, Yipeng Hu, Tom Vercauteren, Mattias Heinrich, Lasse Hansen, Marc Modat, Bob de Vos, Yiming Xiao, Hassan Rivaz, Matthieu Chabanas, Ingerid Reinertsen, Bennett Landman, Jorge Cardoso, Bram van Ginneken, Alessa Hering, and Keelin Murphy. (2020, March 19). Learn2Reg - The Challenge. Zenodo. http://doi.org/10.5281/zenodo.3715652

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 timepoint 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#.XsUWXsBpFhE

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/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`.

```
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 `--full` to use the original training configuration, such as

```
python demos/unpaired_ct_lung/demo_train.py --full
```

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
```

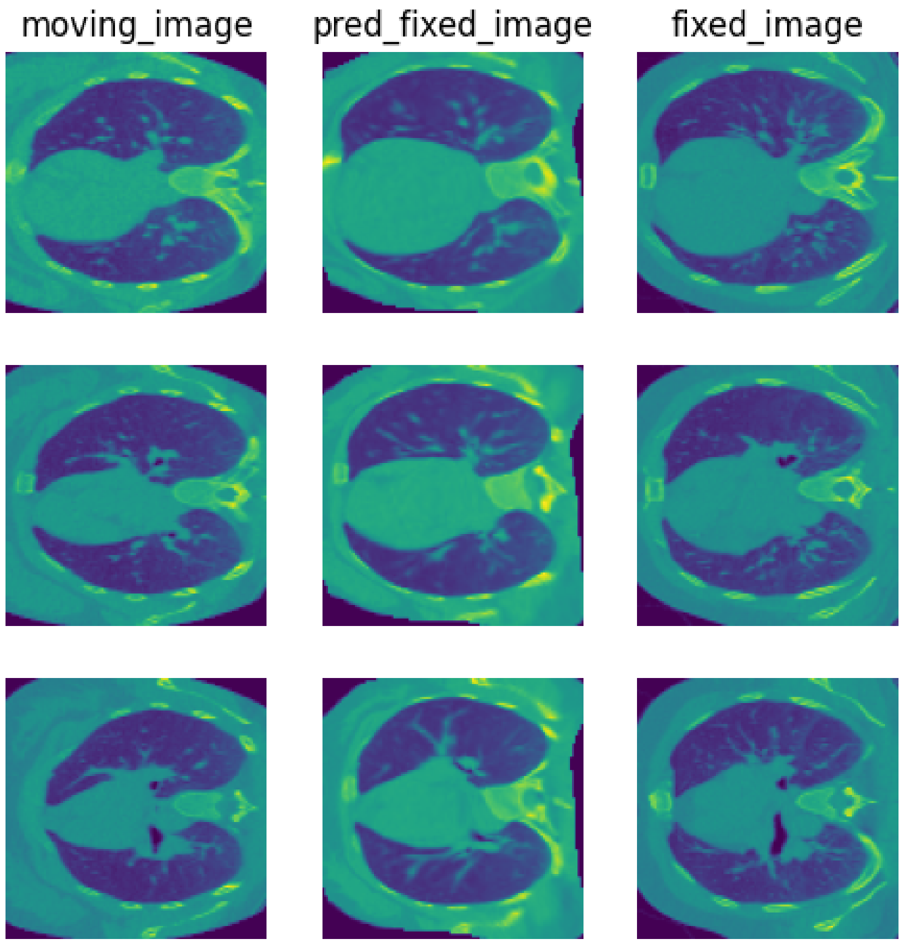
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/unpaired_ct_lung/logs_predict/<time-stamp>/test/<pair-
→number>/moving_image.nii.gz, demos/unpaired_ct_lung/logs_predict/<time-stamp>/test/
→<pair-number>/pred_fixed_image.nii.gz, demos/unpaired_ct_lung/logs_predict/<time-
→stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '40,48,56' -s demos/
→unpaired_ct_lung/logs_predict
```

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

[1] Hering A, Murphy K, and van Ginneken B. (2020). Lean2Reg Challenge: CT Lung Registration - Training Data [Data set]. Zenodo. http://doi.org/10.5281/zenodo.3835682

[2] Li B, Christensen GE, Hoffman EA, McLennan G, Reinhardt JM. Establishing a normative atlas of the human lung: intersubject warping and registration of volumetric CT images. Acad Radiol. 2003;10(3):255-265. doi:10.1016/s1076-6332(03)80099-5

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=1RvJIjG2loU8uGkWzUuGjqVcGQW2RzNYA

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 `--full` to use the original training configuration, such as

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

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.

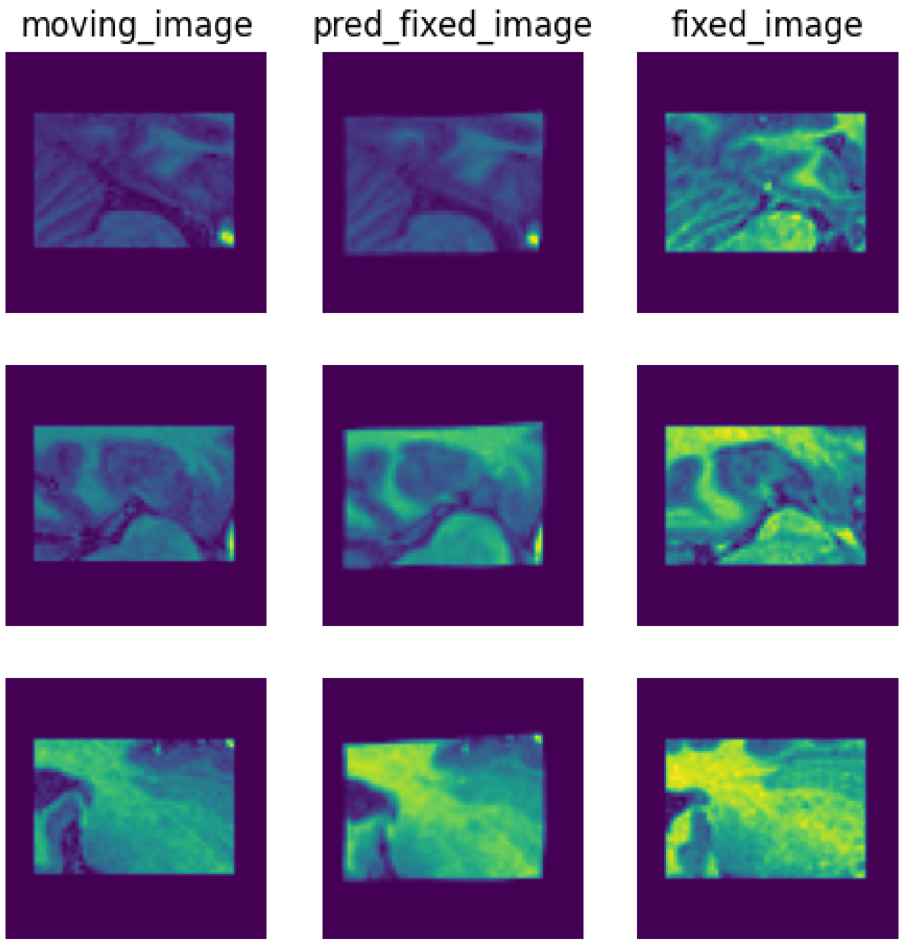
```
python demos/unpaired_mr_brain/demo_predict.py
```

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.

```
deepreg_vis -m 2 -i 'demos/unpaired_mr_brain/logs_predict/<time-stamp>/test/<pair-
↪number>/moving_image.nii.gz, demos/unpaired_mr_brain/logs_predict/<time-stamp>/test/
↪<pair-number>/pred_fixed_image.nii.gz, demos/unpaired_mr_brain/logs_predict/<time-
↪stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '20,32,44' -s demos/
↪unpaired_mr_brain/logs_predict/
```

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

[1] AL Simpson et al., *A large annotated medical image dataset for the development and evaluation of segmentation algorithms* (2019). https://arxiv.org/abs/1902.09063

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 `--full` to use the original training configuration, such as

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

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
```

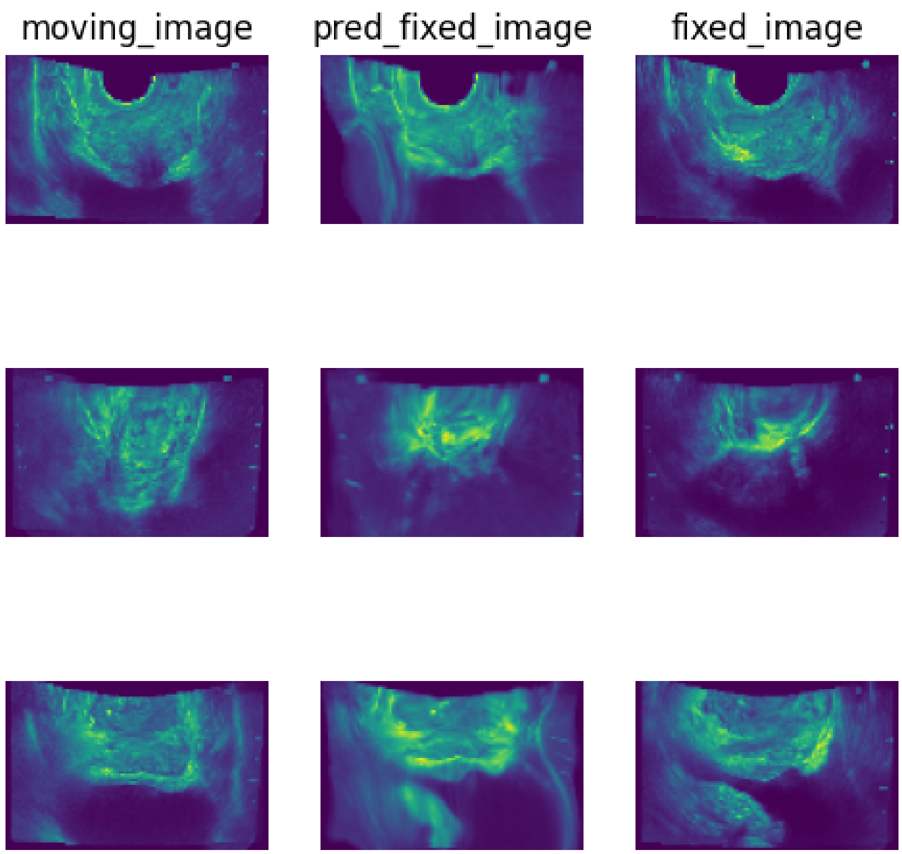
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

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.

```
deepreg_vis -m 2 -i 'demos/unpaired_us_prostate_cv/logs_predict/<time-stamp>/test/
↪<pair-number>/moving_image.nii.gz, demos/unpaired_us_prostate_cv/logs_predict/<time-
↪stamp>/test/<pair-number>/pred_fixed_image.nii.gz, demos/unpaired_us_prostate_cv/
↪logs_predict/<time-stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '50,
↪65,35' -s demos/unpaired_us_prostate_cv/logs_predict/
```

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.

