

---

# DeepReg

## DeepReg

Nov 29, 2020



# GETTING STARTED

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Contributors</b>	<b>5</b>
<b>3</b>	<b>Contact</b>	<b>7</b>
3.1	Installation . . . . .	7
3.2	Quick Start . . . . .	10
3.3	Image Registration with Deep Learning . . . . .	11
3.4	Design Experiments . . . . .	13
3.5	Custom Functionalities . . . . .	14
3.6	Introduction to DeepReg Demos . . . . .	24
3.7	Paired Images . . . . .	25
3.8	Unpaired Images . . . . .	30
3.9	Grouped Images . . . . .	37
3.10	Classical Registration . . . . .	40
3.11	Command Line Tools . . . . .	43
3.12	Configuration File . . . . .	49
3.13	Dataset Loader . . . . .	49
3.14	Experimental Features . . . . .	59
3.15	Entry Point . . . . .	60
3.16	Dataset Loader . . . . .	62
3.17	File Loader . . . . .	66
3.18	Network . . . . .	69
3.19	Network Backbone . . . . .	73
3.20	Layer . . . . .	76
3.21	Loss . . . . .	88
3.22	Optimizer . . . . .	94
3.23	Guidelines . . . . .	94
3.24	Unit Test . . . . .	95
3.25	DeepReg demo . . . . .	96
3.26	Documentation . . . . .	98
3.27	Release . . . . .	98
	<b>Python Module Index</b>	<b>101</b>
	<b>Index</b>	<b>103</b>



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!



## **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.





## CONTRIBUTORS

DeepReg is maintained and led by a team of developers and researchers. People with significant contributions to DeepReg are listed below (in alphabetical order).

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

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](#)).



## 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](mailto: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](mailto:nina.brown.15@ucl.ac.uk)), Ester Bonmati ([e.bonmati@ucl.ac.uk](mailto:e.bonmati@ucl.ac.uk)), Matt Clarkson ([m.clarkson@ucl.ac.uk](mailto:m.clarkson@ucl.ac.uk)).



## 3.1 Installation

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

There are multiple different methods to install DeepReg:

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

### 3.1.1 Install via Conda

The recommended method is to install DeepReg in a dedicated virtual environment using [Anaconda](#) / [Miniconda](#).

Please clone [DeepReg](#) first and change current directory to the DeepReg root directory:

```
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

## DeepReg

---

### Windows

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
```

Update DeepReg without GPU support.

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

Update DeepReg with GPU support.

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

Install DeepReg without GPU support.

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

Update DeepReg without GPU support.

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

Install/update DeepReg with GPU support.

**Warning:** Not supported or tested.

Install/update DeepReg without GPU support.

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

Install/update DeepReg with GPU support.

**Warning:** Not supported or tested.

After activating the conda environment, please install DeepReg locally:

```
pip install -e .
```

### 3.1.2 Install via docker

We also provide the docker file for building the docker image. Please clone [DeepReg](https://github.com/DeepRegNet/DeepReg.git) 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
```

#### Note

1. All dependencies, APIs and command-line tools will be installed automatically via each installation method. However, the PyPI release currently does not ship with test data and demos. Running examples in this documentation may require downloading test data and changing default paths to user-installed packages with the PyPI release. These examples include those in the [Quick Start](#) and [DeepReg Demo](#).

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:

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

where:

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

### 3.2.2 Evaluate a trained network

Once trained, evaluate the network using a test dataset:

```
deeppreg_predict --gpu "" --ckpt_path logs/test/save/weights-epoch2.ckpt --mode test
```

where:

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

### 3.2.3 Warp an image

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

```
deeppreg_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”:

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

“cv\_run2.yaml”:

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

“cv\_run3.yaml”:

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

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

“test.yaml”:

```
dataset:
  dir:
    train: ""
    valid: ""
    test: "data/test/h5/paired/test" # validation data set
```

## 3.5 Custom Functionalities

Besides the implemented features provided in DeepReg, we provide the following tutorials for implementing custom functionalities.

- Custom loss function
- Custom network

### 3.5.1 Custom loss function

This tutorial will take an image intensity based loss function (mutual information) as an example to show how to add a new loss to DeepReg.

#### A brief review of the types of loss functions in DeepReg

Three main types of the loss functions are supported in DeepReg: intensity (image) based loss, label based loss and deformation loss. See [Docs here](#) for details. The corresponding source files for the losses is included in `deepreg/model/loss`.

#### Step 1: Add the new function in loss source code

The first step is to add your own loss function, which should take at least 2 parameters, `y_true` for the ground truth and `y_pred` for the prediction. e.g. in `deepreg/model/loss/image.py`. The loss can be defined as:

```
def global_mutual_information(y_true: tf.Tensor, y_pred: tf.Tensor) -> tf.Tensor:
    """
    differentiable global mutual information loss via Parzen windowing method.
    reference: https://dspace.mit.edu/handle/1721.1/123142, Section 3.1, equation 3.1-
    ↪ 3.5
    :y_true: shape = (batch, dim1, dim2, dim3, ch)
```

(continues on next page)

(continued from previous page)

```

:y_pred: shape = (batch, dim1, dim2, dim3, ch)
:return: shape = (batch,)
"""
...
return tf.reduce_sum(pab * tf.math.log(pab / papb + eps), axis=[1, 2])

```

In order to be compatible with the pipeline in DeepReg, another modification is needed in `deeepreg/model/loss/image.py`. The modification is simply adding an `elif` branch to the `dissimilarity_fn` function. The following code block shows the added `elif` branch where we use "gmi" to represent the global mutual information:

```

def dissimilarity_fn(
    y_true: tf.Tensor, y_pred: tf.Tensor, name: str, **kwargs
) -> tf.Tensor:
    """
    :param y_true: fixed_image, shape = (batch, f_dim1, f_dim2, f_dim3)
    :param y_pred: warped_moving_image, shape = (batch, f_dim1, f_dim2, f_dim3)
    :param name: name of the dissimilarity function
    :param kwargs: absorb additional parameters
    :return: shape = (batch,)
    """
    assert name in ["lncc", "ssd", "gmi"]
    # shape = (batch, f_dim1, f_dim2, f_dim3, 1)
    y_true = tf.expand_dims(y_true, axis=4)
    y_pred = tf.expand_dims(y_pred, axis=4)
    if name == "lncc":
        return -local_normalized_cross_correlation(y_true, y_pred, **kwargs)
    elif name == "ssd":
        return ssd(y_true, y_pred)
    elif name == "gmi":
        return -global_mutual_information(y_true, y_pred)
    else:
        raise ValueError("Unknown loss type.")

```

## Step 2: Add test functions (for contributing developers, optional for users)

Add corresponding unit test for the new added functions to `deeepreg/test/unit`. This is optional for the users. Everyone is warmly welcome to make contribution to DeepReg. Please follow our [contribution guidelines](#) here.

## Step 3: Set yaml configuration files

We take the [paired prostate MR and Ultrasound registration demo](#) as an example. In order to use the newly added loss, all that is needed is to modify the loss configuration in the train configuration file `paired_mrus_prostate_train.yaml` (lines 10-15):

```

# define the loss function for training
loss:
  dissimilarity:
    image:
      name: "gmi"
      weight: 1.0

```

That's it. Follow the instructions in the demo to begin training with the newly added loss.

### 3.5.2 Custom network

This tutorial shows how to define a new network and add it to DeepReg, using a specific example for adding a GlobalNet to predict an affine transformation, as opposed to nonrigid transformation.

For general guidance on making a contribution to DeepReg, see the [contribution guidelines](#).

#### Step 1: Create network backbone

The first step is to create a new backbone class, which consists of the neural network architecture you want to use, and place it in the backbone directory `deepreg/model/backbone/`. The affine method uses the GlobalNet network architecture (`deepreg/model/backbone/global_net.py`) from [Hu et al. 2018](#). The GlobalNet network has an encoder-only architecture, which is used to predict the parameters of an affine transformation model, with 12 degrees of freedom.

We recommend using the [tf.keras API](#) to write your network, by defining the layers of your backbone class in `def __init__()` and the network's forward pass in `def call()`. Custom DeepReg layers can be found in `deepreg/model/layer.py`.

```
class GlobalNet(tf.keras.Model):
    """
    Builds GlobalNet for image registration based on
    Y. Hu et al.,
    "Label-driven weakly-supervised learning for multimodal
    deformable image registration,"
    (ISBI 2018), pp. 1070-1074.
    """

    def __init__(
        self,
        image_size,
        out_channels,
        num_channel_initial,
        extract_levels,
        out_kernel_initializer,
        out_activation,
        **kwargs,
    ):
        """
        Image is encoded gradually, i from level 0 to E.
        Then, a densely-connected layer outputs an affine
        transformation.
        :param out_channels: int, number of channels for the output
        :param num_channel_initial: int, number of initial channels
        :param extract_levels: list, which levels from net to extract
        :param out_activation: str, activation at last layer
        :param out_kernel_initializer: str, which kernel to use as initialiser
        :param kwargs:
        """
        super(GlobalNet, self).__init__(**kwargs)
        # save parameters
        self._extract_levels = extract_levels
        self._extract_max_level = max(self._extract_levels) # E
        self.reference_grid = layer_util.get_reference_grid(image_size)
        self.transform_initial = tf.constant_initializer(
            value=[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
        )
```

(continues on next page)

(continued from previous page)

```

# init layer variables
num_channels = [
    num_channel_initial * (2 ** level)
    for level in range(self._extract_max_level + 1)
] # level 0 to E
self._downsample_blocks = [
    layer.DownSampleResnetBlock(
        filters=num_channels[i], kernel_size=7 if i == 0 else 3
    )
    for i in range(self._extract_max_level)
] # level 0 to E-1
self._conv3d_block = layer.Conv3dBlock(filters=num_channels[-1]) # level E
self._dense_layer = layer.Dense(
    units=12, bias_initializer=self.transform_initial
)

def call(self, inputs, training=None, mask=None):
    """
    Build GlobalNet graph based on built layers.
    :param inputs: image batch, shape = [batch, f_dim1, f_dim2, f_dim3, ch]
    :param training:
    :param mask:
    :return:
    """
    # down sample from level 0 to E
    h_in = inputs
    for level in range(self._extract_max_level): # level 0 to E - 1
        h_in, _ = self._downsample_blocks[level](inputs=h_in, training=training)
    h_out = self._conv3d_block(
        inputs=h_in, training=training
    ) # level E of encoding
    # predict affine parameters theta of shape = [batch, 4, 3]
    self.theta = self._dense_layer(h_out)
    self.theta = tf.reshape(self.theta, shape=(-1, 4, 3))
    # warp the reference grid with affine parameters to output a ddf
    grid_warped = layer_util.warp_grid(self.reference_grid, self.theta)
    output = grid_warped - self.reference_grid
    return output`

```

In order to use the backbone network in the DeepReg pipeline, a new option needs to be added to `build_backbone()` from `deepreg/model/network/util.py`. We use the keyword “global” here to refer to our GlobalNet class and “affine” for the method name. This will allow us to define the backbone network directly in the configuration file.

```

def build_backbone(
    image_size: tuple, out_channels: int, model_config: dict, method_name: str
) -> tf.keras.Model:
    """
    Backbone model accepts a single input of shape (batch, dim1, dim2, dim3, ch_in)
    and returns a single output of shape (batch, dim1, dim2, dim3, ch_out)
    :param image_size: tuple, dims of image, (dim1, dim2, dim3)
    :param out_channels: int, number of out channels, ch_out
    :param method_name: str, one of ddf | dvf | conditional
    :param model_config: dict, model configuration, returned from parser.yaml.load
    :return: tf.keras.Model
    """