3.9 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

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.

This demo also implements the feature-based registration described in Morphological Change Forecasting for Prostate Glands using Feature-based Registration and Kernel Density Extrapolation.

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 `--full` to use the original training configuration, such as

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

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
```

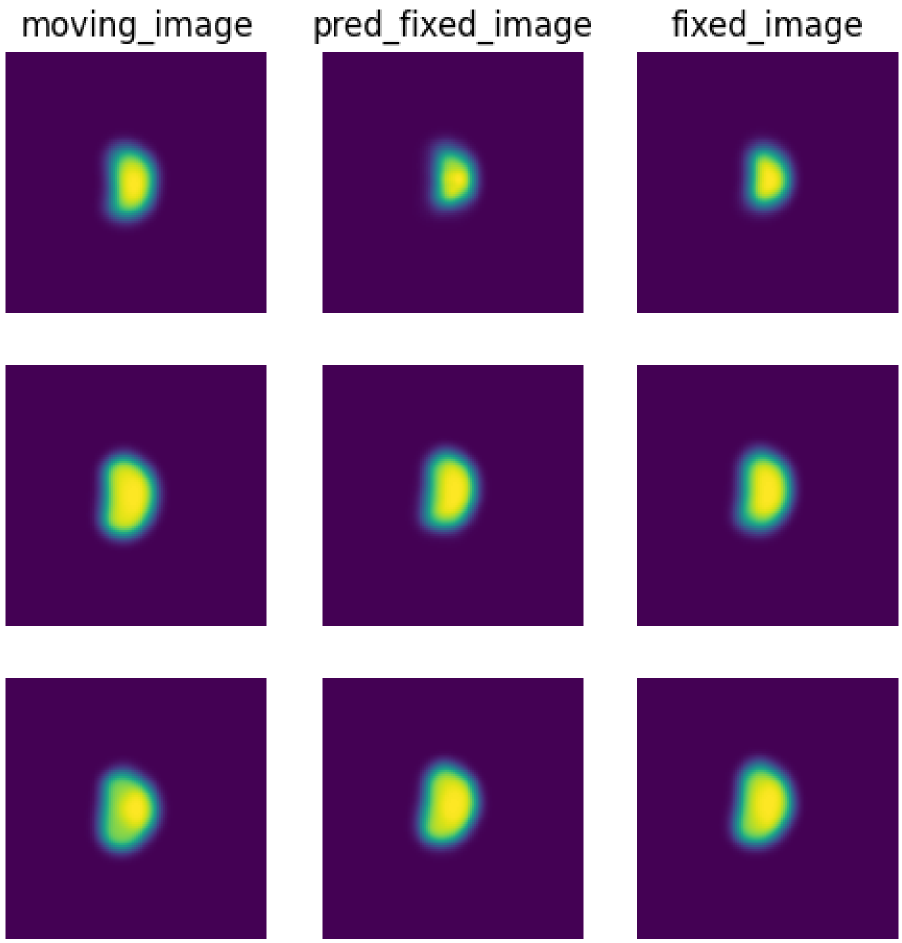
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the 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.

```
deepreg_vis -m 2 -i 'demos/grouped_mask_prostate_longitudinal/logs_predict/<time-
↪stamp>/test/<pair-number>/moving_image.nii.gz, demos/grouped_mask_prostate_
↪longitudinal/logs_predict/<time-stamp>/test/<pair-number>/pred_fixed_image.nii.gz,␣
↪demos/grouped_mask_prostate_longitudinal/logs_predict/<time-stamp>/test/<pair-
↪number>/fixed_image.nii.gz' --slice-inds '10,16,20' -s demos/grouped_mask_prostate_
↪longitudinal/logs_predict
```

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.

3.9.2 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 `--full` to use the original training configuration, such as

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

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
```

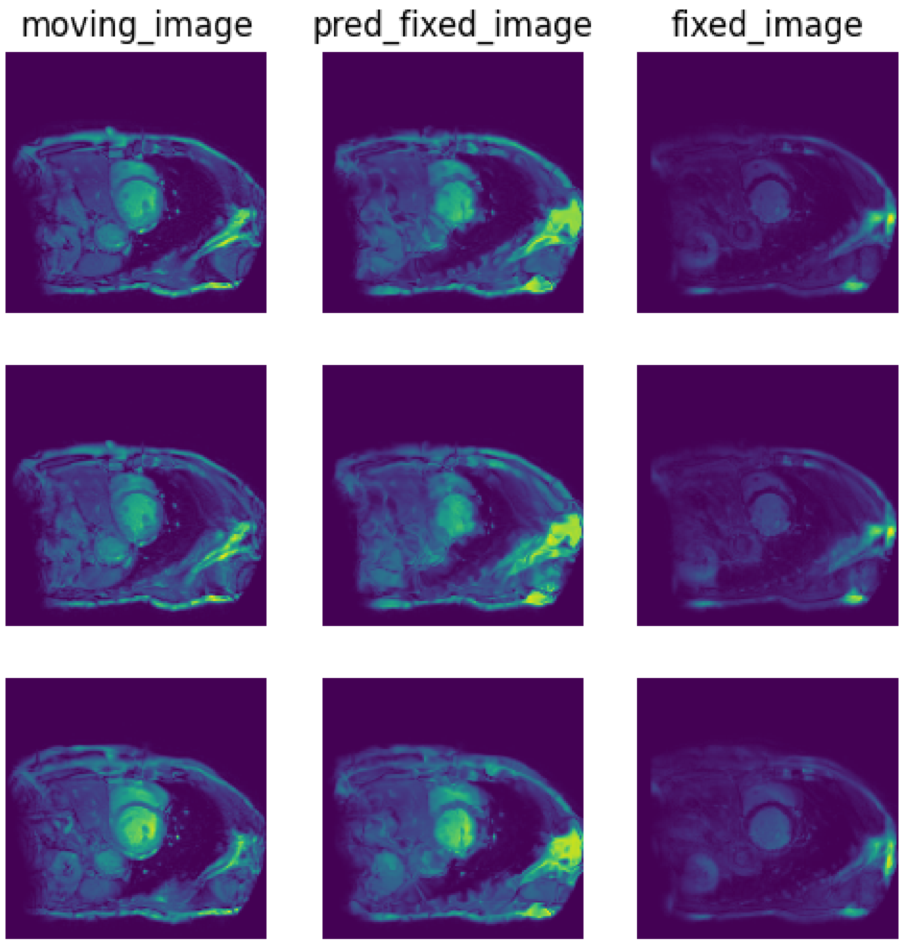
Optionally, the user-trained model can be used by changing the `ckpt_path` variable inside `demo_predict.py`. Note that the path should end with `.ckpt` and checkpoints are saved under `logs_train` as mentioned above.

Visualise

The following command can be executed to generate a plot of three image slices from the 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.

```
deepreg_vis -m 2 -i 'demos/grouped_mr_heart/logs_predict/<time-stamp>/test/<pair-
↪number>/moving_image.nii.gz, demos/grouped_mr_heart/logs_predict/<time-stamp>/test/
↪<pair-number>/pred_fixed_image.nii.gz, demos/grouped_mr_heart/logs_predict/<time-
↪stamp>/test/<pair-number>/fixed_image.nii.gz' --slice-inds '14,10,20' -s demos/
↪grouped_mr_heart/logs_predict
```

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

[1] Xiahai Zhuang: Multivariate mixture model for myocardial segmentation combining multi-source images. IEEE Transactions on Pattern Analysis and Machine Intelligence (T PAMI), vol. 41, no. 12, 2933-2946, Dec 2019. link.

[2] Xiahai Zhuang: Multivariate mixture model for cardiac segmentation from multi-sequence MRI. International Conference on Medical Image Computing and Computer-Assisted Intervention, pp.581-588, 2016.

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 anf 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 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.

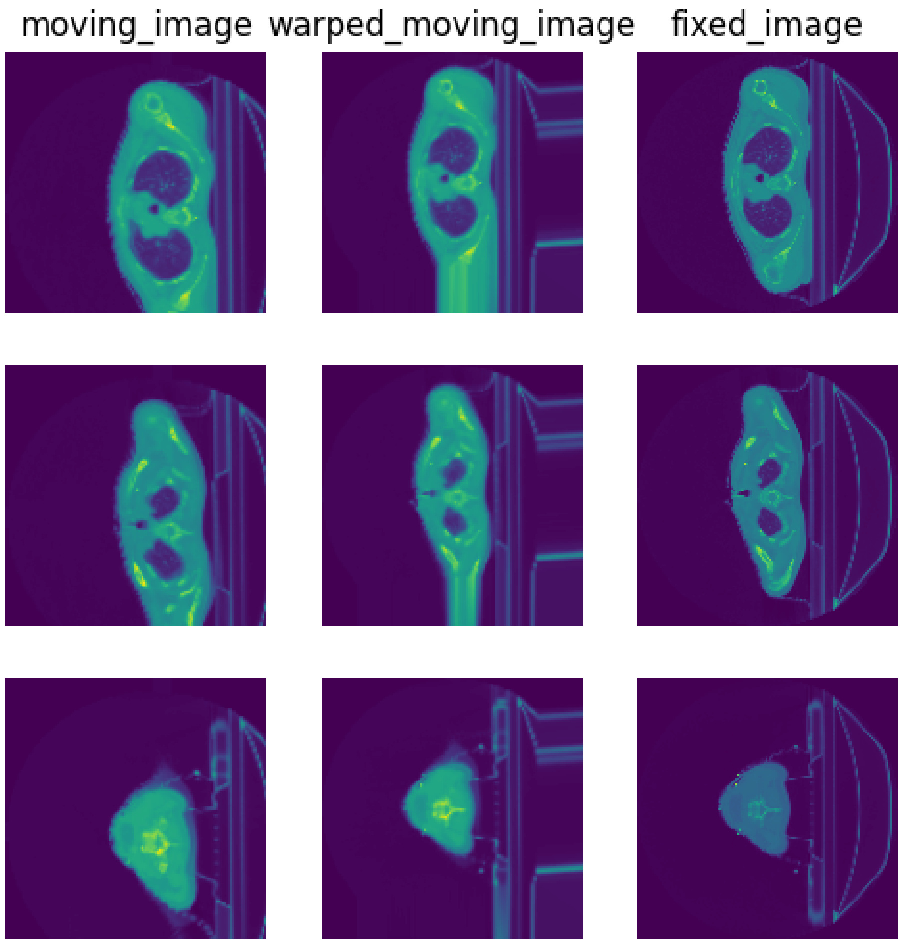
```
python demos/classical_ct_headneck_affine/demo_register.py
```