```

(continues on next page)

```

if not (
    (isinstance(image_size, tuple) or isinstance(image_size, list))
    and len(image_size) == 3
):
    raise ValueError(f"image_size must be tuple of length 3, got {image_size}")
if not (isinstance(out_channels, int) and out_channels >= 1):
    raise ValueError(f"out_channels must be int >=1, got {out_channels}")
if not (isinstance(model_config, dict) and "backbone" in model_config.keys()):
    raise ValueError(
        f"model_config must be a dict having key 'backbone', got {model_config}"
    )
if method_name not in ["ddf", "dvf", "conditional", "affine"]:
    raise ValueError(
        "method name has to be one of ddf/dvf/conditional/affine in build_
↪backbone, "
        "got {}".format(method_name)
    )

if method_name in ["ddf", "dvf"]:
    out_activation = None
    # TODO try random init with smaller number
    out_kernel_initializer = "zeros" # to ensure small ddf and dvf
elif method_name in ["conditional"]:
    out_activation = "sigmoid" # output is probability
    out_kernel_initializer = "glorot_uniform"
elif method_name in ["affine"]:
    out_activation = None
    out_kernel_initializer = "zeros"
else:
    raise ValueError("Unknown method name {}".format(method_name))

if model_config["backbone"] == "local":
    return LocalNet(
        image_size=image_size,
        out_channels=out_channels,
        out_kernel_initializer=out_kernel_initializer,
        out_activation=out_activation,
        **model_config["local"],
    )
elif model_config["backbone"] == "global":
    return GlobalNet(
        image_size=image_size,
        out_channels=out_channels,
        out_kernel_initializer=out_kernel_initializer,
        out_activation=out_activation,
        **model_config["global"],
    )
elif model_config["backbone"] == "unet":
    return UNet(
        image_size=image_size,
        out_channels=out_channels,
        out_kernel_initializer=out_kernel_initializer,
        out_activation=out_activation,
        **model_config["unet"],
    )
else:
    raise ValueError("Unknown model name")

```

## Step 2: Create network model

We can now create a network model for the affine method in `deepreg/model/network/affine.py`. We first need to write the model's forward pass, which makes use of the backbone network class to predict an affine transformation which will be used to output a dense displacement field (DDF).

```
def affine_forward(
    backbone: tf.keras.Model,
    moving_image: tf.Tensor,
    fixed_image: tf.Tensor,
    moving_label: (tf.Tensor, None),
    moving_image_size: tuple,
    fixed_image_size: tuple,
):
    """
    Perform the network forward pass
    :param backbone: model architecture object, e.g. model.backbone.local_net
    :param moving_image: tensor of shape (batch, m_dim1, m_dim2, m_dim3)
    :param fixed_image: tensor of shape (batch, f_dim1, f_dim2, f_dim3)
    :param moving_label: tensor of shape (batch, m_dim1, m_dim2, m_dim3) or None
    :param moving_image_size: tuple like (m_dim1, m_dim2, m_dim3)
    :param fixed_image_size: tuple like (f_dim1, f_dim2, f_dim3)
    :return: tuple(_affine, _ddf, _pred_fixed_image, _pred_fixed_label)
    :return: tuple(affine, ddf, pred_fixed_image, pred_fixed_label, fixed_grid), where
        - affine is the affine transformation matrix predicted by the network (batch, 4, 3)
        - ddf is the dense displacement field of shape (batch, f_dim1, f_dim2, f_dim3, 3)
        - pred_fixed_image is the predicted (warped) moving image of shape (batch, f_dim1,
        f_dim2, f_dim3)
        - pred_fixed_label is the predicted (warped) moving label of shape (batch, f_dim1,
        f_dim2, f_dim3)
        - fixed_grid is the grid of shape (f_dim1, f_dim2, f_dim3, 3)
    """

    # expand dims
    # need to be squeezed later for warping
    moving_image = tf.expand_dims(
        moving_image, axis=4
    ) # (batch, m_dim1, m_dim2, m_dim3, 1)
    fixed_image = tf.expand_dims(
        fixed_image, axis=4
    ) # (batch, f_dim1, f_dim2, f_dim3, 1)

    # adjust moving image
    moving_image = layer_util.resize3d(
        image=moving_image, size=fixed_image_size
    ) # (batch, f_dim1, f_dim2, f_dim3, 1)

    # ddf, dvf
    inputs = tf.concat(
        [moving_image, fixed_image], axis=4
    ) # (batch, f_dim1, f_dim2, f_dim3, 2)
    ddf = backbone(inputs=inputs) # (batch, f_dim1, f_dim2, f_dim3, 3)
    affine = backbone.theta

    # prediction, (batch, f_dim1, f_dim2, f_dim3)
    warping = layer.Warping(fixed_image_size=fixed_image_size)
    grid_fixed = tf.squeeze(warping.grid_ref, axis=0) # (f_dim1, f_dim2, f_dim3, 3)
```

(continues on next page)

(continued from previous page)

```

pred_fixed_image = warping(inputs=[ddf, tf.squeeze(moving_image, axis=4)])
pred_fixed_label = (
    warping(inputs=[ddf, moving_label]) if moving_label is not None else None
)
return affine, ddf, pred_fixed_image, pred_fixed_label, grid_fixed

```

Similar to `build_backbone` we also need to write the `build_affine_model` function, which consists of building the model according to the networks' inputs, backbone and loss function.

```

def build_affine_model(
    moving_image_size: tuple,
    fixed_image_size: tuple,
    index_size: int,
    labeled: bool,
    batch_size: int,
    model_config: dict,
    loss_config: dict,
):
    """
    :param moving_image_size: (m_dim1, m_dim2, m_dim3)
    :param fixed_image_size: (f_dim1, f_dim2, f_dim3)
    :param index_size: int, the number of indices for identifying a sample
    :param labeled: bool, indicating if the data is labeled
    :param batch_size: int, size of mini-batch
    :param model_config: config for the model
    :param loss_config: config for the loss
    :return: the built tf.keras.Model
    """

    # inputs
    (moving_image, fixed_image, moving_label, fixed_label, indices) = build_inputs(
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
        index_size=index_size,
        batch_size=batch_size,
        labeled=labeled,
    )

    # backbone
    backbone = build_backbone(
        image_size=fixed_image_size,
        out_channels=3,
        model_config=model_config,
        method_name=model_config["method"],
    )

    # forward
    affine, ddf, pred_fixed_image, pred_fixed_label, grid_fixed = affine_forward(
        backbone=backbone,
        moving_image=moving_image,
        fixed_image=fixed_image,
        moving_label=moving_label,
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
    )

```

(continues on next page)



(continued from previous page)

```

# build model
inputs = {
    "moving_image": moving_image,
    "fixed_image": fixed_image,
    "indices": indices,
}
outputs = {"ddf": ddf, "affine": affine}
model_name = model_config["method"].upper() + "RegistrationModel"
if moving_label is None: # unlabeled
    model = tf.keras.Model(
        inputs=inputs, outputs=outputs, name=model_name + "WithoutLabel"
    )
else: # labeled
    inputs["moving_label"] = moving_label
    inputs["fixed_label"] = fixed_label
    outputs["pred_fixed_label"] = pred_fixed_label
    model = tf.keras.Model(
        inputs=inputs, outputs=outputs, name=model_name + "WithLabel"
    )

# add loss and metric
model = add_ddf_loss(model=model, ddf=ddf, loss_config=loss_config)
model = add_image_loss(
    model=model,
    fixed_image=fixed_image,
    pred_fixed_image=pred_fixed_image,
    loss_config=loss_config,
)
model = add_label_loss(
    model=model,
    grid_fixed=grid_fixed,
    fixed_label=fixed_label,
    pred_fixed_label=pred_fixed_label,
    loss_config=loss_config,
)

return model

```

Finally, the last step consists of adding the `build_affine_model` option to `deepreg/model/network/build.py` to be able to parse it from the configuration file.

```

def build_model(
    moving_image_size: tuple,
    fixed_image_size: tuple,
    index_size: int,
    labeled: bool,
    batch_size: int,
    model_config: dict,
    loss_config: dict,
):
    """
    Parsing algorithm types to model building functions
    :param moving_image_size: [m_dim1, m_dim2, m_dim3]
    :param fixed_image_size: [f_dim1, f_dim2, f_dim3]
    :param index_size: dataset size
    :param labeled: true if the label of moving/fixed images are provided
    """

```

(continues on next page)

(continued from previous page)

```

:param batch_size: mini-batch size
:param model_config: model configuration, e.g. dictionary return from parser.yaml.
↪load
:param loss_config: loss configuration, e.g. dictionary return from parser.yaml.
↪load
:return: the built tf.keras.Model
"""
if model_config["method"] in ["ddf", "dvf"]:
    return build_ddf_dvf_model(
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
        index_size=index_size,
        labeled=labeled,
        batch_size=batch_size,
        model_config=model_config,
        loss_config=loss_config,
    )
elif model_config["method"] == "conditional":
    return build_conditional_model(
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
        index_size=index_size,
        labeled=labeled,
        batch_size=batch_size,
        model_config=model_config,
        loss_config=loss_config,
    )
elif model_config["method"] == "affine":
    return build_affine_model(
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
        index_size=index_size,
        labeled=labeled,
        batch_size=batch_size,
        model_config=model_config,
        loss_config=loss_config,
    )
else:
    raise ValueError("Unknown model method")

```

### Step 3: Testing (for contributing developers, optional for users)

Everyone is warmly welcome to make contributions to DeepReg and add corresponding unit test for the newly added functions to `test/unit/`. Recommendations regarding testing style can be found in the [contribution guidelines](#). Here is a practical example of unit tests made for our affine model in `test/unit/test_affine.py`:

```

def test_affine_forward():
    """
    Testing that affine_forward function returns the tensors with correct shapes
    """

    moving_image_size = (1, 3, 5)
    fixed_image_size = (2, 4, 6)
    batch_size = 1

```

(continues on next page)

(continued from previous page)

```

global_net = build_backbone(
    image_size=fixed_image_size,
    out_channels=3,
    model_config={
        "backbone": "global",
        "global": {"num_channel_initial": 4, "extract_levels": [1, 2, 3]},
    },
    method_name="affine",
)

# Check conditional mode network output shapes - Pass
affine, ddf, pred_fixed_image, pred_fixed_label, grid_fixed = affine_forward(
    backbone=global_net,
    moving_image=tf.ones((batch_size,) + moving_image_size),
    fixed_image=tf.ones((batch_size,) + fixed_image_size),
    moving_label=tf.ones((batch_size,) + moving_image_size),
    moving_image_size=moving_image_size,
    fixed_image_size=fixed_image_size,
)
assert affine.shape == (batch_size,) + (4,) + (3,)
assert ddf.shape == (batch_size,) + fixed_image_size + (3,)
assert pred_fixed_image.shape == (batch_size,) + fixed_image_size
assert pred_fixed_label.shape == (batch_size,) + fixed_image_size
assert grid_fixed.shape == fixed_image_size + (3,)

def test_build_affine_model():
    """
    Testing that build_affine_model function returns the tensors with correct shapes
    """
    moving_image_size = (1, 3, 5)
    fixed_image_size = (2, 4, 6)
    batch_size = 1

    model = build_affine_model(
        moving_image_size=moving_image_size,
        fixed_image_size=fixed_image_size,
        index_size=1,
        labeled=True,
        batch_size=batch_size,
        model_config={
            "method": "affine",
            "backbone": "global",
            "global": {"num_channel_initial": 4, "extract_levels": [1, 2, 3]},
        },
        loss_config={
            "dissimilarity": {
                "image": {"name": "lncc", "weight": 0.1},
                "label": {
                    "name": "multi_scale",
                    "weight": 1,
                    "multi_scale": {
                        "loss_type": "dice",
                        "loss_scales": [0, 1, 2, 4, 8, 16, 32],
                    },
                },
            },
        },
    )

```

(continues on next page)

(continued from previous page)

```

        "regularization": {"weight": 0.0, "energy_type": "bending"},
    },
)

inputs = {
    "moving_image": tf.ones((batch_size,) + moving_image_size),
    "fixed_image": tf.ones((batch_size,) + fixed_image_size),
    "indices": 1,
    "moving_label": tf.ones((batch_size,) + moving_image_size),
    "fixed_label": tf.ones((batch_size,) + fixed_image_size),
}

outputs = model(inputs)

expected_outputs_keys = ["affine", "ddf", "pred_fixed_label"]
assert all(keys in expected_outputs_keys for keys in outputs)
assert outputs["pred_fixed_label"].shape == (batch_size,) + fixed_image_size
assert outputs["affine"].shape == (batch_size,) + (4,) + (3,)
assert outputs["ddf"].shape == (batch_size,) + fixed_image_size + (3,)

```

#### Step 4: Set yaml configuration files

An example of yaml configuration file for the affine method is available in `config/unpaired_labeled_affine.yaml`. For using both the GlobalNet backbone and affine method you will need to add their aforementioned keyword “global” and “affine”. Optional parameters such as `out_kernel_initializer` or `num_channel_initial` can also be specified. A snippet of `config/unpaired_labeled_affine.yaml` is shown below. Please see the [configuration documentation](#) for more details.

```

model:
method: "affine"
backbone:
  name: "global"
  out_kernel_initializer: "zeros"
  out_activation: ""
global:
  num_channel_initial: 1
  extract_levels: [0, 1, 2, 3, 4]

```

## 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 `--no-test` to use the original training configuration, such as

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

#### Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/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.