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.

```
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

[1] Vallières, M. et al. Radiomics strategies for risk assessment of tumour failure in head-and-neck cancer. Sci Rep 7, 10117 (2017). doi: 10.1038/s41598-017-10371-5

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.

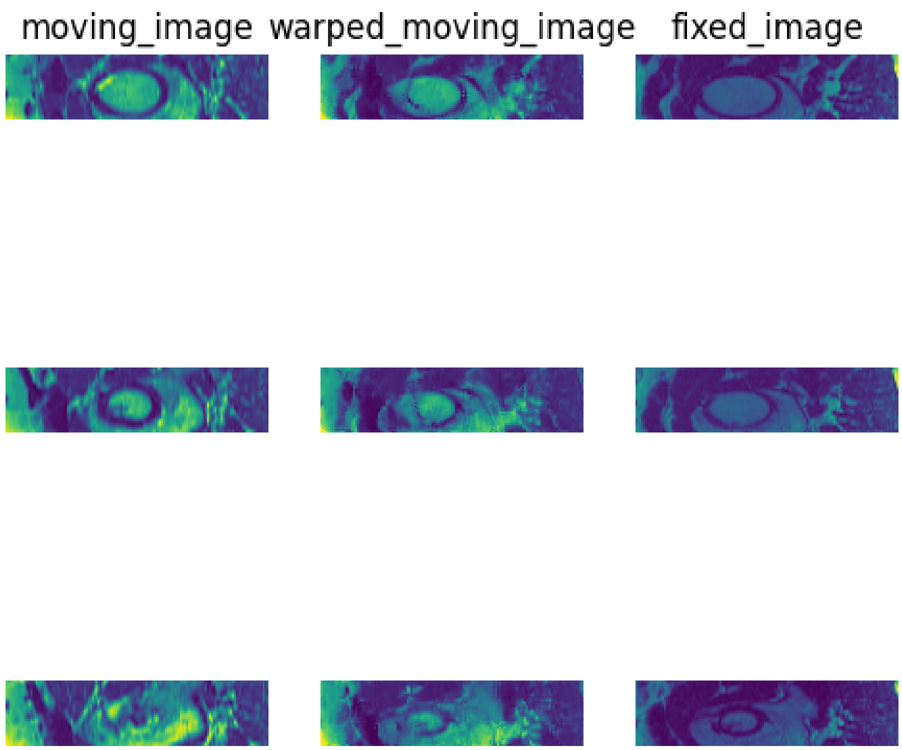
```
python demos/classical_mr_prostate_nonrigid/demo_register.py
```

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.

```
deepreg_vis -m 2 -i 'demos/classical_mr_prostate_nonrigid/logs_reg/moving_image.nii.
↪gz, demos/classical_mr_prostate_nonrigid/logs_reg/warped_moving_image.nii.gz, demos/
↪classical_mr_prostate_nonrigid/logs_reg/fixed_image.nii.gz' --slice-inds '4,8,12' -
↪s demos/classical_mr_prostate_nonrigid/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

[1] Litjens, G., Toth, R., van de Ven, W., Hoeks, C., Kerkstra, S., van Ginneken, B., Vincent, G., Guillard, G., Birbeck, N., Zhang, J. and Strand, R., 2014. Evaluation of prostate segmentation algorithms for MRI: the PROMISE12 challenge. Medical image analysis, 18(2), pp.359-373.

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

- **CPU allocation**:

  `--num_workers`, if given, TensorFlow will use limited CPUs.

  By default, it uses only 1 CPUs. Setting it to non-positive values will be using all CPUs.

  Example usage:

  - `--num_workers 2` for using at most 2 CPUs.

- **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.

- **Log directory**:

  `--log_dir`, specifies the log directory for logging output information and results.

  By default, it is `logs` under the package root.

  Example usage:

  - `--log_dir logs` for specifying the log directory `logs/` under current directory.

- **Experiment name**:

  `--exp_name` or `-n`, specifies the name of an experiment (every time a training or a prediction is run), which will be used together with the log directory (via `log_dir`) to specify the sub-folder that saves the output information and results from individual experiments (runs).

  If this is not provided, it creates a timestamp-named sub-folder under the `log_dir`, by default, e.g. `logs/20200810-194042/`.

  Example usage:

  - `--exp_name test --log_dir logs` for saving under `logs/test/`.
  - `--log_dir logs` for saving under `logs/20210101-120000/`, assuming `20210101-120000` is current time.
  - `--exp_name test` for saving under `DeepReg/logs/test/`, assuming `DeepReg` is the package root.

- **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

`deepreg_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**:

  `--split`, specifies in which data set the prediction is performed.

  It must be one of `train` / `valid` / `test`.

  Example usage:

  - `--split test` for evaluating the model on test split.

Optional arguments

- **CPU allocation**:

  `--num_workers`, if given, TensorFlow will use limited CPUs.

  By default, it uses all available CPUs.

  Example usage:

  – `--num_workers 2` for using at most 2 CPUs.

- **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.

- **Log directory**:

  `--log_dir`, specifies the log directory for logging output information and results.

  By default, it is `logs` under the package root.

  Example usage:

  – `--log_dir logs` for specifying the log directory `logs/` under current directory.

- **Experiment name**:

  `--exp_name` or `-n`, specifies the name of an experiment (every time a training or a prediction is run), which will be used together with the log directory (via `log_dir`) to specify the sub-folder that saves the output information and results from individual experiments (runs).

  If this is not provided, it creates a timestamp-named sub-folder under the `log_dir`, by default, e.g. `logs/20200810-194042/`.

  Example usage:

  – `--exp_name test --log_dir logs` for saving under `logs/test/`.

  – `--log_dir logs` for saving under `logs/20210101-120000/`, assuming `20210101-120000` is current time.

  – `--exp_name test` for saving under `DeepReg/logs/test/`, assuming `DeepReg` is the package root.

- **Batch size**:

  `--batch_size` or `-b`, specifies the number of samples per step for prediction. If using multiple GPUs, i.e. `n` GPUs, each GPU will have mini batch size `batch_size / n`. Thus, `batch_size` should be divided by `n` evenly.

  The default value is 1.

  Example usage:

  – `--batch_size 2` for using a global 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.
  - `--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

`deepreg_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 Nifti 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 Nifti 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 Logging

When running DeepReg, there are two types of printed messages:

1. DeepReg messages, which are entirely controlled by DeepReg code.
2. TensorFlow messages, which are automatically printed by TensorFlow during certain workflows.

Therefore, we use two independent environment variables to control these two types of messages. The log level controls which types of log messages would be printed. Further details are explained below.

3.12.1 DeepReg logging

DeepReg uses the *Python module ``logging`* <https://docs.python.org/3/library/logging.html>`_ to log the messages. The log level is controlled by the environment variable `DEEPREG_LOG_LEVEL`. The levels are given in the table below. The default level is "2".

To adjust the logging level, there are two options. Take the training as example,

- You can first define the environment variable, then run the job.

```
export DEEPREG_LOG_LEVEL=1
deepreg_train --gpu "" --config_path config/unpaired_labeled_ddf.yaml --exp_name␣
↪test
```

- Alternatively, you can define the environment variable while running the job.

```
DEEPREG_LOG_LEVEL=1 deepreg_train --gpu "" --config_path config/unpaired_labeled_
↪ddf.yaml --exp_name test
```

DEEPREG_LOG_LEVEL	Behavior
"0"	Log all messages, equivalent to `logging.DEBUG`. Same as log level "1".
"1"	Log all messages, equivalent to `logging.DEBUG`.
"2"	Log all messages except DEBUG, equivalent to `logging.INFO`. (default)
"3"	Log all messages except DEBUG and INFO, equivalent to `logging.WARNING`.
"4"	Log all messages except DEBUG, INFO, and WARNING, equivalent to `logging.ERROR`.
"5"	Log all messages except DEBUG, INFO, WARNING, and ERROR, equivalent to `logging.CRITICAL`.

3.12.2 TensorFlow logging

With TensorFlow 2.3, its log level is controlled by the environment variable `TF_CPP_MIN_LOG_LEVEL`. The levels are given in the table below. The default level is "2".

To adjust the logging level, there are two options. Take the training as example,

- You can first define the environment variable, then run the job.

```
export TF_CPP_MIN_LOG_LEVEL=1
deepreg_train --gpu "" --config_path config/unpaired_labeled_ddf.yaml --exp_name␣
↪test
```

- Alternatively, you can define the environment variable while running the job.

```
TF_CPP_MIN_LOG_LEVEL=1 deepreg_train --gpu "" --config_path config/unpaired_
↪labeled_ddf.yaml --exp_name test
```

TF_CPP_MIN_LOG_LEVEL	Behavior
"0"	Log all messages.
"1"	Log all messages except INFO.
"2"	Log all messages except INFO and WARNING. (default)
"3"	Log all messages except INFO, WARNING, and ERROR.

3.13 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.

3.13.1 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.

Split keys - Required

The data paths, data format, and label availability of the training, validation and testing data are specified under the corresponding sections separately:

```yaml
dataset:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
```

(continues on next page)

(continued from previous page)

```yaml
    format: "h5"
    labeled: true
```

For data paths, multiple dataset directories can be specified, such that data are sampled across several folders:

```yaml
dataset:
  train:
    dir:
      - "data/test/h5/paired/train1"
      - "data/test/h5/paired/train2"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
```

For data format, different formats requires different file structure and format. Thus different file loader will be used. Check the data loader configuration for more details.

Currently, DeepReg file 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:
  train:
    dir:
      - "data/test/nifti/paired/train1"
      - "data/test/nifti/paired/train2"
    format: "nifti"
    labeled: true
```

The `labeled` key indicates whether segmentation labels are available for training or evaluation. Use `true` and `false` to indicate the availability and unavailability correspondingly. In particular, if the value passed is false, the labels will not be used even if they are available in the associated directories.

```yaml
dataset:
  train:
    dir:
      - "data/test/nifti/paired/train1"
      - "data/test/nifti/paired/train2"
    format: "nifti"
    labeled: false # labels are not available
```

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:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  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`.

```yaml
dataset:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  type: "paired" # one of "paired", "unpaired" or "grouped"
  sample_label: "sample" # one of "sample", "all" or None
```

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`: Union[Tuple[int, . . .], List[int]] of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D moving image.
- `fixed_image_shape`: Union[Tuple[int, . . .], List[int]] of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D fixed image.

```yaml
dataset:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  type: "paired" # one of "paired", "unpaired" or "grouped"
  sample_label: "sample" # one of "sample", "all" or None
  moving_image_shape: [16, 16, 3]
  fixed_image_shape: [16, 16, 3]
```

Unpaired

- `image_shape`: Union[Tuple[int, . . .], List[int]] of ints, len 3, corresponding to (dim1, dim2, dim3) of the 3D image.

```yaml
dataset:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  type: "unpaired" # one of "paired", "unpaired" or "grouped"
  sample_label: "sample" # one of "sample", "all" or None
  image_shape: [16, 16, 3]
```

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`: Union[Tuple[int, ...], List[int]] len 3, corresponding to (dim1, dim2, dim3) of the 3D image.

```yaml
dataset:
  train:
    dir: "data/test/h5/paired/train"
    format: "h5"
    labeled: true
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  type: "grouped" # one of "paired", "unpaired" or "grouped"
  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.

3.13.2 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.

```yaml
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.

```
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "unet" # One of unet, local, global: networks currently supported by DeepReg
    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.

```
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.
    depth: 3
    pooling: false
    concat_skip: true
```

LocalNet

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.

- depth: Depth of the encoder, depth=2 means there are in total 3 layers where 0 is the top layer and 2 is the bottom.
- extract_levels: indices of layer from which the output will be extracted, the value range is [0, depth] both side inclusive.

```
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.
```

(continues on next page)

(continued from previous page)

```
    depth: 2
    extract_levels: [0, 1, 2]
```

GlobalNet

The GlobalNet has a U-net like encoder to encode the image and uses the bottleneck layer to output an affine transformation using a CNN.

- `depth`: Depth of the encoder, `depth=2` means there are in total 3 layers where 0 is the top layer and 2 is the bottom.

```
train:
  method: "ddf" # One of ddf, dvf, conditional
  backbone:
    name: "global" # One of unet, local, global
    num_channel_initial: 16 # Int type, number of initial channels in the network.␣
↪Controls the network size.
    depth: 4
```

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.

	DDF / DVF	Conditional
Image Loss	Applicable	Non-applicable
Label Loss	Applicable if data are labeled	Applicable
Regularization Loss	Applicable	Non-applicable

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".

```
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:
    image:
      name: "lncc" # other options include "lncc", "ssd" and "gmi", for local␣
↪normalised cross correlation,
      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 informatin 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.

```
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.
```

(continues on next page)

(continued from previous page)

```
    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.
  - `background_weight`: float, default=0.0. `background_weight` weights the foreground and background classes by replacing the labels of 1s and 0s with `(1-background_weight)` and `background_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.
  - `background_weight`: float, default=0.0. `background_weight` weights the foreground and background classes by replacing the labels of 1s and 0s with `(1-background_weight)` and `background_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.

```
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"
```

or

```
train:
  method: "ddf" # One of ddf, dvf, conditional
```

(continues on next page)

(continued from previous page)

```
  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: "gradient" # options include "bending", "gradient"
      l1: false
```

Composite Loss

The loss function can be a composite of different loss categories by adding all fields in the same configuration file.

```
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:
    image:
      name: "gmi"
      weight: 1.0
    label:
      weight: 1.0
      name: "dice"
```

Moreover, one may want to specify several loss functions for each category. In that case, a dashed line (-) indicates the specification of a new loss function under each field.

E.G.

```
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:
    image:
      - name: "lncc"
        weight: 0.5
        kernel_size: 5
      - name: "gmi"
        weight: 0.5
    label:
      name: "dice"
      weight: 0.5
```

Optimizer - required

The optimizer can be defined by using a `name` with other required arugment. The name must be the same to the class name under `tf.keras.optimizers`.

For instance, to use a default Keras Adam optimizer, the configuration should be

```
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: "Adam"
```

For a Keras SGD optimizer with learning rate 0.001, the configuration should be

```
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"
    learning_rate: 0.001
```

Preprocess - required

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

- `batch_size`: int, specifies the number of samples per step for prediction. If using multiple GPUs, i.e. `n` GPUs, each GPU will have mini batch size `batch_size / n`. Thus, `batch_size` should be divided by `n` evenly.
- `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.
- `num_parallel_calls`: int, it defines the number of cpus used during preprocessing, -1 means unlimited and it may take all cpus and significantly more memory.

```
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.
```

(continues on next page)

(continued from previous page)

```
    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
    num_parallel_calls: -1 # number elements to process asynchronously in parallel
↪during preprocessing, -1 means unlimited, heuristically it should be set to the
↪number of CPU cores available
```

Epochs - required

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

```
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
```

Saving frequency - required

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
```

3.14 Dataset Loader

3.14.1 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.

3.14.2 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-min(x)+EPS) / (max(x)-min(x)+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 `(dim1, dim2, dim3)`, its label shape can be `(dim1, dim2, dim3)` (single label) or `(dim1, dim2, dim3, 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 `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 or partially labelled data, consider using all-zero masks as a workaround.
  - See further discussion in Label sampling.

3.14.3 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 `(m_dim1, m_dim2, m_dim3)`
  - fixed image of shape `(f_dim1, f_dim2, f_dim3)`
- When the image is labeled and there is only one label,
  - moving image of shape `(m_dim1, m_dim2, m_dim3)`
  - fixed image of shape `(f_dim1, f_dim2, f_dim3)`
  - moving label of shape `(m_dim1, m_dim2, m_dim3)`
  - fixed label of shape `(f_dim1, f_dim2, f_dim3)`
- When the image is labeled and there are multiple labels,

- moving image of shape `(m_dim1, m_dim2, m_dim3)`
- fixed image of shape `(f_dim1, f_dim2, f_dim3)`
- moving label of shape `(m_dim1, m_dim2, m_dim3, num_labels)`
- fixed label of shape `(f_dim1, f_dim2, f_dim3, 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:
  train:
    dir: "data/test/h5/paired/train" # folder containing data
    format: "h5" # nifti or h5
    labeled: true # true or false
  valid:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  type: "paired" # value should be paired / unpaired / grouped
  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:

```yaml
dataset:
  train:
    dir: # folders containing data
      - "data/test/h5/paired/train1"
      - "data/test/h5/paired/train2"
    format: "h5" # nifti or h5
    labeled: true # true or false
  valid:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  type: "paired" # value should be paired / unpaired / grouped
  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]
```

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

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.

3.14.4 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:
  train:
    dir: "data/test/h5/paired/train" # folder containing data
    format: "h5" # nifti or h5
    labeled: true # true or false
  valid:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/unpaired/test"
    format: "h5"
    labeled: true
  type: "unpaired" # value should be paired / unpaired / grouped
  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.

`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/
  - 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.

3.14.5 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 with `sample_image_in_group` set to false.

Configuration

An example configuration for grouped dataset is provided as follows.

```
dataset:
  train:
    dir: "data/test/h5/paired/train" # folder containing data
    format: "h5" # nifti or h5
    labeled: true # true or false
  valid:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  test:
    dir: "data/test/h5/paired/test"
    format: "h5"
    labeled: true
  type: "unpaired" # value should be paired / unpaired / grouped
  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.

3.15 Registry

DeepReg adopts the registry system to facilitate the addition of custom functionalities.

3.15.1 Description

The class `Registry` maintains a dictionary mapping `(category, key)` to `value`, where

- `category` is the class category, e.g. `"backbone_class""` for backbone classes.
- `key` is the name of the registered class, e.g. `"unet"` for the class `UNet`.
- `value` is the registered class, e.g. `UNet` corresponding to `"unet"`.

A global variable `REGISTRY = Registry()` is defined to provide a central control of all classes. The supported categories and the registered classes are resumed in the registered classes page.

Register a class

To register a class into REGISTRY, it is recommended to use register as a **decorator**. Consider the UNet class for backbone as an example.

```
from deepreg.registry import REGISTRY


@REGISTRY.register(category="backbone_class", name="unet")
class UNet:
    """UNet-style backbone."""
```

The decorator automatically registers the class upon import. To ensure that this class is registered when import deepreg is called, this class and related parent modules need to be imported in the __init__.py files.

For the purpose of code simplicity, a specific register function is defined for each category. For instance, we can use register_backbone for UNet:

```
from deepreg.registry import REGISTRY


@REGISTRY.register_backbone(name="unet")
class UNet:
    """UNet-style backbone."""
```

Instantiate a class

To instantiate a registered example in REGISTRY, we call build_from_config , which allows creating a class instance from a config directly. The config should be a dictionary containing the key name and other required keys for the class. Please check the configuration documentation and the docstring of the related classes for detailed configuration requirements. The path of the registered classes are resumed in the registered classes page.

For instance, to instantiate a UNet class,

```
from deepreg.registry import REGISTRY

config = dict(name="unet",
              image_size=(16,16,16),
              out_channels=3,
              num_channel_initial=2,
              out_kernel_initializer="he_normal",
              out_activation="")
unet = REGISTRY.build_from_config(category="backbone_class", config=config)
```

where kwargs represents other required arguments.

Similarly, for the purpose of code simplicity, a specific build function is defined for each category. For instance, we can use build_backbone for UNet:

```
from deepreg.registry import REGISTRY


config = dict(name="unet",
              image_size=(16,16,16),
              out_channels=3,
              num_channel_initial=2,
              out_kernel_initializer="he_normal",
```

(continues on next page)

(continued from previous page)

```
                out_activation="")
unet = REGISTRY.build_backbone(config=config)
```

3.15.2 Example usages

Apart from the table of registered classes, to further explain how to use `REGISTRY` for using customized classes, detailed examples are provided for the following categories:

- backbone
- loss

Custom backbone

To register a custom backbone class, the steps are as follows

1. Subclass the `Backbone` and implement a custom backbone class.
2. Import `REGISTRY` and use the decorator `@REGISTRY.register_backbone` to register the custom class.
3. Use the registered name in the config for using the registered custom backbone.

Please check the self-contained example script for further details.

Custom Loss

To register a custom loss class for images and labels, the steps are as follows

1. Subclass the `tf.keras.losses.Loss` and implement a custom backbone class.
2. Import `REGISTRY` and use the decorator `@REGISTRY.register_loss` to register the custom class.
3. Use the registered name in the config for using the registered custom loss.

Please check the self-contained example script for further details. There is also a more complicated example of parameterized custom loss.

3.16 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

3.16.1 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 or missing labels

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

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.

3.17 Entry Point

3.17.1 Train

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

deepreg.train.**build_config**(*config_path: Union[str, List[str]]*, *log_dir: str*, *exp_name: str*, *ckpt_path: str*, *max_epochs: int = - 1*) → Tuple[Dict, str, 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_dir** – path of the log directory
- **exp_name** – name of the experiment
- **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
- exp_name: the path of directory to save logs

deepreg.train.**main**(*args=None*)

Entry point for train script.

Parameters **args** – arguments

deepreg.train.**train**(*gpu: str*, *config_path: Union[str, List[str]]*, *ckpt_path: str*, *num_workers: int = 1*, *gpu_allow_growth: bool = True*, *exp_name: str = ''*, *log_dir: str = 'logs'*, *max_epochs: int = - 1*)

Function to train a model.

Parameters

- **gpu** – which local gpu to use to train.
- **config_path** – path to configuration set up.
- **ckpt_path** – where to store training checkpoints.
- **num_workers** – number of cpu cores to be used, <=0 means not limited.
- **gpu_allow_growth** – whether to allocate whole GPU memory for training.
- **log_dir** – path of the log directory.
- **exp_name** – experiment name.
- **max_epochs** – if max_epochs > 0, will use it to overwrite the configuration.

3.17.2 Predict

Module to perform predictions on data using command line interface.