## 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](#).

[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 `--no-test` to use the original training configuration, such as

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

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

### Predict

Please execute the following command to run the prediction with pre-trained model. The prediction logs and visualization results will be saved under `demos/paired_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.

### 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](#).

#### 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 `--no-test` to use the original training configuration, such as

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

### Predict

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

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

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

### 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](#).

[Source Code](#)

### Author

DeepReg Development Team (Ester Bonmati)

## Application

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

## Data

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

## Instruction

### Installation

Please install DeepReg following the [instructions](#) and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

### Download data

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

```
python demos/unpaired_ct_abdomen/demo_data.py
```

### Launch demo training

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

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

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

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

### Predict

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

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

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

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

### Data

The dataset for this demo comes from [1] and can be downloaded from: <https://zenodo.org/record/3835682#.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 `--no-test` to use the original training configuration, such as

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

## Predict

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

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

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

## 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](#).

[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 `--no-test` to use the original training configuration, such as

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

## Predict

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

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

## 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 `--no-test` to use the original training configuration, such as

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

#### Predict

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

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

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



## 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.

## Author

DeepReg Development Team

## Application

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

## Data

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

### Instruction

### Installation

Please install DeepReg following the [instructions](#) and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

### Download data

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

```
python demos/grouped_mask_prostate_longitudinal/demo_data.py
```

### Launch demo training

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

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

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

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

### Predict

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

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

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

### 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 `--no-test` to use the original training configuration, such as

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

## Predict

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

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

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

## 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 and fixed images.

## Author

DeepReg Development Team

## Application

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

## Data

Data is an [example CT volume](#) with two labels.

## Instruction

### Installation

Please install DeepReg following the [instructions](#) and change the current directory to the root directory of DeepReg project, i.e. DeepReg/.

### Download data

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

```
python 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
```

## 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
```

## 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

- **GPU memory allocation:**

`--gpu_allow_growth` or `-gr`, if given, TensorFlow will only grow the memory usage as is needed.

By default it allocates all available GPU memory.

Example usage:

- `--gpu_allow_growth`, no extra argument is needed.

- **Load checkpoint:**

`--ckpt_path` or `-k`, specifies the path of the saved model checkpoint, so that the training will be resumed from the given checkpoint.

The path must end with `.ckpt`.

By default it starts training from a random initialization.

Example usage:

- `--ckpt_path weights-epoch2.ckpt` for reloading the given checkpoint.

- **Output root:**

`--log_root`, specifies the directory for saving all logs.

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

Example usage:

- `--log_root /logs` for saving all logs under `/logs`.

- **Output directory:**

`--log_dir` or `-l`, specifies the directory name to save logs.

The directory will be under the `log_root` which is `logs` by default.

By default it creates a timestamp-named directory, e.g. `logs/20200810-194042/`.

Example usage:

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

- **Maximum number of epochs:**

`--max_epochs`, specifies the maximum number of epochs for training and overwrites the value defined in the configuration.

By default, the value is `-1`, meaning the number of epochs will be defined by configuration.

Example usage:

- `--max_epochs 2` for run training only for two epochs.



## Output

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

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

### 3.11.2 Predict

`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:**

`--mode` or `-m`, specifies in which data set the prediction is performed.

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

Example usage:

- `--mode test` for evaluating the model on test data.

## Optional arguments

- **GPU memory allocation:**

`--gpu_allow_growth` or `-gr`, if given, TensorFlow will only grow the memory usage as is needed.

By default it allocates all availables in the GPU memory.

Example usage:

- `--gpu_allow_growth`, no extra argument is needed.

- **Output root:**

`--log_root`, specifies the directory for saving all logs.

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

Example usage:

- `--log_root /logs` for saving all logs under `/logs`.

- **Output directory:**

`--log_dir` or `-l`, specifies the directory name to save logs.

The directory will be under the `log_root` which is `logs` by default.

By default it creates a timestamp-named directory, e.g. `logs/20200810-194042/`.

Example usage:

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

- **Batch size:**

`--batch_size` or `-b`, specifies the mini-batch size (per GPU) for prediction.

The default value is 1.

Example usage:

- `--batch_size 2` for using a mini-batch size of 2.

- **Save outputs in Nifti format:**

The predicted 3D tensors can be saved in Nifti format for further calculation.

By default it saves outputs in Nifti format.

Example usage:

- `--save_nifti`, for saving the outputs in Nifti format.
- `--no_nifti`, for not saving the outputs in Nifti format.

- **Save outputs in png format:**

The predicted 3D tensors can be saved as a slice of 2D images for quick visualization.

As values have to be normalized between 0~255 (or 0~1) for png files (Nifti files are not impacted), all images (`moving_image`, `fixed_image` and `pred_fixed_image`) and displacement/velocity fields (`ddf` and `dvf`) will be normalized before being saved. Labels (`moving_label`, `fixed_label` and `pred_fixed_label`) are not affected as they are already within 0~1.

By default it saves the outputs in png format.

Example usage:

- `--save_png`, for saving the outputs in png format.

- `--save_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 Configuration File

Besides 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`.

### 3.12.1 Dataset section

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

### 3.12.2 Train section

The `train` section defines the neural network training hyper-parameters, by specifying subsections, `model`, `loss`, `optimizer`, `preprocess` and other training hyper-parameter, including `epochs` and `save_period`. See an [example configuration](#), with comments on the available options in each subsection.

This section is highly application-specific. More examples can be found in [DeepReg Demos](#).

## 3.13 Dataset Loader

### 3.13.1 Dataset type

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

- **Paired images**

Images are organized into moving and fixed image pairs.

An example case is two-modalities intra-subject registration, such as registering one subject's MR image to the corresponding ultrasound image.

- **Unpaired images**

Images may be considered independent samples.

An example case is single-modality inter-subject registration, such as registering one CT image to another from different subjects.

- **Grouped images**

Images are organized into multiple groups.

An example case is single-modality intra-subject registration, such as registering time-series images within individual subjects, a group is one subject in this case.

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

### 3.13.2 Dataset requirements

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

- Image
  - DeepReg currently supports 3D images. But images do not have to be of the same shape, and it will be resized to the required shape using linear interpolation.
  - Currently, DeepReg only supports images stored in Nifti files or H5 files. Check [Nifti\\_loader](#) and [h5\\_loader](#) for more details.
  - **Images are automatically normalized** at per-image level: the intensity values  $x$  equals to  $(x - \min(x) + \text{EPS}) / (\max(x) - \min(x) + \text{EPS})$  so that its values are between  $[0, 1]$ . Check `GeneratorDataLoader.data_generator` in [loader interface](#) for more details.
- Label
  - If an image is labeled, the label shape is recommended to be the same as the image shape. Otherwise, the resize might give unexpected behaviours. But each image can have more than one labels.  
  
For instance, an image of shape  $(\text{dim1}, \text{dim2}, \text{dim3})$ , its label shape can be  $(\text{dim1}, \text{dim2}, \text{dim3})$  (single label) or  $(\text{dim1}, \text{dim2}, \text{dim3}, \text{num\_labels})$  (multiple labels).
  - **All labels are assumed to have values between  $[0, 1]$ .** So DeepReg accepts binary segmentation masks or soft labels with float values between  $[0, 1]$ . This is to prevent accidental use of non-one-hot encoding to represent multiple class labels. In case of multi labels, please use one-hot encoding to transform them into multiple channels such that each class has its own binary label.
  - When the images are paired, the moving and fixed images must have the same number of labels.
  - When there are multiple labels, it is assumed that the labels are ordered, such that the channel of index `label_idx` is the same anatomical or pathological structure.
  - Currently, if the data are labeled, each data sample must have at least one label. For missing labels, consider using all zeros as a workaround.

### 3.13.3 Paired images

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

Specifically, given a pair of images

- When the image is unlabeled,
  - moving image of shape  $(m\_dim1, m\_dim2, m\_dim3)$
  - 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:
  dir:
    train: "data/test/h5/paired/train" # folder contains training data
    valid: "data/test/h5/paired/test" # folder contains validation data
    test: "data/test/h5/paired/test" # folder contains test data
  format: "nifti" # value should be nifti / h5
  type: "paired" # value should be paired / unpaired / grouped
  labeled: true # value should be true / false
  moving_image_shape: [16, 16, 16] # value should be like [dim1, dim2, dim3]
  fixed_image_shape: [8, 8, 8] # value should be like [dim1, dim2, dim3]
```

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

- Common configurations
  - dir/train gives the directory containing training data. Same for dir/valid and dir/test.
  - format can only be Nifti or h5 currently.
  - type can be paired, unpaired or grouped, corresponding to the dataset type described above.
  - labeled is a boolean indicating if the data is labeled or not.
- Paired images configurations
  - moving\_image\_shape is the shape of moving images, a list of three integers.
  - fixed\_image\_shape is the shape of fixed images, a list of three integers.

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

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

This is particularly useful when performing an [experiment](#) such as [cross-validation](#).

### File loader

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

### Nifti

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

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

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

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

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

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

Optionally, the data may not be all saved directly under the top directory. They can be further grouped in subdirectories as long as the data paths are consistent.



## H5

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

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

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

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

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

### 3.13.4 Unpaired images

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

Specifically,

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

## Sampling

During each epoch, image pairs will be sampled without replacement. Therefore, given  $N$  images, one epoch will thereby have  $\text{floor}(N / 2)$  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.

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

where

- Common configurations
  - Same as *paired images*.
- Unpaired images configurations
  - `image_shape` is the shape of images, a list of three integers.

## File loader

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

### Nifti

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

`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.

### 3.13.5 Grouped images

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

The requirements are the same as unpaired images. Specifically,

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

### Sampling

For sampling image pairs, DeepReg provides the following options:

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

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

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

### Intra-group

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

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

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

### Inter-group

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

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

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

### Mixed

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

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

## Iterated

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

## Configuration

An example configuration for grouped dataset is provided as follows.

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

where

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

## File loader

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

### Nifti

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

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

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

The leaf directories will be considered as different groups, and file names should be consistent between top directories, e.g.:

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

Check [test grouped Nifti data](#) as an example.

## H5

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

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

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

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

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

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

## 3.14 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](#)

### 3.14.1 Label sampling

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

#### Corresponding label pairs

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

#### Consistent label pairs

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

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

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

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

However,

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

#### Partially labeled image data

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

### Option for iterating all available label pairs

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

## 3.15 Entry Point

### 3.15.1 Train

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

`deepreg.train.build_callbacks` (*log\_dir: str, histogram\_freq: int, save\_period: int*) → list  
Function to prepare callbacks for training.

#### Parameters

- **log\_dir** – directory of logs
- **histogram\_freq** – save the histogram every X epochs
- **save\_period** – save the checkpoint every X epochs

**Returns** a list of callbacks

`deepreg.train.build_config` (*config\_path: (<class 'str'>, <class 'list'>), log\_root: str, log\_dir: str, ckpt\_path: str, max\_epochs: int = -1*) → [*<class 'dict'>*, *<class 'str'>*]  
Function to initialise log directories, assert that checkpointed model is the right type and to parse the configuration for training.

#### Parameters

- **config\_path** – list of str, path to config file
- **log\_root** – str, root of logs
- **log\_dir** – str, path to where training logs to be stored.
- **ckpt\_path** – str, path where model is stored.
- **max\_epochs** – int, if max\_epochs > 0, will use it to overwrite the configuration

#### Returns

- **config**: a dictionary saving configuration
- **log\_dir**: the path of directory to save logs

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

`deepreg.train.train` (*gpu: str, config\_path: (<class 'str'>, <class 'list'>), gpu\_allow\_growth: bool, ckpt\_path: str, log\_dir: str, log\_root: str = 'logs', max\_epochs: int = -1*)  
Function to train a model.

#### Parameters

- **gpu** – str, which local gpu to use to train
- **config\_path** – str, path to configuration set up
- **gpu\_allow\_growth** – bool, whether or not to allocate whole GPU memory to training
- **ckpt\_path** – str, where to store training checkpoints



- **log\_root** – str, root of logs
- **log\_dir** – str, where to store logs in training
- **max\_epochs** – int, if max\_epochs > 0, will use it to overwrite the configuration

### 3.15.2 Predict

Module to perform predictions on data using command line interface.

`deepreg.predict.build_config` (*config\_path*: (<class 'str'>, <class 'list'>), *log\_root*: str, *log\_dir*: str, *ckpt\_path*: str) → [<class 'dict'>, <class 'str'>]

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

#### Parameters

- **config\_path** – string or list of strings, path of configuration files
- **log\_root** – str, root of logs
- **log\_dir** – string, path to store logs.
- **ckpt\_path** – str, path where model is stored.

#### Returns

- config, configuration dictionary
- log\_dir, path of the directory for saving outputs

`deepreg.predict.build_pair_output_path` (*indices*: list, *save\_dir*: str) -> (<class 'str'>, <class 'str'>)

Create directory for saving the paired data

#### Parameters

- **indices** – indices of the pair, the last one is for label
- **save\_dir** – directory of output

#### Returns

- save\_dir, str, directory for saving the moving/fixed image
- label\_dir, str, directory for saving the rest outputs

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

Function to run in command line with argparse to predict results on data for a given model.

`deepreg.predict.predict` (*gpu*: str, *gpu\_allow\_growth*: bool, *ckpt\_path*: str, *mode*: str, *batch\_size*: int, *log\_dir*: str, *sample\_label*: str, *config\_path*: (<class 'str'>, <class 'list'>), *save\_nifti*: bool = True, *save\_png*: bool = True, *log\_root*: str = 'logs')

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

#### Parameters

- **gpu** – str, which env gpu to use.
- **gpu\_allow\_growth** – bool, whether to allow gpu growth or not
- **ckpt\_path** – str, where model is stored, should be like log\_folder/save/xxx.ckpt
- **mode** – train / valid / test, to define which split of dataset to be evaluated
- **batch\_size** – int, batch size to perform predictions in

- **log\_dir** – str, path to store logs
- **sample\_label** – sample/all, not used
- **save\_nifti** – if true, outputs will be saved in nifti format
- **save\_png** – if true, outputs will be saved in png format
- **config\_path** – to overwrite the default config

```
deepreg.predict.predict_on_dataset (dataset: tensorflow.data.Dataset, fixed_grid_ref:
                                     tensorflow.Tensor, model: tensorflow.keras.Model,
                                     model_method: str, 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
- **model\_method** – str, ddf / dvf / affine / conditional
- **save\_dir** – str, 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

### 3.15.3 Warp

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

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

Entry point for warp script.

```
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

## 3.16 Dataset Loader

### 3.16.1 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, mov-
                                                         ing_image_shape:
                                                         (<class 'list'>,
                                                         <class 'tuple'>),
                                                         fixed_image_shape:
                                                         (<class 'list'>,
                                                         <class 'tuple'>))
```

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

#### Parameters

- **file\_loader** –
- **data\_dir\_paths** – path of the directories storing data, the data has to be saved under four different sub-directories: `moving_images`, `fixed_images`, `moving_labels`, `fixed_labels`
- **labeled** – true if the data are labeled
- **sample\_label** –
- **seed** –
- **moving\_image\_shape** – (width, height, depth)
- **fixed\_image\_shape** – (width, height, depth)

**sample\_index\_generator()**

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

**validate\_data\_files()**

Verify all loaders have the same files.

### 3.16.2 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], la-
                                                         beled: bool,
                                                         sample_label:
                                                         str, seed, im-
                                                         age_shape:
                                                         (<class 'list'>,
                                                         <class 'tu-
                                                         ple'>))
```

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 has to be saved under four different sub-directories: images, labels
- **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).

### 3.16.3 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](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],      la-  
        beled:          bool,  
    sample_label:  
        (<class 'str'>,  
         None),         in-  
    tra_group_prob:  
        float,          in-  
    tra_group_option:  
        str,            sam-  
    ple_image_in_group:  
        bool,           seed:  
        (<class 'int'>,  
         None),         im-  
    age_shape:  
        (<class 'list'>,  
         <class 'tuple'>))
```

Load grouped data. Yield indexes of images to load using sample\_index\_generator from GeneratorDataLoader. AbstractUnpairedLoader handles different file formats

#### Parameters

- **file\_loader** – a subclass of FileLoader
- **data\_dir\_paths** – paths of the directory storing data, the data has to be saved under two different sub-directories:
  - images
  - labels
- **labeled** – bool, true if the data is labeled, false if unlabeled
- **sample\_label** – “sample” or “all”, read *get\_label\_indices* in deepreg/dataset/util.py for more details.

- **intra\_group\_prob** – float between 0 and 1,
  - 0 means generating only inter-group samples,
  - 1 means generating only intra-group samples
- **intra\_group\_option** – str, “forward”, “backward, or “unconstrained”
- **sample\_image\_in\_group** – bool,
  - if true, only one image pair will be yielded for each group, so one epoch has num\_groups pairs of data,
  - if false, iterate through this loader will generate all possible pairs
- **seed** – controls the randomness in sampling, if seed=None, then the randomness is not fixed
- **image\_shape** – list or tuple of length 3, corresponding to (dim1, dim2, dim3) of the 3D image

**close()**

Close file loaders

**get\_inter\_sample\_indices()** → list

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

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

All pairs of images in the dataset are registered. Assuming group  $i$  has  $n_i$  images, and that  $N=[n_1, n_2, \dots, n_I]$ , 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  $n_i$  images, then in total the number of samples are -  $\sum(n_i * (n_i-1) / 2)$  for forward/backward -  $\sum(n_i * (n_i-1))$  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.

## 3.17 File Loader

### 3.17.1 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: (<class 'int'>, <class 'tuple'>)*)

Get one data array by specifying an index.

**Parameters index** – the data index which is required

- for paired or unpaired, the index is one single int, data\_index
- for grouped, the index is a tuple of two ints, (group\_index, in\_group\_data\_index)

**Returns** the data array at the specified index

**get\_data\_ids()**

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]

### 3.17.2 Nifti Loader

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

Generalized loader for nifti files.

#### 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 files.

```
get_data (index: (<class 'int'>, <class 'tuple'>)) → numpy.ndarray
```

Get one data array by specifying an index

**Parameters** **index** – the data index which is required

- for paired or unpaired, the index is one single int, data\_index
- for grouped, the index is a tuple of two ints, (group\_index, in\_group\_data\_index)

**Returns** **arr** the data array at the specified index

```
get_data_ids ()
```

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

### 3.17.3 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.

**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 h5 file handles.

**get\_data** (*index: (<class 'int'>, <class 'tuple'>)*) → `numpy.ndarray`

Get one data array by specifying an index

**Parameters** **index** – the data index which is required

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

**Returns** **arr** the data array at the specified index

**get\_data\_ids()**

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]`.



## 3.18 Network

### 3.18.1 DDF / DVF Network

`deepreg.model.network.ddf_dvf.build_ddf_dvf_model` (*moving\_image\_size: tuple, fixed\_image\_size: tuple, index\_size: int, labeled: bool, batch\_size: int, train\_config: dict*)  
→ `tensorflow.keras.Model`

Build a model which outputs DDF/DVF.

#### Parameters

- **moving\_image\_size** – (m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image\_size** – (f\_dim1, f\_dim2, f\_dim3)
- **index\_size** – int, the number of indices for identifying a sample
- **labeled** – bool, indicating if the data is labeled
- **batch\_size** – int, size of mini-batch
- **train\_config** – config for the model and loss

**Returns** the built `tf.keras.Model`

`deepreg.model.network.ddf_dvf.ddf_dvf_forward` (*backbone: tensorflow.keras.Model, moving\_image: tensorflow.Tensor, fixed\_image: tensorflow.Tensor, moving\_label: tensorflow.Tensor, None, moving\_image\_size: tuple, fixed\_image\_size: tuple, output\_dvf: bool*) → [`tensorflow.Tensor`, `None`, `tensorflow.Tensor`, `tensorflow.Tensor`, `tensorflow.Tensor`, `None`, `tensorflow.Tensor`]

Perform the network forward pass.

#### Parameters

- **backbone** – model architecture object, e.g. `model.backbone.local_net`
- **moving\_image** – tensor of shape (batch, m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- **moving\_label** – tensor of shape (batch, m\_dim1, m\_dim2, m\_dim3) or `None`
- **moving\_image\_size** – tuple like (m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image\_size** – tuple like (f\_dim1, f\_dim2, f\_dim3)
- **output\_dvf** – bool, if true, model outputs dvf, if false, model outputs ddf

#### Returns

(dvf, ddf, pred\_fixed\_image, pred\_fixed\_label, fixed\_grid), where

- dvf is the dense velocity field of shape (batch, f\_dim1, f\_dim2, f\_dim3, 3)
- ddf is the dense displacement field of shape (batch, f\_dim1, f\_dim2, f\_dim3, 3)
- pred\_fixed\_image is the predicted (warped) moving image of shape (batch, f\_dim1, f\_dim2, f\_dim3)

- `pred_fixed_label` is the predicted (warped) moving label of shape (batch, `f_dim1`, `f_dim2`, `f_dim3`)
- `fixed_grid` is the grid of shape(`f_dim1`, `f_dim2`, `f_dim3`, 3)

### 3.18.2 Conditional Network

`deepreg.model.network.cond.build_conditional_model` (*moving\_image\_size: tuple, fixed\_image\_size: tuple, index\_size: int, labeled: bool, batch\_size: int, train\_config: dict*) → `tensorflow.keras.Model`

Build a model which outputs predicted fixed label.

#### Parameters

- **`moving_image_size`** – (`m_dim1`, `m_dim2`, `m_dim3`)
- **`fixed_image_size`** – (`f_dim1`, `f_dim2`, `f_dim3`)
- **`index_size`** – int, the number of indices for identifying a sample
- **`labeled`** – bool, indicating if the data is labeled
- **`batch_size`** – int, size of mini-batch
- **`train_config`** – config for the model and loss

**Returns** the built `tf.keras.Model`

`deepreg.model.network.cond.conditional_forward` (*backbone: tensorflow.keras.Model, moving\_image: tensorflow.Tensor, fixed\_image: tensorflow.Tensor, moving\_label: tensorflow.Tensor, None, moving\_image\_size: tuple, fixed\_image\_size: tuple*) → [`tensorflow.Tensor`, `tensorflow.Tensor`]

Perform the network forward pass.

#### Parameters

- **`backbone`** – model architecture object, e.g. `model.backbone.local_net`
- **`moving_image`** – tensor of shape (batch, `m_dim1`, `m_dim2`, `m_dim3`)
- **`fixed_image`** – tensor of shape (batch, `f_dim1`, `f_dim2`, `f_dim3`)
- **`moving_label`** – tensor of shape (batch, `m_dim1`, `m_dim2`, `m_dim3`) or None
- **`moving_image_size`** – tuple like (`m_dim1`, `m_dim2`, `m_dim3`)
- **`fixed_image_size`** – tuple like (`f_dim1`, `f_dim2`, `f_dim3`)

#### Returns

(`pred_fixed_label`, `fixed_grid`), where

- `pred_fixed_label` is the predicted (warped) moving label of shape (batch, `f_dim1`, `f_dim2`, `f_dim3`)
- `fixed_grid` is the grid of shape(`f_dim1`, `f_dim2`, `f_dim3`, 3)

### 3.18.3 Affine Network

`deepreg.model.network.affine.affine_forward` (*backbone: tensorflow.keras.Model, moving\_image: tensorflow.Tensor, fixed\_image: tensorflow.Tensor, moving\_label: tensorflow.Tensor, None, moving\_image\_size: tuple, fixed\_image\_size: tuple*)

Perform the network forward pass.

#### Parameters

- **backbone** – model architecture object, e.g. `model.backbone.local_net`
- **moving\_image** – tensor of shape (batch, m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- **moving\_label** – tensor of shape (batch, m\_dim1, m\_dim2, m\_dim3) or None
- **moving\_image\_size** – tuple like (m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image\_size** – tuple like (f\_dim1, f\_dim2, f\_dim3)

#### Returns

tuple(affine, ddf, pred\_fixed\_image, pred\_fixed\_label, fixed\_grid), where

- affine is the affine transformation matrix predicted by the network (batch, 4, 3)
- ddf is the dense displacement field of shape (batch, f\_dim1, f\_dim2, f\_dim3, 3)
- pred\_fixed\_image is the predicted (warped) moving image of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- pred\_fixed\_label is the predicted (warped) moving label of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- fixed\_grid is the grid of shape (f\_dim1, f\_dim2, f\_dim3, 3)

`deepreg.model.network.affine.build_affine_model` (*moving\_image\_size: tuple, fixed\_image\_size: tuple, index\_size: int, labeled: bool, batch\_size: int, train\_config: dict*)

Build a model which outputs the parameters for affine transformation.

#### Parameters

- **moving\_image\_size** – (m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image\_size** – (f\_dim1, f\_dim2, f\_dim3)
- **index\_size** – int, the number of indices for identifying a sample
- **labeled** – bool, indicating if the data is labeled
- **batch\_size** – int, size of mini-batch
- **train\_config** – config for the model and loss

**Returns** the built `tf.keras.Model`

### 3.18.4 Utils

`deepreg.model.network.build.build_model` (*moving\_image\_size: tuple, fixed\_image\_size: tuple, index\_size: int, labeled: bool, batch\_size: int, train\_config: dict*)

Parsing algorithm types to model building functions.

#### Parameters

- **moving\_image\_size** – [m\_dim1, m\_dim2, m\_dim3]
- **fixed\_image\_size** – [f\_dim1, f\_dim2, f\_dim3]
- **index\_size** – dataset size
- **labeled** – true if the label of moving/fixed images are provided
- **batch\_size** – mini-batch size
- **train\_config** – train configuration

**Returns** the built `tf.keras.Model`

Module to build backbone modules based on passed inputs.

`deepreg.model.network.util.add_ddf_loss` (*model: tensorflow.keras.Model, ddf: tensorflow.Tensor, loss\_config: dict*) → `tensorflow.keras.Model`

Add regularization loss of ddf into model.

#### Parameters

- **model** – `tf.keras.Model`
- **ddf** – tensor of shape (batch, m\_dim1, m\_dim2, m\_dim3, 3)
- **loss\_config** – config for loss

`deepreg.model.network.util.add_image_loss` (*model: tensorflow.keras.Model, fixed\_image: tensorflow.Tensor, pred\_fixed\_image: tensorflow.Tensor, loss\_config: dict*) → `tensorflow.keras.Model`

Add image dissimilarity loss of ddf into model.

#### Parameters

- **model** – `tf.keras.Model`
- **fixed\_image** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- **pred\_fixed\_image** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- **loss\_config** – config for loss

`deepreg.model.network.util.add_label_loss` (*model: tensorflow.keras.Model, grid\_fixed: tensorflow.Tensor, fixed\_label: tensorflow.Tensor, None, pred\_fixed\_label: tensorflow.Tensor, None, loss\_config: dict*) → `tensorflow.keras.Model`

Add label dissimilarity loss of ddf into model.

#### Parameters

- **model** – `tf.keras.Model`
- **grid\_fixed** – tensor of shape (f\_dim1, f\_dim2, f\_dim3, 3)
- **fixed\_label** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)

- **pred\_fixed\_label** – tensor of shape (batch, f\_dim1, f\_dim2, f\_dim3)
- **loss\_config** – config for loss