`deepreg.predict.`**`build_config`**(*config_path: Union[str, List[str]]*, *log_dir: str*, *exp_name: str*, *ckpt_path: str*) → Tuple[Dict, str, str]

Function to create new directory to log directory to store results.

Parameters

- **`config_path`** – path of configuration files.
- **`log_dir`** – path of the log directory.
- **`exp_name`** – experiment name.
- **`ckpt_path`** – path where model is stored.

Returns

- config, configuration dictionary.
- exp_name, path of the directory for saving outputs.

`deepreg.predict.`**`build_pair_output_path`**(*indices: list*, *save_dir: str*) → Tuple[str, 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

`deepreg.predict.`**`main`**(*args=None*)

Entry point for predict script.

Parameters **`args`** –

`deepreg.predict.`**`predict`**(*gpu: str*, *ckpt_path: str*, *split: str*, *batch_size: int*, *exp_name: str*, *config_path: Union[str, List[str]]*, *num_workers: int = 1*, *gpu_allow_growth: bool = True*, *save_nifti: bool = True*, *save_png: bool = True*, *log_dir: str = 'logs'*)

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

Parameters

- **`gpu`** – which env gpu to use.
- **`ckpt_path`** – where model is stored, should be like log_folder/save/ckpt-x.
- **`split`** – train / valid / test, to define the split to be evaluated.
- **`batch_size`** – int, batch size to perform predictions.
- **`exp_name`** – name of the experiment.
- **`config_path`** – to overwrite the default config.
- **`num_workers`** – number of cpu cores to be used, <=0 means not limited.
- **`gpu_allow_growth`** – whether to allocate whole GPU memory for training.
- **`save_nifti`** – if true, outputs will be saved in nifti format.

- **save_png** – if true, outputs will be saved in png format.
- **log_dir** – path of the log directory.

deepreg.predict.**predict_on_dataset**(*dataset: tensorflow.python.data.ops.dataset_ops.DatasetV2*, *fixed_grid_ref: tensorflow.python.framework.ops.Tensor*, *model: tensorflow.python.keras.engine.training.Model*, *save_dir: str*, *save_nifti: bool*, *save_png: bool*)

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
- **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

3.17.3 Warp

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

deepreg.warp.**main**(*args=None*)

Entry point for warp script.

Parameters **args** –

deepreg.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

3.18 Dataset Loader

3.18.1 Paired Loader

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

class deepreg.dataset.loader.paired_loader.**PairedDataLoader**(*file_loader, data_dir_paths: List[str], labeled: bool, sample_label: str, seed, moving_image_shape: Union[Tuple[int, ...], List[int]], fixed_image_shape: Union[Tuple[int, ...], List[int]]*)

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.

3.18.2 Unpaired Loader

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

class deepreg.dataset.loader.unpaired_loader.**UnpairedDataLoader**(*file_loader, data_dir_paths: List[str], labeled: bool, sample_label: str, seed: int, image_shape: Union[Tuple[int, ...], List[int]]*)

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).

3.18.3 Grouped Loader

Load grouped data. Supported formats: h5 and Nifti. Image data can be labeled or unlabeled. Read https://deepreg.readthedocs.io/en/latest/api/loader.html#module-deepreg.dataset.loader.grouped_loader for more details.

class deepreg.dataset.loader.grouped_loader.**GroupedDataLoader**(*file_loader, data_dir_paths: List[str], labeled: bool, sample_label: Optional[str], intra_group_prob: float, intra_group_option: str, sample_image_in_group: bool, seed: Optional[int], image_shape: Union[Tuple[int, ...], List[int]]*)

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: 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 - sum(ni * (ni-1) / 2) for forward/backward - 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.

3.19 File Loader

3.19.1 Interface

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: Union[int, Tuple[int, ...]]*) → 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 the data array at the specified index

get_data_ids() → List

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]

3.19.2 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: Union[int, Tuple[int, …]]*) → 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() → List

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() → int

Returns 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]

deepreg.dataset.loader.nifti_loader.**load_nifti_file**(*file_path: str*) → numpy.ndarray

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

Returns return the numpy array

3.19.3 H5 Loader

Load h5 files and associated information.

class deepreg.dataset.loader.h5_loader.**H5FileLoader**(*dir_paths: List[str]*, *name: str*, *grouped: 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: Union[int, Tuple[int, ...]]*) → 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() → List

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

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

get_num_images() → 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].

3.20 Registry

class deepreg.registry.**Registry**

Registry maintains a dictionary which maps *(category, key)* to *value*.

Multiple __init__.py files have been modified so that the classes are registered when executing:

```
from deepreg.registry import REGISTRY
```

References:

- https://github.com/ray-project/ray/blob/00ef1179c012719a17c147a5c3b36d6bdbe97195/python/ray/tune/registry.py#L108
- https://github.com/open-mmlab/mmdetection/blob/master/mmdet/models/builder.py
- https://github.com/open-mmlab/mmcv/blob/master/mmcv/utils/registry.py
- https://towardsdatascience.com/whats-init-for-me-d70a312da583

Init registry with empty dict.

_register(*category: str*, *key: str*, *value: Callable*, *force: bool*)

Registers the value with the registry.

Parameters

- **category** – name of the class category
- **key** – unique identity
- **value** – class to be registered
- **force** – if True, overwrite the existing value in case the key has been registered.

build_backbone(*config: Dict*, *default_args: Optional[dict] = None*) → Any

Instantiate a registered backbone class.

Parameters

- **config** – config having key *name*.
- **default_args** – optionally some default arguments.

Returns a backbone instance

build_data_augmentation(*config: Dict*, *default_args: Optional[dict] = None*) → Callable

Instantiate a registered data augmentation class.

Parameters

- **config** – config having key *name*.
- **default_args** – optionally some default arguments.

Returns a data augmentation instance

build_data_loader(*config: Dict*, *default_args: Optional[dict] = None*) → Any

Instantiate a registered data loader class.

Parameters

- **config** – config having key *name*.
- **default_args** – optionally some default arguments.

Returns a loss instance

build_from_config (*category: str*, *config: Dict*, *default_args: Optional[dict] = None*) → Any
Build a class instance from config dict.

Parameters

- **category** – category name.
- **config** – a dict which must contain the key "name".
- **default_args** – optionally some default arguments.

Returns the instantiated class.

build_loss (*config: Dict*, *default_args: Optional[dict] = None*) → Callable
Instantiate a registered loss class.

Parameters

- **config** – config having key *name*.
- **default_args** – optionally some default arguments.

Returns a loss instance

build_model (*config: Dict*, *default_args: Optional[dict] = None*) → Any
Instantiate a registered model class.

Parameters

- **config** – config having key *name*.
- **default_args** – optionally some default arguments.

Returns a model instance

contains (*category: str*, *key: str*) → bool
Verify if the key has been registered for the category.

Parameters

- **category** – category name.
- **key** – value name.

Returns *True* if registered.

copy ()
Deep copy the registry.

get (*category: str*, *key: str*) → Callable
Return the registered class.

Parameters

- **category** – category name.
- **key** – value name.

Returns registered value.

register (*category: str*, *name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a py class. A record will be added to *self._dict*, whose key is the class name or the specified name, and value is the class itself. It can be used as a decorator or a normal function.

Parameters

- **category** – The type of the category.
- **name** – The class name to be registered. If not specified, the class name will be used.

- **force** – Whether to override an existing class with the same name.
- **cls** – Class to be registered.

Returns The given class or a decorator.

register_backbone (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a backbone class.

Parameters

- **name** – backbone name
- **cls** – backbone class
- **force** – whether overwrite if already registered

Returns the registered class

register_data_augmentation (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a data augmentation class.

Parameters

- **name** – data augmentation name
- **cls** – data augmentation class
- **force** – whether overwrite if already registered

Returns the registered class

register_data_loader (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a data loader class.

Parameters

- **name** – loss name
- **cls** – loss class
- **force** – whether overwrite if already registered

Returns the registered class

register_file_loader (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a file loader class.

Parameters

- **name** – loss name
- **cls** – loss class
- **force** – whether overwrite if already registered

Returns the registered class

register_loss (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a loss class.

Parameters

- **name** – loss name
- **cls** – loss class

- **force** – whether overwrite if already registered

Returns the registered class

register_model (*name: str*, *cls: Optional[Callable] = None*, *force: bool = False*) → Callable
Register a model class.

Parameters

- **name** – model name
- **cls** – model class
- **force** – whether overwrite if already registered

Returns the registered class

3.21 Network

class deepreg.model.network.**ConditionalModel** (**args: Any*, ***kwargs: Any*)
A registration model predicts fixed image label without DDF or DVF.

Init.

Parameters

- **moving_image_size** – (m_dim1, m_dim2, m_dim3)
- **fixed_image_size** – (f_dim1, f_dim2, f_dim3)
- **index_size** – number of indices for identify each sample
- **labeled** – if the data is labeled
- **batch_size** – total number of samples consumed per step, over all devices. When using multiple devices, TensorFlow automatically split the tensors. Therefore, input shapes should be defined over batch_size.
- **config** – config for method, backbone, and loss.
- **name** – name of the model

build_model ()
Build the model to be saved as self._model.

postprocess (*inputs: Dict[str, tensorflow.Tensor]*, *outputs: Dict[str, tensorflow.Tensor]*) → Tuple[tensorflow.Tensor, Dict]
Return a dict used for saving inputs and outputs.

Parameters

- **inputs** – dict of model inputs
- **outputs** – dict of model outputs

Returns tuple, indices and a dict. In the dict, each value is (tensor, normalize, on_label), where - normalize = True if the tensor need to be normalized to [0, 1] - on_label = True if the tensor depends on label

class deepreg.model.network.**DDFModel** (**args: Any*, ***kwargs: Any*)
A registration model predicts DDF.

When using global net as backbone, the model predicts an affine transformation parameters, and a DDF is calculated based on that.

Init.

Parameters

- **moving_image_size** – (m_dim1, m_dim2, m_dim3)
- **fixed_image_size** – (f_dim1, f_dim2, f_dim3)
- **index_size** – number of indices for identify each sample
- **labeled** – if the data is labeled
- **batch_size** – total number of samples consumed per step, over all devices. When using multiple devices, TensorFlow automatically split the tensors. Therefore, input shapes should be defined over batch_size.
- **config** – config for method, backbone, and loss.
- **name** – name of the model

build_loss()

Build losses according to configs.

build_model()

Build the model to be saved as self._model.

postprocess(*inputs: Dict[str, tensorflow.Tensor]*, *outputs: Dict[str, tensorflow.Tensor]*) → Tuple[tensorflow.Tensor, Dict]

Return a dict used for saving inputs and outputs.

Parameters

- **inputs** – dict of model inputs
- **outputs** – dict of model outputs

Returns tuple, indices and a dict. In the dict, each value is (tensor, normalize, on_label), where - normalize = True if the tensor need to be normalized to [0, 1] - on_label = True if the tensor depends on label

class deepreg.model.network.**DVFModel**(**args: Any*, ***kwargs: Any*)

A registration model predicts DVF.