```
deepreg.model.network.util.build_backbone(image_size: tuple, out_channels: int,
                                           config: dict, method_name: str, registry: deepreg.registry.Registry = <deepreg.registry.Registry object>) → tensorflow.keras.Model
```

Backbone model accepts a single input of shape (batch, dim1, dim2, dim3, ch\_in) and returns a single output of shape (batch, dim1, dim2, dim3, ch\_out).

#### Parameters

- **image\_size** – tuple, dims of image, (dim1, dim2, dim3)
- **out\_channels** – int, number of out channels, ch\_out
- **method\_name** – str, one of ddf, dvf and conditional
- **config** – dict, backbone configuration
- **registry** – the registry object having all backbone classes

**Returns** tf.keras.Model

```
deepreg.model.network.util.build_inputs(moving_image_size: tuple, fixed_image_size: tuple, index_size: int, batch_size: int, labeled: bool)
→ [tensorflow.keras.Input, tensorflow.keras.Input, tensorflow.keras.Input, tensorflow.keras.Input, tensorflow.keras.Input]
```

Configure a pair of moving and fixed images and a pair of moving and fixed labels as model input and returns model input tf.keras.Input

TODO do we absolutely need the batch\_size in Input?

#### Parameters

- **moving\_image\_size** – tuple, dims of moving images, (m\_dim1, m\_dim2, m\_dim3)
- **fixed\_image\_size** – tuple, dims of fixed images, (f\_dim1, f\_dim2, f\_dim3)
- **index\_size** – int, dataset size (number of images)
- **batch\_size** – int, mini-batch size
- **labeled** – Boolean, true if we have label data

**Returns** 5 (if labeled=True) or 3 (if labeled=False) tf.keras.Input objects

## 3.19 Network Backbone

### 3.19.1 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>

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

So, `extract_levels` are between  $[0, E]$  with  $E = \max(\text{extract\_levels})$ , and  $D = \min(\text{extract\_levels})$ .

#### Parameters

- **image\_size** – tuple, such as (dim1, dim2, dim3)
- **out\_channels** – int, number of channels for the extractions
- **num\_channel\_initial** – int, number of initial channels.
- **extract\_levels** – list of int, number of extraction levels.
- **out\_kernel\_initializer** – str, initializer to use for kernels.
- **out\_activation** – str, activation to use at end layer.
- **kwargs** –

**call** (*inputs*, *training=None*, *mask=None*)

Build LocalNet graph based on built layers.

#### Parameters

- **inputs** – image batch, shape = (batch, f\_dim1, f\_dim2, f\_dim3, ch)
- **training** – None or bool.
- **mask** – None or tf.Tensor.

**Returns** tf.Tensor, shape = (batch, f\_dim1, f\_dim2, f\_dim3, out\_channels)

### 3.19.2 Global Net

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

Build GlobalNet for image registration.

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)
- **out\_channels** – int, number of channels for the output
- **num\_channel\_initial** – int, number of initial channels
- **extract\_levels** – list, which levels from net to extract
- **out\_kernel\_initializer** – str, which kernel to use as initializer
- **out\_activation** – str, activation at last layer
- **kwargs** –

**call** (*inputs*, *training=None*, *mask=None*)

Build GlobalNet graph based on built layers.

**Parameters**

- **inputs** – image batch, shape = (batch, f\_dim1, f\_dim2, f\_dim3, ch)
- **training** – None or bool.
- **mask** – None or tf.Tensor.

**Returns** tf.Tensor, shape = (batch, dim1, dim2, dim3, 3)

### 3.19.3 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** – tuple, (dim1, dim2, dim3), dims of input image.
- **out\_channels** – int, number of channels for the output
- **num\_channel\_initial** – int, number of initial channels
- **depth** – int, input is at level 0, bottom is at level depth
- **out\_kernel\_initializer** – str, which kernel to use as initializer
- **out\_activation** – str, activation at last layer
- **pooling** – Boolean, for downsampling, use non-parameterized pooling if true, otherwise use conv3d
- **concat\_skip** – Boolean, when upsampling, concatenate skipped tensor if true, otherwise use addition
- **kwargs** –

**call** (inputs, training=None, mask=None)

Builds graph based on built layers.

**Parameters**

- **inputs** – shape = [batch, f\_dim1, f\_dim2, f\_dim3, in\_channels]
- **training** –
- **mask** –

**Returns** shape = [batch, f\_dim1, f\_dim2, f\_dim3, out\_channels]

## 3.20 Layer

### 3.20.1 Layer

#### Activation

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

Layer wraps tf.keras.activations.get().

##### Parameters

- **identifier** – e.g. “relu”
- **kwargs** –

#### AdditiveUpSampling

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

Layer up-samples 3d tensor and reduce channels using split and sum.

##### Parameters

- **output\_shape** – (out\_dim1, out\_dim2, out\_dim3)
- **strides** – int, 1-D Tensor or list
- **kwargs** –

**call** (inputs, \*\*kwargs)

##### Parameters

- **inputs** – shape = (batch, dim1, dim2, dim3, channels)
- **kwargs** –

**Returns** shape = (batch, out\_dim1, out\_dim2, out\_dim3, channels//stride]

#### Conv3d

**class** deepreg.model.layer.**Conv3d** (\*args: Any, \*\*kwargs: Any)

Layer wraps tf.keras.layers.Conv3D.

##### Parameters

- **filters** – number of channels of the output
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – str, same or valid
- **activation** – str, defines the activation function
- **use\_bias** – bool, whether add bias to output
- **kernel\_initializer** – str, defines the initialization method, defines the initialization method



### Conv3dBlock

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

#### Parameters

- **filters** – number of channels of the output
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – str, same or valid

**call** (inputs, training=None, \*\*kwargs)

#### Parameters

- **inputs** – shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** –

**Returns** shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)

### Conv3dWithResize

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

#### Parameters

- **output\_shape** – tuple, (out\_dim1, out\_dim2, out\_dim3)
- **filters** – int, number of channels of the output
- **kernel\_initializer** – str, defines the initialization method
- **activation** – str, defines the activation function
- **kwargs** –

**call** (inputs, \*\*kwargs)

#### Parameters

- **inputs** – shape = (batch, dim1, dim2, dim3, channels)
- **kwargs** –

**Returns** shape = (batch, out\_dim1, out\_dim2, out\_dim3, channels)

## Deconv3d

**class** deepreg.model.layer.**Deconv3d** (\*args: Any, \*\*kwargs: Any)

Layer wraps tf.keras.layers.Conv3DTranspose and does not requires input shape when initializing.

### Parameters

- **filters** – number of channels of the output
- **output\_shape** – (out\_dim1, out\_dim2, out\_dim3)
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – str, same or valid
- **kwargs** –

## Deconv3dBlock

**class** deepreg.model.layer.**Deconv3dBlock** (\*args: Any, \*\*kwargs: Any)

A deconv3d block having deconv3d - norm - activation.

### Parameters

- **filters** – number of channels of the output
- **output\_shape** – (out\_dim1, out\_dim2, out\_dim3)
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **padding** – str, same or valid
- **kwargs** –

**call** (inputs, training=None, \*\*kwargs)

### Parameters

- **inputs** – shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** –

**Return output** shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)

## Dense

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

Layer wraps tf.keras.layers.Dense and flattens input if necessary.

### Parameters

- **units** – number of hidden units
- **bias\_initializer** – str, default “zeros”
- **kwargs** –

**call** (inputs, \*\*kwargs)

**Parameters**

- **inputs** – shape = (batch, \*vol\_dim, channels)
- **kwargs** – (not used)

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

**DownSampleResnetBlock**

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

A down-sampling resnet conv3d block, with max-pooling or conv3d.

1. convded = conv3d\_block(inputs) # adjust channel
2. skip = residual\_block(convded) # develop feature
3. pooled = pool(skip) # down-sample

**Parameters**

- **filters** – number of channels of the output
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **padding** – str, same or valid

**call** (inputs, training=None, \*\*kwargs)

**Parameters**

- **inputs** – shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** –

**Returns**

- (pooled, skip)
- downsampled, shape = (batch, in\_dim1//2, in\_dim2//2, in\_dim3//2, channels)
  - skipped, shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)

**IntDVF**

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

Layer calculates DVF from DDF.

Reference:

- integrate\_vec of neuron <https://github.com/adalca/neurite/blob/legacy/neuron/utils.py>

**Parameters**

- **fixed\_image\_size** – tuple, (f\_dim1, f\_dim2, f\_dim3)
- **num\_steps** – int, number of steps for integration
- **kwargs** –

**call** (inputs, \*\*kwargs)

**Parameters**

- **inputs** – dvf, shape = (batch, f\_dim1, f\_dim2, f\_dim3, 3), type = float32
- **kwargs** –

**Returns** ddf, shape = (batch, f\_dim1, f\_dim2, f\_dim3, 3)

**LocalNetResidual3dBlock**

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

A resnet conv3d block, simpler than Residual3dBlock.

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

**Parameters**

- **filters** – number of channels of the output
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **kwargs** –

**LocalNetUpSampleResnetBlock**

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

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

**Parameters**

- **filters** – int, number of output channels
- **use\_additive\_upsampling** – bool to used additive upsampling (default is True)
- **kwargs** –

**build** (input\_shape)

**Parameters** **input\_shape** – tuple (nonskip\_tensor\_shape, skip\_tensor\_shape)

**call** (inputs, training=None, \*\*kwargs)

**Parameters**

- **inputs** – list = [inputs\_nonskip, inputs\_skip]
- **training** – training flag for normalization layers (default: None)
- **kwargs** –

**Returns**

## MaxPool3d

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

### Parameters

- **pool\_size** – int or tuple of 3 ints
- **strides** – int or tuple of 3 ints or None, if None default will be pool\_size
- **padding** – str, same or valid
- **kwargs** –

## Norm

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

### Parameters

- **name** – str, batch\_norm or layer\_norm
- **axis** – int
- **kwargs** –

## Residual3dBlock

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

1. convd = conv3d(conv3d\_block(inputs))
2. out = act(norm(convd) + inputs)

### Parameters

- **filters** – int, number of filters in the convolutional layers
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **strides** – int or tuple of 3 ints, e.g. (1,1,1) or 1
- **kwargs** –

**call** (inputs, training=None, \*\*kwargs)

### Parameters

- **inputs** – shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)
- **training** – training flag for normalization layers (default: None)
- **kwargs** –

**Return output** shape = (batch, in\_dim1, in\_dim2, in\_dim3, channels)

## UpSampleResnetBlock

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

An up-sampling resnet conv3d block, with deconv3d.

### Parameters

- **filters** – number of channels of the output
- **kernel\_size** – int or tuple of 3 ints, e.g. (3,3,3) or 3
- **concat** – bool, specify how to combine input and skip connection images. If True, use concatenation if false use sum (default=False).
- **kwargs** –

**build** (input\_shape)

Parameters **input\_shape** – tuple, (downsampled\_image\_shape, skip\_connection image\_shape)

**call** (inputs, training=None, \*\*kwargs)

### Parameters

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

**Returns** shape = (batch, \*skip\_connection\_image\_shape, filters]

## Warping

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

A layer warps an image using DDF.

Reference:

- transform of neuron <https://github.com/adalca/neurite/blob/legacy/neuron/utils.py>  
where vol = image, loc\_shift = ddf

### Parameters

- **fixed\_image\_size** – shape = (f\_dim1, f\_dim2, f\_dim3) or (f\_dim1, f\_dim2, f\_dim3, ch) with the last channel for features
- **kwargs** –

**call** (inputs, \*\*kwargs)

### Parameters

- **inputs** – (ddf, image)
  - ddf, shape = (batch, f\_dim1, f\_dim2, f\_dim3, 3), dtype = float32
  - image, shape = (batch, m\_dim1, m\_dim2, m\_dim3), dtype = float32
- **kwargs** –

**Returns** shape = (batch, f\_dim1, f\_dim2, f\_dim3)

### 3.20.2 Util

Module containing utilities for layer inputs

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

Function returning list containing all combinations of n bits. Given num\_bits binary bits, each bit has value 0 or 1, there are in total  $2^{**}n\_bits$  combinations.

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

**Returns** a list of length  $2^{**}n\_bits$ , `return[i]` is the binary representation of the decimal integer.

**Example**

```
>>> from deepreg.model.layer_util import get_n_bits_combinations
>>> get_n_bits_combinations(3)
[[0, 0, 0], # 0
 [0, 0, 1], # 1
 [0, 1, 0], # 2
 [0, 1, 1], # 3
 [1, 0, 0], # 4
 [1, 0, 1], # 5
 [1, 1, 0], # 6
 [1, 1, 1]] # 7
```

`deepreg.model.layer_util.get_reference_grid(grid_size: (<class 'tuple'>, <class 'list'>)) → tensorflow.Tensor`

Generate a 3D grid with given size.

Reference:

- `volshape_to_meshgrid` of neuron <https://github.com/adalca/neurite/blob/legacy/neuron/utlis.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, weights: 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, weights = [w1, w2, w3] (ignoring the batch dimension).

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

- weight for x = w1 if x = 0 else 1-w1
- weight for y = w2 if y = 0 else 1-w2
- weight for z = w3 if z = 0 else 1-w3

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

$$\begin{aligned} W_{xyz} &= ((1-x) * w1 + x * (1-w1)) * ((1-y) * w2 + y * (1-w2)) * ((1-z) * w3 + z * (1-w3)) \\ &= (W_{xy} * (1-z)) * w3 + (W_{xy} * z) * (1-w3) \end{aligned}$$

where  $W_{xy}$  is the weight for (x, y), let

- $W_{xy0} = W_{xy} * w3$
- $W_{xy1} = W_{xy} * (1-w3)$

So, the final sum V equals

$$\begin{aligned} &\text{sum over } x,y,z (V_{xyz} * W_{xyz}) \\ &= \text{sum over } x,y (V_{xy0} * W_{xy0} + V_{xy1} * W_{xy1}) \\ &= \text{sum over } x,y (W_{xy0} * W_{xy} * w3 + V_{xy1} * W_{xy} * (1-w3)) \\ &= \text{sum over } x,y (W_{xy} * (V_{xy0} * w3 + V_{xy1} * (1-w3))) \end{aligned}$$

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
- **weights** – a list having weights of floor 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.**random\_transform\_generator**(batch\_size: int, scale: float, seed: (<class 'int'>, None) = None) → tensorflow.Tensor

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



for 3D coordinates, affine transformation is

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

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

$$\text{new} = \text{old} * T$$

where

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

the equation can be simplified to

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} * \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

so that

$$\text{new} = \text{old} * T$$

where

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

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

$$x = \text{np.linalg.lstsq}(a, b)$$

such that

$$a x = b$$

In our case,

- a = old
- b = new
- x = T

To generate random transformation, we choose to add random perturbation to corner coordinates as follows: for corner of coordinates (x, y, z), the noise is

$$-(x, y, z) .* (r1, r2, r3)$$

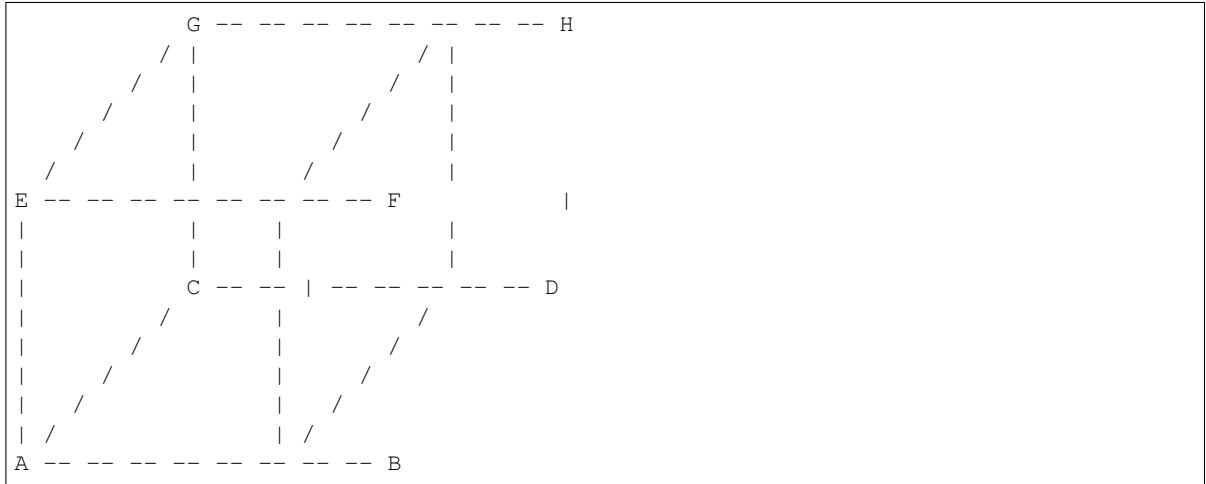
where ri is a random number between (0, scale). So

$$(x', y', z') = (x, y, z) .* (1-r1, 1-r2, 1-r3)$$

Thus, we can directly sample between 1-scale and 1 instead

We choose to calculate the transformation based on four corners in a cube centered at (0, 0, 0). A cube is shown as below, where

- $C = (-1, -1, -1)$
- $G = (-1, -1, 1)$
- $D = (-1, 1, -1)$
- $A = (1, -1, -1)$



#### Parameters

- **batch\_size** – int
- **scale** – a float number between 0 and 1
- **seed** – control the randomness

**Returns** shape = (batch, 4, 3)

`deepreg.model.layer_util.resample(vol, loc, interpolation='linear')`

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 `interp` <https://github.com/adalca/neurite/blob/legacy/neuron/utis.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  $\text{loc}[b, l1, \dots, l_n, :] = [v1, \dots, v_n]$  is of shape (n,), which represents a point in vol, with coordinates (v1, ..., vn)
- **interpolation** – linear only, TODO support nearest

**Returns** shape = (batch, l\_dim 1, ..., l\_dim n)

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

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

- resize dim2 and dim3
- resize dim1 and dim2

#### Parameters

- **image** – tensor of shape = (batch, dim1, dim2, dim3, channels) or (batch, dim1, dim2, dim3) or (dim1, dim2, dim3)
- **size** – tuple, (out\_dim1, out\_dim2, out\_dim3)
- **method** – str, one of tf.image.ResizeMethod

**Returns** tensor of shape = (batch, out\_dim1, out\_dim2, out\_dim3, channels) or (batch, dim1, dim2, dim3) or (dim1, dim2, dim3)

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

Perform transformation on the grid.

- $\text{grid\_padded}[i,j,k,:] = [i \ j \ k \ 1]$
- $\text{grid\_warped}[b,i,j,k,p] = \text{sum\_over\_q} (\text{grid\_padded}[i,j,k,q] * \text{theta}[b,q,p])$

#### Parameters

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

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

`deepreg.model.layer_util.warp_image_ddf` (*image*: tensorflow.Tensor, *ddf*: tensorflow.Tensor, *grid\_ref*: tensorflow.Tensor, None) → tensorflow.Tensor

Warp an image with given DDF.

#### Parameters

- **image** – an image to be warped, shape = (batch, m\_dim1, m\_dim2, m\_dim3) or (batch, m\_dim1, m\_dim2, m\_dim3, ch)
- **ddf** – shape = (batch, f\_dim1, f\_dim2, f\_dim3, 3)
- **grid\_ref** – shape = (1, f\_dim1, f\_dim2, f\_dim3, 3) or None, if None grid\_ref will be calculated based on ddf

**Returns** shape = (batch, f\_dim1, f\_dim2, f\_dim3) or (batch, f\_dim1, f\_dim2, f\_dim3, ch)

## 3.21 Loss

### 3.21.1 Image Loss

Module provides different loss functions for calculating the dissimilarities between images.

`deepreg.model.loss.image.dissimilarity_fn` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*, *name*: *str*, *\*\*kwargs*) → *tensorflow.Tensor*

Returns the calculated dissimilarity for each batch.

#### Parameters

- **y\_true** – fixed\_image, shape = (batch, f\_dim1, f\_dim2, f\_dim3)
- **y\_pred** – warped\_moving\_image, shape = (batch, f\_dim1, f\_dim2, f\_dim3)
- **name** – name of the dissimilarity function
- **kwargs** – absorb additional parameters

**Returns** shape = (batch,)

`deepreg.model.loss.image.global_mutual_information` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*, *num\_bins*: *int* = 23, *sigma\_ratio*: *float* = 0.5) → *tensorflow.Tensor*

Differentiable global mutual information loss via Parzen windowing method.

Reference: <https://dspace.mit.edu/handle/1721.1/123142>, Section 3.1, equation 3.1-3.5, Algorithm 1

#### Parameters

- **y\_true** – shape = (batch, dim1, dim2, dim3, ch)
- **y\_pred** – shape = (batch, dim1, dim2, dim3, ch)
- **num\_bins** – int, number of bins for intensity
- **sigma\_ratio** – float, a hyper param for gaussian function

**Returns** shape = (batch,)

`deepreg.model.loss.image.local_normalized_cross_correlation` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*, *kernel\_size*: *int* = 9, *kernel\_type*: *str* = 'rectangular', *\*\*kwargs*) → *tensorflow.Tensor*

Local squared zero-normalized cross-correlation. The loss is based on a moving kernel/window over the *y\_true*/*y\_pred*, within the window the square of zncc is calculated. The kernel can be a rectangular / triangular / gaussian window. The final loss is the averaged loss over all windows.

Reference:

- Zero-normalized cross-correlation (ZNCC): <https://en.wikipedia.org/wiki/Cross-correlation>
- Code: <https://github.com/voxelmorph/voxelmorph/blob/legacy/src/losses.py>

#### Parameters

- **y\_true** – shape = (batch, dim1, dim2, dim3, ch)

- **y\_pred** – shape = (batch, dim1, dim2, dim3, ch)
- **kernel\_size** – int. Kernel size or kernel sigma for kernel\_type='gauss'.
- **kernel\_type** – str ('triangular', 'gaussian' default: 'rectangular')
- **kwargs** – absorb additional parameters

**Returns** shape = (batch,)

`deepreg.model.loss.image.ssd(y_true: tensorflow.Tensor, y_pred: tensorflow.Tensor) → tensorflow.Tensor`  
Sum of squared distance between y\_true and y\_pred.

**Parameters**

- **y\_true** – shape = (batch, dim1, dim2, dim3, ch)
- **y\_pred** – shape = (batch, dim1, dim2, dim3, ch)

**Returns** shape = (batch,)

### 3.21.2 Label Loss

Module provides different loss functions for calculating the dissimilarities between labels.

`deepreg.model.loss.label.cauchy_kernel1d(sigma: int) → tensorflow.Tensor`  
Approximating cauchy kernel in 1d.

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

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

`deepreg.model.loss.label.compute_centroid(mask: tensorflow.Tensor, grid: tensorflow.Tensor) → tensorflow.Tensor`  
Calculate the centroid of the mask.

**Parameters**

- **mask** – shape = (batch, dim1, dim2, dim3)
- **grid** – shape = (dim1, dim2, dim3, 3)

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