DDF is calculated based on DVF.

Init.

Parameters

- **moving_image_size** – (m_dim1, m_dim2, m_dim3)
- **fixed_image_size** – (f_dim1, f_dim2, f_dim3)
- **index_size** – number of indices for identify each sample
- **labeled** – if the data is labeled
- **batch_size** – total number of samples consumed per step, over all devices. When using multiple devices, TensorFlow automatically split the tensors. Therefore, input shapes should be defined over batch_size.
- **config** – config for method, backbone, and loss.
- **name** – name of the model

build_loss()

Build losses according to configs.

build_model()
Build the model to be saved as self._model.

postprocess(*inputs: Dict[str, tensorflow.Tensor]*, *outputs: Dict[str, tensorflow.Tensor]*) → Tuple[tensorflow.Tensor, Dict]
Return a dict used for saving inputs and outputs.

Parameters

- **inputs** – dict of model inputs
- **outputs** – dict of model outputs

Returns tuple, indices and a dict. In the dict, each value is (tensor, normalize, on_label), where - normalize = True if the tensor need to be normalized to [0, 1] - on_label = True if the tensor depends on label

class deepreg.model.network.**RegistrationModel**(**args: Any*, ***kwargs: Any*)
Interface for registration model.

Init.

Parameters

- **moving_image_size** – (m_dim1, m_dim2, m_dim3)
- **fixed_image_size** – (f_dim1, f_dim2, f_dim3)
- **index_size** – number of indices for identify each sample
- **labeled** – if the data is labeled
- **batch_size** – total number of samples consumed per step, over all devices. When using multiple devices, TensorFlow automatically split the tensors. Therefore, input shapes should be defined over batch_size.
- **config** – config for method, backbone, and loss.
- **name** – name of the model

build_inputs() → Dict[str, tensorflow.keras.layers.Input]
Build input tensors.

Returns dict of inputs.

abstract build_loss()
Build losses according to configs.

abstract build_model()
Build the model to be saved as self._model.

call(*inputs: Dict[str, tensorflow.Tensor]*, *training=None*, *mask=None*) → Dict[str, tensorflow.Tensor]
Call the self._model.

Parameters

- **inputs** – a dict of tensors.
- **training** – training or not.
- **mask** – maks for inputs.

Returns

concat_images(*moving_image: tensorflow.Tensor*, *fixed_image: tensorflow.Tensor*, *moving_label: Optional[tensorflow.Tensor] = None*) → tensorflow.Tensor
Adjust image shape and concatenate them together.

Parameters

- **moving_image** – registration source
- **fixed_image** – registration target
- **moving_label** – optional, only used for conditional model.

Returns

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

log_tensor_stats(*tensor: tensorflow.Tensor*, *name: str*)
Log statistics of a given tensor.

Parameters

- **tensor** – tensor to monitor.
- **name** – name of the tensor.

plot_model(*output_dir: str*)
Save model structure in png.

Parameters **output_dir** – path to the output dir.

abstract postprocess(*inputs: Dict[str, tensorflow.Tensor]*, *outputs: Dict[str, tensorflow.Tensor]*) → Tuple[tensorflow.Tensor, Dict]
Return a dict used for saving inputs and outputs.

Parameters

- **inputs** – dict of model inputs
- **outputs** – dict of model outputs

Returns tuple, indices and a dict. In the dict, each value is (tensor, normalize, on_label), where - normalize = True if the tensor need to be normalized to [0, 1] - on_label = True if the tensor depends on label

deepreg.model.network.**dict_without**(*d: dict*, *key*) → dict
Return a copy of the given dict without a certain key.

Parameters

- **d** – dict to be copied.
- **key** – key to be removed.

Returns the copy without a key

3.22 Backbone

3.22.1 Interface

3.22.2 Local Net

class deepreg.model.backbone.local_net.**LocalNet**(**args: Any*, ***kwargs: Any*)
Build LocalNet for image registration.

Reference:

- Hu, Yipeng, et al. "Weakly-supervised convolutional neural networks for multimodal image registration." Medical image analysis 49 (2018): 1-13. https://doi.org/10.1016/j.media.2018.07.002
- Hu, Yipeng, et al. "Label-driven weakly-supervised learning for multimodal deformable image registration," https://arxiv.org/abs/1711.01666

Init.

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

So, extract_levels are between [0, D].

Parameters

- **image_size** – such as (dim1, dim2, dim3)
- **num_channel_initial** – number of initial channels.
- **extract_levels** – from which depths the output will be built.
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.
- **out_channels** – number of channels for the extractions
- **depth** – depth of the encoder. If depth is not given, depth = max(extract_levels) will be used.
- **use_additive_upsampling** – whether use additive up-sampling layer for decoding.
- **pooling** – for down-sampling, use non-parameterized pooling if true, otherwise use conv3d
- **concat_skip** – when up-sampling, concatenate skipped tensor if true, otherwise use addition
- **name** – name of the backbone.
- **kwargs** – additional arguments.

build_bottom_block (*filters: int*, *kernel_size: int*, *padding: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for bottom layer.

This block do not change the tensor shape (width, height, depth), it only changes the number of channels.

Parameters

- **filters** – number of channels for output
- **kernel_size** – arg for conv3d
- **padding** – arg for conv3d

Returns a block consists of one or multiple layers

build_output_block (*image_size: Tuple[int, ...]*, *extract_levels: Tuple[int, ...]*, *out_channels: int*, *out_kernel_initializer: str*, *out_activation: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for output.

The input to this block is a list of tensors.

Parameters

- **image_size** – such as (dim1, dim2, dim3)

- **extract_levels** – number of extraction levels.
- **out_channels** – number of channels for the extractions
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.

Returns a block consists of one or multiple layers

build_up_sampling_block (*filters: int*, *output_padding: Union[Tuple[int, ...], int]*, *kernel_size: Union[Tuple[int, ...], int]*, *padding: str*, *strides: Union[Tuple[int, ...], int]*, *output_shape: Tuple[int, ...]*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for up-sampling.

This block changes the tensor shape (width, height, depth), but it does not changes the number of channels.

Parameters

- **filters** – number of channels for output
- **output_padding** – padding for output
- **kernel_size** – arg for deconv3d
- **padding** – arg for deconv3d
- **strides** – arg for deconv3d
- **output_shape** – shape of the output tensor

Returns a block consists of one or multiple layers

get_config () → dict

Return the config dictionary for recreating this class.

3.22.3 Global Net

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

Build GlobalNet for image registration.

GlobalNet is a special UNet where the decoder for up-sampling is skipped. The network's outputs come from the bottom layer from the encoder directly.

Reference:

- Hu, Yipeng, et al. "Label-driven weakly-supervised learning for multimodal deformable image registration," https://arxiv.org/abs/1711.01666

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)
- **num_channel_initial** – int, number of initial channels
- **extract_levels** – list, which levels from net to extract, deprecated. If depth is not given, depth = max(extract_levels) will be used.
- **depth** – depth of the encoder. If given, extract_levels is not used.
- **name** – name of the backbone.
- **kwargs** – additional arguments.

build_output_block(*image_size: Tuple[int, ...], extract_levels: Tuple[int, ...], out_channels: int, out_kernel_initializer: str, out_activation: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for output.

The input to this block is a list of length 1. The output has two tensors.

Parameters

- **image_size** – such as (dim1, dim2, dim3)
- **extract_levels** – not used
- **out_channels** – not used
- **out_kernel_initializer** – not used
- **out_activation** – not used

Returns a block consists of one or multiple layers

3.22.4 U-Net

class deepreg.model.backbone.u_net.**UNet**(**args: Any, **kwargs: Any*)

Class that implements an adapted 3D UNet.

Reference:

- O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation,", Lecture Notes in Computer Science, 2015, vol. 9351, pp. 234–241. https://arxiv.org/abs/1505.04597

Initialise UNet.

Parameters

- **image_size** – (dim1, dim2, dim3), dims of input image.
- **num_channel_initial** – number of initial channels
- **depth** – input is at level 0, bottom is at level depth.
- **out_kernel_initializer** – kernel initializer for the last layer
- **out_activation** – activation at the last layer
- **out_channels** – number of channels for the output
- **extract_levels** – list, which levels from net to extract.
- **pooling** – for down-sampling, use non-parameterized pooling if true, otherwise use conv3d
- **concat_skip** – when up-sampling, concatenate skipped tensor if true, otherwise use addition
- **encode_kernel_sizes** – kernel size for down-sampling
- **decode_kernel_sizes** – kernel size for up-sampling
- **encode_num_channels** – filters/channels for down-sampling, by default it is doubled at each layer during down-sampling
- **decode_num_channels** – filters/channels for up-sampling, by default it is the same as encode_num_channels

- **strides** – strides for down-sampling
- **padding** – padding mode for all conv layers
- **name** – name of the backbone.
- **kwargs** – additional arguments.

build_bottom_block (*filters: int*, *kernel_size: int*, *padding: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for bottom layer.

This block do not change the tensor shape (width, height, depth), it only changes the number of channels.

Parameters

- **filters** – number of channels for output
- **kernel_size** – arg for conv3d
- **padding** – arg for conv3d

Returns a block consists of one or multiple layers

build_decode_conv_block (*filters: int*, *kernel_size: int*, *padding: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a conv block for up-sampling

This block do not change the tensor shape (width, height, depth), it only changes the number of channels.

Parameters

- **filters** – number of channels for output
- **kernel_size** – arg for conv3d
- **padding** – arg for conv3d

Returns a block consists of one or multiple layers

build_decode_layers (*tensor_shapes: List[Tuple]*, *image_size: Tuple*, *num_channels: Tuple*, *depth: int*, *extract_levels: Tuple[int, ...]*, *decode_kernel_sizes: Union[int, List[int]]*, *strides: int*, *padding: str*, *out_kernel_initializer: str*, *out_activation: str*, *out_channels: int*)

Build layers for decoding.

Parameters

- **tensor_shapes** – shapes calculated in encoder
- **image_size** – (dim1, dim2, dim3).
- **num_channels** – number of channels for each layer, starting from the top layer.
- **depth** – network starts with d = 0, and the bottom has d = depth.
- **extract_levels** – from which depths the output will be built.
- **decode_kernel_sizes** – kernel size for up-sampling
- **strides** – strides for down-sampling
- **padding** – padding mode for all conv layers
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.
- **out_channels** – number of channels for the extractions

build_down_sampling_block (*filters: int*, *kernel_size: int*, *padding: str*, *strides: int*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for down-sampling.

This block changes the tensor shape (width, height, depth), but it does not changes the number of channels.

Parameters

- **filters** – number of channels for output, arg for conv3d
- **kernel_size** – arg for pool3d or conv3d
- **padding** – arg for pool3d or conv3d
- **strides** – arg for pool3d or conv3d

Returns a block consists of one or multiple layers

build_encode_conv_block (*filters: int*, *kernel_size: int*, *padding: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a conv block for down-sampling

This block do not change the tensor shape (width, height, depth), it only changes the number of channels.

Parameters

- **filters** – number of channels for output
- **kernel_size** – arg for conv3d
- **padding** – arg for conv3d

Returns a block consists of one or multiple layers

build_encode_layers (*image_size: Tuple*, *num_channels: Tuple*, *depth: int*, *encode_kernel_sizes: Union[int, List[int]]*, *strides: int*, *padding: str*) → List[Tuple]

Build layers for encoding.

Parameters

- **image_size** – (dim1, dim2, dim3).
- **num_channels** – number of channels for each layer, starting from the top layer.
- **depth** – network starts with d = 0, and the bottom has d = depth.
- **encode_kernel_sizes** – kernel size for down-sampling
- **strides** – strides for down-sampling
- **padding** – padding mode for all conv layers

Returns list of tensor shapes starting from d = 0

build_layers (*image_size: tuple*, *num_channel_initial: int*, *depth: int*, *extract_levels: Tuple[int, ...]*, *encode_kernel_sizes: Union[int, List[int]]*, *decode_kernel_sizes: Union[int, List[int]]*, *encode_num_channels: Optional[Tuple]*, *decode_num_channels: Optional[Tuple]*, *strides: int*, *padding: str*, *out_kernel_initializer: str*, *out_activation: str*, *out_channels: int*)

Build layers that will be used in call.

Parameters

- **image_size** – (dim1, dim2, dim3).
- **num_channel_initial** – number of initial channels.
- **depth** – network starts with d = 0, and the bottom has d = depth.

- **extract_levels** – from which depths the output will be built.
- **encode_kernel_sizes** – kernel size for down-sampling
- **decode_kernel_sizes** – kernel size for up-sampling
- **encode_num_channels** – filters/channels for down-sampling, by default it is doubled at each layer during down-sampling
- **decode_num_channels** – filters/channels for up-sampling, by default it is the same as encode_num_channels
- **strides** – strides for down-sampling
- **padding** – padding mode for all conv layers
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.
- **out_channels** – number of channels for the extractions

build_output_block(*image_size: Tuple[int, ...]*, *extract_levels: Tuple[int, ...]*, *out_channels: int*, *out_kernel_initializer: str*, *out_activation: str*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for output.

The input to this block is a list of tensors.

Parameters

- **image_size** – such as (dim1, dim2, dim3)
- **extract_levels** – number of extraction levels.
- **out_channels** – number of channels for the extractions
- **out_kernel_initializer** – initializer to use for kernels.
- **out_activation** – activation to use at end layer.

Returns a block consists of one or multiple layers

build_skip_block() → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for combining skipped tensor and up-sampled one.

This block do not change the tensor shape (width, height, depth), it only changes the number of channels.

The input to this block is a list of tensors.

Returns a block consists of one or multiple layers

build_up_sampling_block(*filters: int*, *output_padding: Union[Tuple[int, ...], int]*, *kernel_size: Union[Tuple[int, ...], int]*, *padding: str*, *strides: Union[Tuple[int, ...], int]*, *output_shape: Tuple[int, ...]*) → Union[tensorflow.keras.Model, tensorflow.keras.layers.Layer]

Build a block for up-sampling.

This block changes the tensor shape (width, height, depth), but it does not changes the number of channels.

Parameters

- **filters** – number of channels for output
- **output_padding** – padding for output
- **kernel_size** – arg for deconv3d
- **padding** – arg for deconv3d

- **strides** – arg for deconv3d
- **output_shape** – shape of the output tensor

Returns a block consists of one or multiple layers

call (*inputs: tensorflow.Tensor*, *training=None*, *mask=None*) → tensorflow.Tensor
Build compute 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)

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

3.23 Layer

3.23.1 Layer

Activation

AdditiveUpSampling

Conv3d

Conv3dBlock

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

Init.

Parameters

- **name** – name of the layer
- **kwargs** – additional arguments.

Conv3dWithResize

Deconv3d

Deconv3dBlock

class deepreg.model.layer.**Deconv3dBlock** (**args: Any*, ***kwargs: Any*)
A deconv3d block having conv3d - norm - activation.

Init.

Parameters

- **name** – name of the layer

- **kwargs** – additional arguments.

Dense

DownSampleResnetBlock

IntDVF

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

Integrate DVF to get DDF.

Reference:

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

Init.

Parameters

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

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

Parameters

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

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

get_config() → dict

Return the config dictionary for recreating this class.

LocalNetResidual3dBlock

LocalNetUpSampleResnetBlock

MaxPool3d

Norm

Residual3dBlock

UpSampleResnetBlock

Warping

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

Warps an image with DDF.

Reference:

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

Init.

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
- **name** – name of the layer
- **kwargs** – additional arguments.

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

Parameters

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

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

get_config() → dict

Return the config dictionary for recreating this class.

3.23.2 Util

Module containing utilities for layer inputs

deepreg.model.layer_util.**deconv_output_padding**(*input_shape: Union[Tuple[int, ...], int]*, *output_shape: Union[Tuple[int, ...], int]*, *kernel_size: Union[Tuple[int, ...], int]*, *stride: Union[Tuple[int, ...], int]*, *padding: str*) → Union[Tuple[int, ...], int]

Calculate output padding for Conv3DTranspose in any dimension.

Parameters

- **input_shape** – shape of Conv3DTranspose input tensor, without batch or channel
- **output_shape** – shape of Conv3DTranspose output tensor, without batch or channel
- **kernel_size** – kernel size of Conv3DTranspose layer
- **stride** – stride of Conv3DTranspose layer
- **padding** – padding of Conv3DTranspose layer

Returns output_padding for Conv3DTranspose layer

deepreg.model.layer_util.**gaussian_filter_3d**(*kernel_sigma: Union[Tuple, List]*) → 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.**get_n_bits_combinations**(*num_bits: int*) → List[List[int]]

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

```
>>> 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: Union[Tuple[int, . . .], List[int]]*) → tensorflow.Tensor

Generate a 3D grid with given size.

Reference:

- volshape_to_meshgrid of neuron https://github.com/adalca/neurite/blob/legacy/neuron/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

```
[[0, 0, 0], # even
 [0, 0, 1], # odd
 [0, 1, 0], # even
 [0, 1, 1], # odd
 [1, 0, 0], # even
 [1, 0, 1], # odd
 [1, 1, 0], # even
 [1, 1, 1]] # odd
```

values[::2] correspond to the corners with last coordinate == 0

```
[[0, 0, 0],
 [0, 1, 0],
 [1, 0, 0],
 [1, 1, 0]]
```

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

```
[[0, 0, 1],
 [0, 1, 1],
 [1, 0, 1],
 [1, 1, 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) * f1 + x * c1)
      * ((1-y) * f2 + y * c2)
      * ((1-z) * f3 + z * c3)
```

Let

```
W_xy = ((1-x) * f1 + x * c1)
     * ((1-y) * f2 + y * c2)
```

Then

- W_xy0 = W_xy * f3
- W_xy1 = W_xy * c3

Similar to W_xyz, denote V_xyz the value at (x, y, z), the final sum V equals

```
  sum over x,y,z (V_xyz * W_xyz)
= sum over x,y (V_xy0 * W_xy0 + V_xy1 * W_xy1)
= sum over x,y (V_xy0 * W_xy * f3 + V_xy1 * W_xy * c3)
= sum over x,y (V_xy0 * W_xy) * f3 + sum over x,y (V_xy1 * W_xy) * 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) `deepreg.model.layer_util.resample(vol: tensorflow.Tensor, loc: tensorflow.Tensor, interpolation: str = 'linear', zero_boundary: bool = True) → tensorflow.Tensor`

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)

`deepreg.model.layer_util.warp_grid(grid: tensorflow.Tensor, theta: tensorflow.Tensor) → tensorflow.Tensor`

Perform transformation on the grid.

- grid_padded[i,j,k,:] = [i j k 1]
- grid_warped[b,i,j,k,p] = sum_over_q (grid_padded[i,j,k,q] * 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)

3.24 Loss

3.24.1 Image Loss

Provide different loss or metrics classes for images.

class deepreg.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, 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
- **name** – name of the loss
- **kwargs** – additional arguments.

call(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*) → 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() → dict

Return the config dictionary for recreating this class.

class deepreg.loss.image.**GlobalMutualInformationLoss**(**args: Any*, ***kwargs: Any*)

Revert the sign of GlobalMutualInformation.

Init without required arguments.

Parameters **kwargs** – additional arguments.

class deepreg.loss.image.**GlobalNormalizedCrossCorrelation**(**args: Any*, ***kwargs: Any*)

Global squared zero-normalized cross-correlation.

Compute the squared cross-correlation between the reference and moving images y_true and y_pred have to be at least 4d tensor, including batch axis.

Reference:

- **Zero-normalized cross-correlation (ZNCC):** https://en.wikipedia.org/wiki/Cross-correlation

Init.

Parameters

- **name** – name of the loss
- **kwargs** – additional arguments.

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,)

class deepreg.loss.image.**GlobalNormalizedCrossCorrelationLoss**(**args: Any*, ***kwargs: Any*)

Revert the sign of GlobalNormalizedCrossCorrelation.

Init without required arguments.

Parameters **kwargs** – additional arguments.

class deepreg.loss.image.**LocalNormalizedCrossCorrelation**(**args: Any*, ***kwargs: Any*)

Local squared zero-normalized cross-correlation.

Denote y_true as t and y_pred as p. Consider a window having n elements. Each position in the window corresponds a weight w_i for i=1:n.

Define the discrete expectation in the window E[t] as

E[t] = sum_i(w_i * t_i) / sum_i(w_i)

Similarly, the discrete variance in the window V[t] is

V[t] = E[t**2] - E[t] ** 2

The local squared zero-normalized cross-correlation is therefore

E[(t-E[t]) * (p-E[p])] ** 2 / V[t] / V[p]

where the expectation in numerator is

E[(t-E[t]) * (p-E[p])] = E[t * p] - E[t] * E[p]

Different kernel corresponds to different weights.

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

Reference:

- **Zero-normalized cross-correlation (ZNCC):** https://en.wikipedia.org/wiki/Cross-correlation
- Code: 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
- **smooth_nr** – small constant added to numerator in case of zero covariance.
- **smooth_dr** – small constant added to denominator in case of zero variance.
- **name** – name of the loss.
- **kwargs** – additional arguments.

calc_ncc(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*) → tensorflow.Tensor

Return NCC for a batch.

The kernel should not be normalized, as normalizing them leads to computation with small values and the precision will be reduced. Here both numerator and denominator are actually multiplied by kernel volume, which helps the precision as well. However, when the variance is zero, the obtained value might be negative due to machine error. Therefore a hard-coded clipping is added to prevent division by zero.