`deepreg.model.loss.label.compute_centroid_distance(y_true: tensorflow.Tensor, y_pred: tensorflow.Tensor, grid: tensorflow.Tensor) → tensorflow.Tensor`  
Calculate the L2-distance between two tensors' centroids.

**Parameters**

- **y\_true** – tensor, shape = (batch, dim1, dim2, dim3)
- **y\_pred** – tensor, shape = (batch, dim1, dim2, dim3)
- **grid** – tensor, shape = (dim1, dim2, dim3, 3)

**Returns** shape = (batch,)

`deepreg.model.loss.label.dice_score(y_true: tensorflow.Tensor, y_pred: tensorflow.Tensor, binary: bool = False) → tensorflow.Tensor`  
Calculates dice score:

1. num = 2 \* y\_true \* y\_pred

2.  $\text{denom} = y_{\text{true}} + y_{\text{pred}}$
3.  $\text{dice score} = \text{num} / \text{denom}$

where num and denom are summed over the entire image first.

**Parameters**

- **y\_true** – shape = (batch, dim1, dim2, dim3)
- **y\_pred** – shape = (batch, dim1, dim2, dim3)
- **binary** – True if the y should be projected to 0 or 1

**Returns** shape = (batch,)

`deepreg.model.loss.label.dice_score_generalized` (*y\_true: tensorflow.Tensor, y\_pred: tensorflow.Tensor, pos\_weight: float = 1, neg\_weight: float = 0*) → *tensorflow.Tensor*

Calculates weighted dice score:

1. let  $y_{\text{prod}} = y_{\text{true}} * y_{\text{pred}}$  and  $y_{\text{sum}} = y_{\text{true}} + y_{\text{pred}}$
2.  $\text{num} = 2 * (\text{pos\_w} * y_{\text{true}} * y_{\text{pred}} + \text{neg\_w} * (1y_{\text{true}}) * (1y_{\text{pred}}))$   
 $= 2 * ((\text{pos\_w} + \text{neg\_w}) * y_{\text{prod}} - \text{neg\_w} * y_{\text{sum}} + \text{neg\_w})$
3.  $\text{denom} = (\text{pos\_w} * (y_{\text{true}} + y_{\text{pred}}) + \text{neg\_w} * (1y_{\text{true}} + 1y_{\text{pred}}))$   
 $= (\text{pos\_w} - \text{neg\_w}) * y_{\text{sum}} + 2 * \text{neg\_w}$
4.  $\text{dice score} = \text{num} / \text{denom}$

where num and denom are summed over the entire image first.

**Parameters**

- **y\_true** – shape = (batch, dim1, dim2, dim3)
- **y\_pred** – shape = (batch, dim1, dim2, dim3)
- **pos\_weight** – weight of positive class, default = 1
- **neg\_weight** – weight of negative class, default = 0

**Returns** shape = (batch,)

`deepreg.model.loss.label.foreground_proportion` (*y: tensorflow.Tensor*) → *tensorflow.Tensor*

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

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

**Returns** shape = (batch,)

`deepreg.model.loss.label.gauss_kernel1d` (*sigma: int*) → *tensorflow.Tensor*

Calculates a gaussian kernel.

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

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

`deepreg.model.loss.label.get_dissimilarity_fn` (*config: dict*) → *Callable*

Parse arguments from a configuration dictionary and return the loss by averaging batch loss returned by multi- or single-scale loss functions.

**Parameters** **config** – dict, containing configuration for training.

**Returns** loss function, which returns a tensor of shape (batch, )

`deepreg.model.loss.label.jaccard_index` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*) → *tensorflow.Tensor*

Calculates jaccard index:

1.  $\text{num} = y\_true * y\_pred$
2.  $\text{denom} = y\_true + y\_pred - y\_true * y\_pred$
3.  $\text{jaccard index} = \text{num} / \text{denom}$

#### Parameters

- **y\_true** – shape = (batch, dim1, dim2, dim3)
- **y\_pred** – shape = (batch, dim1, dim2, dim3)

**Returns** shape = (batch,)

`deepreg.model.loss.label.multi_scale_loss` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*, *loss\_type*: *str*, *loss\_scales*: *list*) → *tensorflow.Tensor*

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

#### Parameters

- **y\_true** – tensor, shape = (batch, dim1, dim2, dim3)
- **y\_pred** – tensor, shape = (batch, dim1, dim2, dim3)
- **loss\_type** – string, indicating which loss to pass to function `single_scale_loss`.

Supported:

- cross-entropy
- mean-squared
- dice
- dice\_generalized
- jaccard

- **loss\_scales** – list, values of sigma to pass to func `gauss_kernel_1d`.

**Returns** (batch,)

`deepreg.model.loss.label.separable_filter3d` (*tensor*: *tensorflow.Tensor*, *kernel*: *tensorflow.Tensor*) → *tensorflow.Tensor*

Creates a 3d separable filter.

Here `tf.nn.conv3d` accepts the *filters* argument of shape (filter\_depth, filter\_height, filter\_width, in\_channels, out\_channels), where the first axis of *filters* is the depth not batch, and the input to `tf.nn.conv3d` is of shape (batch, in\_depth, in\_height, in\_width, in\_channels).

#### Parameters

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

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

`deepreg.model.loss.label.single_scale_loss` (*y\_true*: *tensorflow.Tensor*, *y\_pred*: *tensorflow.Tensor*, *loss\_type*: *str*) → *tensorflow.Tensor*

Calculate the loss on two tensors based on defined loss.

**Parameters**

- **y\_true** – tensor, shape = (batch, dim1, dim2, dim3)
- **y\_pred** – tensor, shape = (batch, dim1, dim2, dim3)
- **loss\_type** – string, indicating which loss to pass to function `single_scale_loss`.

Supported:

- cross-entropy
- mean-squared
- dice
- dice\_generalized
- jaccard

**Returns** shape = (batch,)

`deepreg.model.loss.label_squared_error` (*y\_true: tensorflow.Tensor, y\_pred: tensorflow.Tensor*) → *tensorflow.Tensor*

Calculates the mean squared difference between `y_true`, `y_pred`.

`mean((y_true - y_pred)(y_true - y_pred))`

**Parameters**

- **y\_true** – tensor, shape = (batch, dim1, dim2, dim3)
- **y\_pred** – shape = (batch, dim1, dim2, dim3)

**Returns** shape = (batch,)

`deepreg.model.loss.label_weighted_binary_cross_entropy` (*y\_true: tensorflow.Tensor, y\_pred: tensorflow.Tensor, pos\_weight: float = 1*) → *tensorflow.Tensor*

Calculates weighted binary cross- entropy:

`-loss = pos_w * y_true log(y_pred) - (1-y_true) log(1-y_pred)`

**Parameters**

- **y\_true** – shape = (batch, dim1, dim2, dim3)
- **y\_pred** – shape = (batch, dim1, dim2, dim3)
- **pos\_weight** – weight of positive class, scalar. Default value is 1

**Returns** shape = (batch,)

### 3.21.3 Deformation Loss

Module provides regularization energy functions for ddf.

`deepreg.model.loss.deform.compute_bending_energy` (*ddf: tensorflow.Tensor*) → *tensorflow.Tensor*

Calculate the bending energy based on second-order differentiation of ddf using central finite difference.

**Parameters** **ddf** – shape = (batch, m\_dim1, m\_dim2, m\_dim3, 3)

**Returns** shape = (batch, )



`deepreg.model.loss.deform.compute_gradient_norm` (*ddf: tensorflow.Tensor, ll: bool = False*) → `tensorflow.Tensor`

Calculate the L1/L2 norm of the first-order differentiation of *ddf* using central finite difference.

**Parameters**

- **ddf** – shape = (batch, m\_dim1, m\_dim2, m\_dim3, 3)
- **ll** – bool true if calculate L1 norm, otherwise L2 norm

**Returns** shape = (batch, )

`deepreg.model.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.model.loss.deform.gradient_dxyz` (*xyz: 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**

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

**Returns** shape = (... , 3)

`deepreg.model.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.model.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)

`deepreg.model.loss.deform.local_displacement_energy` (*ddf: tensorflow.Tensor, energy\_type: str, \*\*kwargs*) → `tensorflow.Tensor`

Calculate the displacement energy of the *ddf* based on finite difference.

**Parameters**

- **ddf** – shape = (batch, m\_dim1, m\_dim2, m\_dim3, 3)
- **energy\_type** – type of the energy
- **kwargs** – absorb additional arguments

**Returns** shape = (batch,)

## 3.22 Optimizer

Functions parsing the config optimizer options

`deepreg.model.optimizer.build_optimizer(optimizer_config: dict)`

Parsing the optimiser options and parameters from config dictionary.

**Parameters** `optimizer_config` – unpacked dictionary for the optimiser returned from `yaml.load`, optimiser options and parameters

**Returns** `tf.keras.optimizers` object

## 3.23 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.

### 3.23.1 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.

### 3.23.2 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 it as working in progress.

## 3.23.3 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](#).

## 3.24 Unit Test

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

The testing is checked via [Travis-CI](#) 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.

### 3.24.1 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.

### 3.24.2 Example unit test

We provide here an example to help understanding the requirements.