Parameters

- **y_true** – shape = (batch, dim1, dim2, dim3, 1)
- **y_pred** – shape = (batch, dim1, dim2, dim3, 1)

Returns shape = (batch, dim1, dim2, dim3. 1)

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

TODO: support channel axis dimension > 1.

Parameters

- **y_true** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, 1)
- **y_pred** – shape = (batch, dim1, dim2, dim3) or (batch, dim1, dim2, dim3, 1)

Returns shape = (batch,)

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

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

Init without required arguments.

Parameters **kwargs** – additional arguments.

3.24.2 Label Loss

Provide different loss or metrics classes for labels.

class deepreg.loss.label.**CrossEntropy**(**args: Any*, ***kwargs: Any*)
Define weighted cross-entropy.

The formulation is: loss = w_fg * y_true log(y_pred) - w_bg * (1y_true) log(1y_pred)

Init.

Parameters

- **binary** – if True, project y_true, y_pred to 0 or 1
- **background_weight** – weight for background, where y == 0.
- **smooth** – smooth constant for log.
- **name** – name of the loss.
- **kwargs** – additional arguments.

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,)

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

class deepreg.loss.label.**CrossEntropyLoss**(**args: Any*, ***kwargs: Any*)
Define loss with multi-scaling options.

Init.

Parameters

- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
- **kwargs** – additional arguments.

class deepreg.loss.label.**DiceLoss**(**args: Any*, ***kwargs: Any*)
Revert the sign of DiceScore and support multi-scaling options.

Init without required arguments.

Parameters **kwargs** – additional arguments.

class deepreg.loss.label.**DiceScore**(**args: Any*, ***kwargs: Any*)
Define dice score.

The formulation is:

0. w_fg + w_bg = 1
1. let y_prod = y_true * y_pred and y_sum = y_true + y_pred
2. **num = 2 * (w_fg * y_true * y_pred + w_bg * (1y_true) * (1y_pred))** = 2 * ((w_fg+w_bg) * y_prod - w_bg * y_sum + w_bg) = 2 * (y_prod - w_bg * y_sum + w_bg)
3. **denom = (w_fg * (y_true + y_pred) + w_bg * (1y_true + 1y_pred))** = (w_fg-w_bg) * y_sum + 2 * w_bg = (1-2*w_bg) * y_sum + 2 * w_bg
4. dice score = num / denom

where num and denom are summed over all axes except the batch axis.

Reference: Sudre, Carole H., et al. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations." Deep learning in medical image analysis and multimodal learning for clinical decision support. Springer, Cham, 2017. 240-248.

Init.

Parameters

- **binary** – if True, project y_true, y_pred to 0 or 1.
- **background_weight** – weight for background, where y == 0.
- **smooth_nr** – small constant added to numerator in case of zero covariance.
- **smooth_dr** – small constant added to denominator in case of zero variance.
- **name** – name of the loss.
- **kwargs** – additional arguments.

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,)

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

class deepreg.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

0. w_fg + w_bg = 1
1. let y_prod = y_true * y_pred and y_sum = y_true + y_pred
2. **num = (w_fg * y_true * y_pred + w_bg * (1y_true) * (1y_pred))** = ((w_fg+w_bg) * y_prod - w_bg * y_sum + w_bg) = (y_prod - w_bg * y_sum + w_bg)
3. **denom = (w_fg * (y_true + y_pred - y_true * y_pred)**
   - w_bg * (1y_true + 1y_pred - (1y_true) * (1y_pred)))

   = w_fg * (y_sum - y_prod) + w_bg * (1-y_prod) = (1-w_bg) * y_sum - y_prod + w_bg
4. dice score = num / denom

Init.

Parameters

- **binary** – if True, project y_true, y_pred to 0 or 1.
- **background_weight** – weight for background, where y == 0.
- **smooth_nr** – small constant added to numerator in case of zero covariance.
- **smooth_dr** – small constant added to denominator in case of zero variance.
- **name** – name of the loss.
- **kwargs** – additional arguments.

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,)

class deepreg.loss.label.**JaccardLoss**(**args: Any*, ***kwargs: Any*)
Revert the sign of JaccardIndex.

Init without required arguments.

Parameters **kwargs** – additional arguments.

class deepreg.loss.label.**SumSquaredDifference**(**args: Any*, ***kwargs: Any*)
Actually, mean of squared distance between y_true and y_pred.

The inconsistent name was for convention.

y_true and y_pred have to be at least 1d tensor, including batch axis.

Init.

Parameters

- **name** – name of the loss.
- **kwargs** – additional arguments.

call(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*) → tensorflow.Tensor
Return mean squared different for a batch.

Parameters

- **y_true** – shape = (batch, . . .)
- **y_pred** – shape = (batch, . . .)

Returns shape = (batch,)

class deepreg.loss.label.**SumSquaredDifferenceLoss**(**args: Any*, ***kwargs: Any*)
Define loss with multi-scaling options.

Init.

Parameters

- **scales** – list of scalars or None, if None, do not apply any scaling.
- **kernel** – gaussian or cauchy.
- **kwargs** – additional arguments.

deepreg.loss.label.**compute_centroid**(*mask: tensorflow.Tensor*, *grid: tensorflow.Tensor*) → tensorflow.Tensor
Calculate the centroid of the mask. :param mask: shape = (batch, dim1, dim2, dim3) :param grid: shape = (1, dim1, dim2, dim3, 3) :return: shape = (batch, 3), batch of vectors denoting

location of centroids.

deepreg.loss.label.**compute_centroid_distance**(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*, *grid: tensorflow.Tensor*) → tensorflow.Tensor
Calculate the L2-distance between two tensors' centroids. :param y_true: tensor, shape = (batch, dim1, dim2, dim3) :param y_pred: tensor, shape = (batch, dim1, dim2, dim3) :param grid: tensor, shape = (1, dim1, dim2, dim3, 3) :return: shape = (batch,)

deepreg.loss.label.**foreground_proportion**(*y: tensorflow.Tensor*) → tensorflow.Tensor
Calculate the percentage of foreground vs background per 3d volume. :param y: shape = (batch, dim1, dim2, dim3), a 3D label tensor :return: shape = (batch,)

3.24.3 Deformation Loss

Provide regularization functions and classes for ddf.

class deepreg.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.

- **kwargs** – additional arguments.

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 = (batch,)

class deepreg.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
- **kwargs** – additional arguments.

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 = (batch,)

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

deepreg.loss.deform.**gradient_dx** (*fx: tensorflow.Tensor*) → tensorflow.Tensor
Calculate gradients on x-axis of a 3D tensor using central finite difference.

It moves the tensor along axis 1 to calculate the approximate gradient, the x axis, dx[i] = (x[i+1] - x[i-1]) / 2.

Parameters **fx** – shape = (batch, m_dim1, m_dim2, m_dim3)

Returns shape = (batch, m_dim1-2, m_dim2-2, m_dim3-2)

deepreg.loss.deform.**gradient_dxyz** (*fxyz: tensorflow.Tensor*, *fn: Callable*) → tensorflow.Tensor
Calculate gradients on x,y,z-axis of a tensor using central finite difference.

The gradients are calculated along x, y, z separately then stacked together.

Parameters

- **fxyz** – shape = (..., 3)
- **fn** – function to call

Returns shape = (..., 3)

deepreg.loss.deform.**gradient_dy** (*fy: tensorflow.Tensor*) → tensorflow.Tensor
Calculate gradients on y-axis of a 3D tensor using central finite difference.

It moves the tensor along axis 2 to calculate the approximate gradient, the y axis, dy[i] = (y[i+1] - y[i-1]) / 2.

Parameters **fy** – shape = (batch, m_dim1, m_dim2, m_dim3)

Returns shape = (batch, m_dim1-2, m_dim2-2, m_dim3-2)

deepreg.loss.deform.**gradient_dz**(*fz: tensorflow.Tensor*) → tensorflow.Tensor

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

It moves the tensor along axis 3 to calculate the approximate gradient, the z axis, dz[i] = (z[i+1] - z[i-1]) / 2.

Parameters **fz** – shape = (batch, m_dim1, m_dim2, m_dim3)

Returns shape = (batch, m_dim1-2, m_dim2-2, m_dim3-2)

3.24.4 Loss Util

Provide helper functions or classes for defining loss or metrics.

class deepreg.loss.util.**MultiScaleMixin**(**args: Any*, ***kwargs: Any*)

Mixin 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.
- **kwargs** – additional arguments.

call(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*) → tensorflow.Tensor

Use super().call to calculate loss at different scales.

Parameters

- **y_true** – ground-truth tensor, shape = (batch, dim1, dim2, dim3).
- **y_pred** – predicted tensor, shape = (batch, dim1, dim2, dim3).

Returns multi-scale loss, shape = (batch,).

get_config() → dict

Return the config dictionary for recreating this class.

class deepreg.loss.util.**NegativeLossMixin**(**args: Any*, ***kwargs: Any*)

Mixin class to revert the sign of the loss value.

Init without required arguments.

Parameters **kwargs** – additional arguments.

call(*y_true: tensorflow.Tensor*, *y_pred: tensorflow.Tensor*) → tensorflow.Tensor

Revert the sign of loss.

Parameters

- **y_true** – ground-truth tensor.
- **y_pred** – predicted tensor.

Returns negated loss.

`deepreg.loss.util.`**`separable_filter`**(*tensor: tensorflow.Tensor*, *kernel: tensorflow.Tensor*) → tensorflow.Tensor

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, 1)
- **`kernel`** – shape = (dim4,)

Returns shape = (batch, dim1, dim2, dim3, 1)

3.25 Optimizer

Functions parsing the config optimizer options

`deepreg.model.optimizer.`**`build_optimizer`**(*optimizer_config: dict*) → tensorflow.python.keras.optimizer_v2.optimizer_v2.OptimizerV2

Parsing the optimiser options and parameters from config dictionary.

Parameters **`optimizer_config`** – has key name and other required arguments

Returns optimizer instant

3.26 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.

3.26.1 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.

3.26.2 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 is as working in progress.

3.26.3 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.

3.27 Unit Test

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

For testing the code locally,

- If you only need to test with python 3.7, please execute `pytest` at the repository root:

```
pytest test/
```

- If you want to test with all supported python versions, please execute `tox` at the repository root:

```
tox
```

Moreover, the testing is checked automatically 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.

3.27.1 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.

3.27.2 Example unit test

We provide here an example to help understanding the requirements.

```
import pytest


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"
    return x - 1


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)
```

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.

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.

3.28 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.

3.28.1 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.

3.28.2 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;

3.28.3 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`.

3.28.4 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`;

3.28.5 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.

3.29 Documentation

We use Sphinx to organize the documentation and it is hosted in ReadTheDocs.

3.29.1 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.

3.29.2 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.readthedocs.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.

3.30 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.

3.30.1 Creating a new Release

From the main DeepReg repository page, head to releases. From here, you can draft a new release.

3.30.2 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.

3.30.3 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.

PYTHON MODULE INDEX

d

INDEX

Symbols

B

C

D

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W