```
import pytest
import logging

def subtract(x: int) -> int:
    """
    A function subtracts one from a non-negative integer.
    :param x: a non-negative integer
```

(continues on next page)

(continued from previous page)

```

        :return: x - 1
        """
        assert isinstance(x, int), f"input {x} is not int"
        assert x >= 0, f"input {x} is negative"
        if x == 0:
            logging.warning("input is zero")
        return x - 1

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

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

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

```

where

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

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

## 3.25 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.

### 3.25.1 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.

### 3.25.2 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;

### 3.25.3 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`.

### 3.25.4 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`;

### 3.25.5 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.

## 3.26 Documentation

We use [Sphinx](#) to organize the documentation and it is hosted in [ReadTheDocs](#).

### 3.26.1 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.

### 3.26.2 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:: <markdown file path>` to include the markdown source.

Check the source code of [paired lung CT image registration page](#) as an example.

## 3.27 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.

### 3.27.1 Creating a new Release

From the main DeepReg repository page, head to [releases](#). From here, you can [draft a new release](#).

### 3.27.2 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.

### 3.27.3 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

- `deepreg.dataset.loader.grouped_loader,`  
[64](#)
- `deepreg.dataset.loader.h5_loader,` [68](#)
- `deepreg.dataset.loader.nifti_loader,` [67](#)
- `deepreg.dataset.loader.paired_loader,`  
[62](#)
- `deepreg.dataset.loader.unpaired_loader,`  
[63](#)
- `deepreg.model.layer_util,` [83](#)
- `deepreg.model.loss.deform,` [92](#)
- `deepreg.model.loss.image,` [88](#)
- `deepreg.model.loss.label,` [89](#)
- `deepreg.model.network.affine,` [71](#)
- `deepreg.model.network.build,` [72](#)
- `deepreg.model.network.cond,` [70](#)
- `deepreg.model.network.ddf_dvf,` [69](#)
- `deepreg.model.network.util,` [72](#)
- `deepreg.model.optimizer,` [94](#)
- `deepreg.predict,` [61](#)
- `deepreg.train,` [60](#)
- `deepreg.warp,` [62](#)



## A

Activation (class in *deepreg.model.layer*), 76

add\_ddf\_loss() (in module *deepreg.model.network.util*), 72

add\_image\_loss() (in module *deepreg.model.network.util*), 72

add\_label\_loss() (in module *deepreg.model.network.util*), 72

AdditiveUpSampling (class in *deepreg.model.layer*), 76

affine\_forward() (in module *deepreg.model.network.affine*), 71

## B

build() (*deepreg.model.layer.LocalNetUpSampleResnetBlock* method), 80

build() (*deepreg.model.layer.UpSampleResnetBlock* method), 82

build\_affine\_model() (in module *deepreg.model.network.affine*), 71

build\_backbone() (in module *deepreg.model.network.util*), 73

build\_callbacks() (in module *deepreg.train*), 60

build\_conditional\_model() (in module *deepreg.model.network.cond*), 70

build\_config() (in module *deepreg.predict*), 61

build\_config() (in module *deepreg.train*), 60

build\_ddf\_dvf\_model() (in module *deepreg.model.network.ddf\_dvf*), 69

build\_inputs() (in module *deepreg.model.network.util*), 73

build\_model() (in module *deepreg.model.network.build*), 72

build\_optimizer() (in module *deepreg.model.optimizer*), 94

build\_pair\_output\_path() (in module *deepreg.predict*), 61

## C

call() (*deepreg.model.backbone.global\_net.GlobalNet* method), 74

call() (*deepreg.model.backbone.local\_net.LocalNet* method), 74

call() (*deepreg.model.backbone.u\_net.UNet* method), 75

call() (*deepreg.model.layer.AdditiveUpSampling* method), 76

call() (*deepreg.model.layer.Conv3dBlock* method), 77

call() (*deepreg.model.layer.Conv3dWithResize* method), 77

call() (*deepreg.model.layer.Deconv3dBlock* method), 78

call() (*deepreg.model.layer.Dense* method), 78

call() (*deepreg.model.layer.DownSampleResnetBlock* method), 79

call() (*deepreg.model.layer.IntDVF* method), 79

call() (*deepreg.model.layer.LocalNetUpSampleResnetBlock* method), 80

call() (*deepreg.model.layer.Residual3dBlock* method), 81

call() (*deepreg.model.layer.UpSampleResnetBlock* method), 82

call() (*deepreg.model.layer.Warping* method), 82

cauchy\_kernelld() (in module *deepreg.model.loss.label*), 89

close() (*deepreg.dataset.loader.grouped\_loader.GroupedDataLoader* method), 65

close() (*deepreg.dataset.loader.h5\_loader.H5FileLoader* method), 68

close() (*deepreg.dataset.loader.interface.FileLoader* method), 66

close() (*deepreg.dataset.loader.nifti\_loader.NiftiFileLoader* method), 67

close() (*deepreg.dataset.loader.unpaired\_loader.UnpairedDataLoader* method), 64

compute\_bending\_energy() (in module *deepreg.model.loss.deform*), 92

compute\_centroid() (in module *deepreg.model.loss.label*), 89

compute\_centroid\_distance() (in module *deepreg.model.loss.label*), 89

compute\_gradient\_norm() (in module *deepreg.model.loss.deform*), 92

`conditional_forward()` (in module `deep-reg.model.network.cond`), 70  
`Conv3d` (class in `deepreg.model.layer`), 76  
`Conv3dBlock` (class in `deepreg.model.layer`), 77  
`Conv3dWithResize` (class in `deepreg.model.layer`), 77

## D

`ddf_dvf_forward()` (in module `deep-reg.model.network.ddf_dvf`), 69  
`Deconv3d` (class in `deepreg.model.layer`), 78  
`Deconv3dBlock` (class in `deepreg.model.layer`), 78  
`deepreg.dataset.loader.grouped_loader` module, 64  
`deepreg.dataset.loader.h5_loader` module, 68  
`deepreg.dataset.loader.nifti_loader` module, 67  
`deepreg.dataset.loader.paired_loader` module, 62  
`deepreg.dataset.loader.unpaired_loader` module, 63  
`deepreg.model.layer_util` module, 83  
`deepreg.model.loss.deform` module, 92  
`deepreg.model.loss.image` module, 88  
`deepreg.model.loss.label` module, 89  
`deepreg.model.network.affine` module, 71  
`deepreg.model.network.build` module, 72  
`deepreg.model.network.cond` module, 70  
`deepreg.model.network.ddf_dvf` module, 69  
`deepreg.model.network.util` module, 72  
`deepreg.model.optimizer` module, 94  
`deepreg.predict` module, 61  
`deepreg.train` module, 60  
`deepreg.warp` module, 62  
`Dense` (class in `deepreg.model.layer`), 78  
`dice_score()` (in module `deepreg.model.loss.label`), 89  
`dice_score_generalized()` (in module `deep-reg.model.loss.label`), 90

`dissimilarity_fn()` (in module `deep-reg.model.loss.image`), 88  
`DownSampleResnetBlock` (class in `deep-reg.model.layer`), 79

## F

`FileLoader` (class in `deep-reg.dataset.loader.interface`), 66  
`foreground_proportion()` (in module `deep-reg.model.loss.label`), 90

## G

`gauss_kernelld()` (in module `deep-reg.model.loss.label`), 90  
`get_data()` (`deepreg.dataset.loader.h5_loader.H5FileLoader` method), 68  
`get_data()` (`deepreg.dataset.loader.interface.FileLoader` method), 66  
`get_data()` (`deepreg.dataset.loader.nifti_loader.NiftiFileLoader` method), 67  
`get_data_ids()` (`deep-reg.dataset.loader.h5_loader.H5FileLoader` method), 68  
`get_data_ids()` (`deep-reg.dataset.loader.interface.FileLoader` method), 66  
`get_data_ids()` (`deep-reg.dataset.loader.nifti_loader.NiftiFileLoader` method), 67  
`get_dissimilarity_fn()` (in module `deep-reg.model.loss.label`), 90  
`get_inter_sample_indices()` (`deep-reg.dataset.loader.grouped_loader.GroupedDataLoader` method), 65  
`get_intra_sample_indices()` (`deep-reg.dataset.loader.grouped_loader.GroupedDataLoader` method), 65  
`get_n_bits_combinations()` (in module `deep-reg.model.layer_util`), 83  
`get_num_groups()` (`deep-reg.dataset.loader.interface.FileLoader` method), 66  
`get_num_images()` (`deep-reg.dataset.loader.h5_loader.H5FileLoader` method), 68  
`get_num_images()` (`deep-reg.dataset.loader.interface.FileLoader` method), 66  
`get_num_images()` (`deep-reg.dataset.loader.nifti_loader.NiftiFileLoader` method), 67  
`get_num_images_per_group()` (`deep-reg.dataset.loader.interface.FileLoader` method), 66

`get_reference_grid()` (in module *deepreg.model.layer\_util*), 83  
`global_mutual_information()` (in module *deepreg.model.loss.image*), 88  
`GlobalNet` (class in *deepreg.model.backbone.global\_net*), 74  
`gradient_dx()` (in module *deepreg.model.loss.deform*), 93  
`gradient_dxyz()` (in module *deepreg.model.loss.deform*), 93  
`gradient_dy()` (in module *deepreg.model.loss.deform*), 93  
`gradient_dz()` (in module *deepreg.model.loss.deform*), 93  
`GroupedDataLoader` (class in *deepreg.dataset.loader.grouped\_loader*), 64

## H

`H5FileLoader` (class in *deepreg.dataset.loader.h5\_loader*), 68

## I

`IntDVF` (class in *deepreg.model.layer*), 79

## J

`jaccard_index()` (in module *deepreg.model.loss.label*), 91

## L

`load_nifti_file()` (in module *deepreg.dataset.loader.nifti\_loader*), 67  
`local_displacement_energy()` (in module *deepreg.model.loss.deform*), 93  
`local_normalized_cross_correlation()` (in module *deepreg.model.loss.image*), 88  
`LocalNet` (class in *deepreg.model.backbone.local\_net*), 73  
`LocalNetResidual3dBlock` (class in *deepreg.model.layer*), 80  
`LocalNetUpSampleResnetBlock` (class in *deepreg.model.layer*), 80

## M

`main()` (in module *deepreg.predict*), 61  
`main()` (in module *deepreg.train*), 60  
`main()` (in module *deepreg.warp*), 62  
`MaxPool3d` (class in *deepreg.model.layer*), 81  
module  
    *deepreg.dataset.loader.grouped\_loader*.`sample_index_generator()` (deep-  
        64  
    *deepreg.dataset.loader.h5\_loader*, 68  
    *deepreg.dataset.loader.nifti\_loader*, 67  
        *deepreg.dataset.loader.paired\_loader*, 62  
        *deepreg.dataset.loader.unpaired\_loader*, 63  
        *deepreg.model.layer\_util*, 83  
        *deepreg.model.loss.deform*, 92  
        *deepreg.model.loss.image*, 88  
        *deepreg.model.loss.label*, 89  
        *deepreg.model.network.affine*, 71  
        *deepreg.model.network.build*, 72  
        *deepreg.model.network.cond*, 70  
        *deepreg.model.network.ddf\_dvf*, 69  
        *deepreg.model.network.util*, 72  
        *deepreg.model.optimizer*, 94  
        *deepreg.predict*, 61  
        *deepreg.train*, 60  
        *deepreg.warp*, 62  
    *multi\_scale\_loss()* (in module *deepreg.model.loss.label*), 91

## N

`NiftiFileLoader` (class in *deepreg.dataset.loader.nifti\_loader*), 67  
`Norm` (class in *deepreg.model.layer*), 81

## P

`PairedDataLoader` (class in *deepreg.dataset.loader.paired\_loader*), 62  
`predict()` (in module *deepreg.predict*), 61  
`predict_on_dataset()` (in module *deepreg.predict*), 62  
`pyramid_combination()` (in module *deepreg.model.layer\_util*), 83

## R

`random_transform_generator()` (in module *deepreg.model.layer\_util*), 84  
`resample()` (in module *deepreg.model.layer\_util*), 86  
`Residual3dBlock` (class in *deepreg.model.layer*), 81  
`resize3d()` (in module *deepreg.model.layer\_util*), 87

## S

`sample_index_generator()` (deep-  
    *reg.dataset.loader.grouped\_loader*.`GroupedDataLoader`  
    method), 65  
`sample_index_generator()` (deep-  
    *reg.dataset.loader.paired\_loader*.`PairedDataLoader`  
    method), 63  
`sample_index_generator()` (deep-  
    *reg.dataset.loader.unpaired\_loader*.`UnpairedDataLoader`  
    method), 64  
`separable_filter3d()` (in module *deepreg.model.loss.label*), 91

`set_data_structure()` (deep-reg.dataset.loader.h5\_loader.H5FileLoader method), 68

`set_data_structure()` (deep-reg.dataset.loader.interface.FileLoader method), 66

`set_data_structure()` (deep-reg.dataset.loader.nifti\_loader.NiftiFileLoader method), 67

`set_group_structure()` (deep-reg.dataset.loader.h5\_loader.H5FileLoader method), 68

`set_group_structure()` (deep-reg.dataset.loader.interface.FileLoader method), 66

`set_group_structure()` (deep-reg.dataset.loader.nifti\_loader.NiftiFileLoader method), 67

`single_scale_loss()` (in module deep-reg.model.loss.label), 91

`squared_error()` (in module deep-reg.model.loss.label), 92

`ssd()` (in module deepreg.model.loss.image), 89

## T

`train()` (in module deepreg.train), 60

## U

UNet (class in deepreg.model.backbone.u\_net), 75

UnpairedDataLoader (class in deep-reg.dataset.loader.unpaired\_loader), 63

UpSampleResnetBlock (class in deep-reg.model.layer), 82

## V

`validate_data_files()` (deep-reg.dataset.loader.grouped\_loader.GroupedDataLoader method), 65

`validate_data_files()` (deep-reg.dataset.loader.paired\_loader.PairedDataLoader method), 63

`validate_data_files()` (deep-reg.dataset.loader.unpaired\_loader.UnpairedDataLoader method), 64

## W

`warp()` (in module deepreg.warp), 62

`warp_grid()` (in module deepreg.model.layer\_util), 87

`warp_image_ddf()` (in module deep-reg.model.layer\_util), 87

Warping (class in deepreg.model.layer), 82

`weighted_binary_cross_entropy()` (in module deepreg.model.loss.label), 92