Department of Medical Physics and Biomedical Engineering
UCL Centre for Medical Image Computing
Wellcome / EPSRC Centre for Surgical and Interventional Sciences

UCL

Open-Source Code, Research, Education and DeepReg, One or Four Things?

Yipeng Hu
yipeng.hu@ucl.ac.uk

Turing Workshop on Open-Source AI Software for Healthcare

21st November 2022, The Alan Turing Institute

CANDI Computer Assisted Navigation, Diagnosis and Intervention

wellcome EPSRC centre interventional+ surgical sciences

cmic Centre for Medical Image Computing

Software - Research - Education
Readability (train script)
Package or tool (network class)
Pre-defined workflow (data loader)
Manual, documentation or tutorial (docs vs papers)
Test (alternative codebase)

Promotion

Clinical deployment

Where are we heading

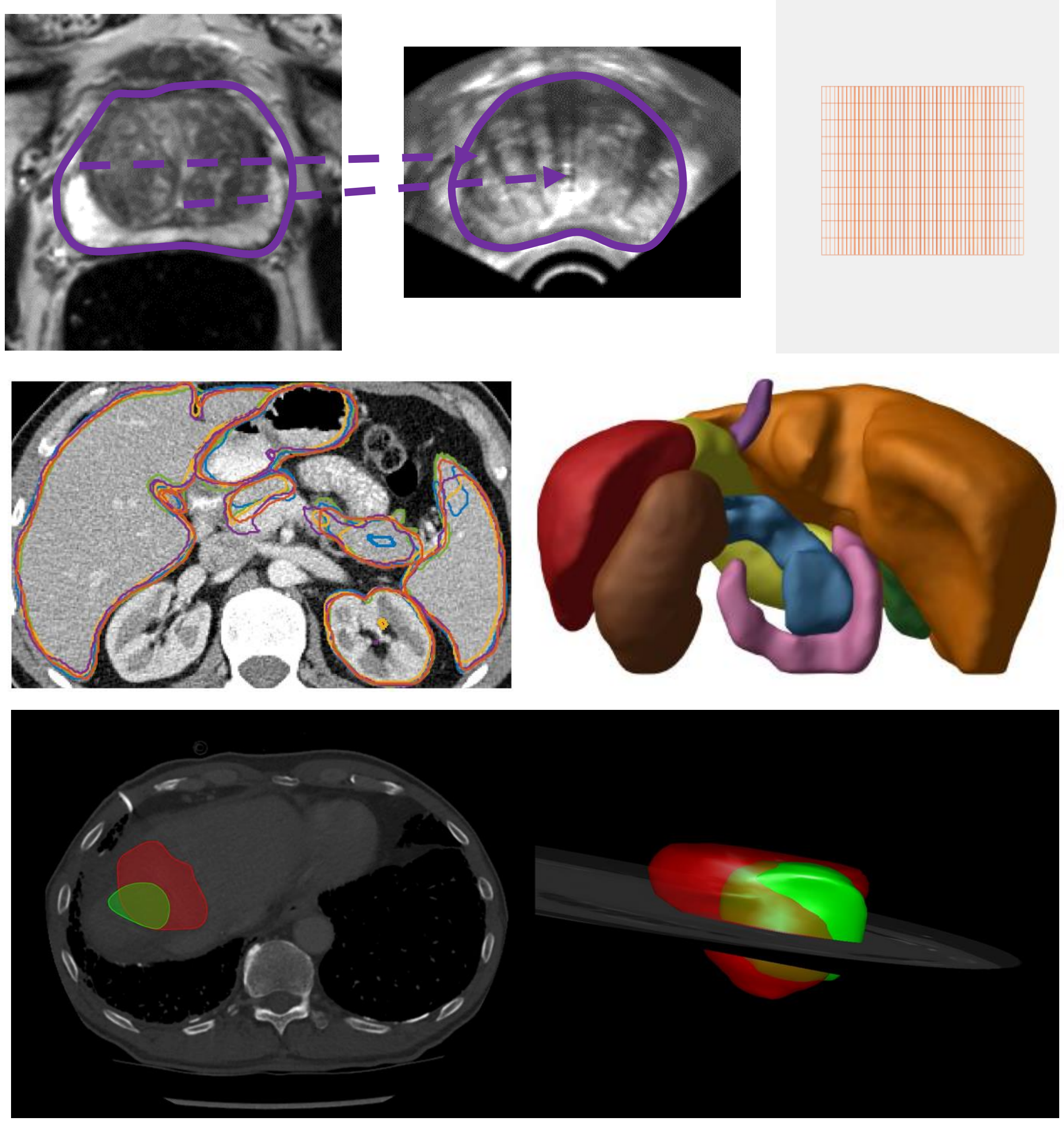
Medical Image Registration

Registration applications

- Multi-modal, e.g. image-guided interventions
- Inter-subject, e.g. atlas-based segmentation
- Medical image analysis, e.g. longitudinal analysis

Why correspondence is a problem

- Different imaging coordinates
- Motion and deformation
- Homology

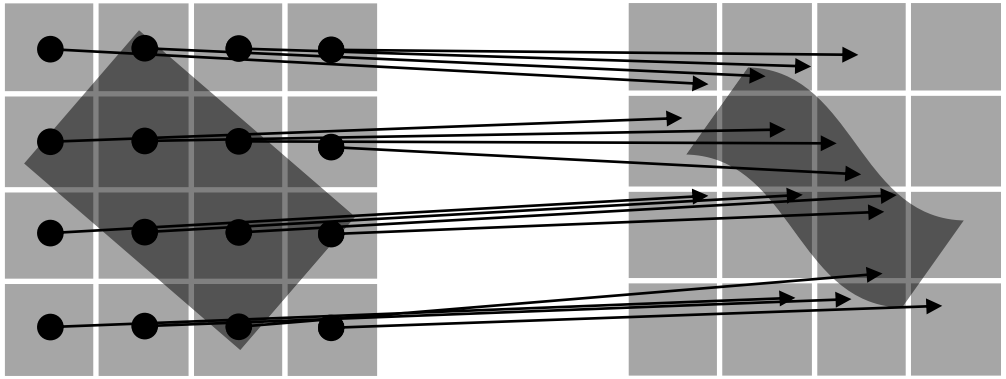


The three-component formulation

- **Regularised transformation models:** $\mathcal{T}\,(I_A, \theta)$, e.g. rigid, affine, TPS, FFD …
- **Image similarity measures:** $\mathcal{S}(I_A, I_B)$, e.g. SSD, CC, MI …
- **Optimisation:** $\hat{\theta} = \max_{\theta} \mathcal{S}[I_{Fixed}, \mathcal{T}(I_{Moving}, \theta)]$

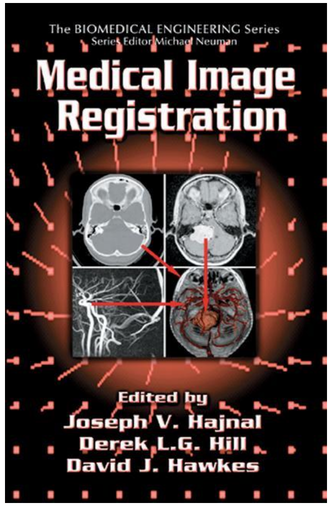
e.g. [Rueckert et al 1999] and two-decades research

- **Intensity- and Feature-based methods**

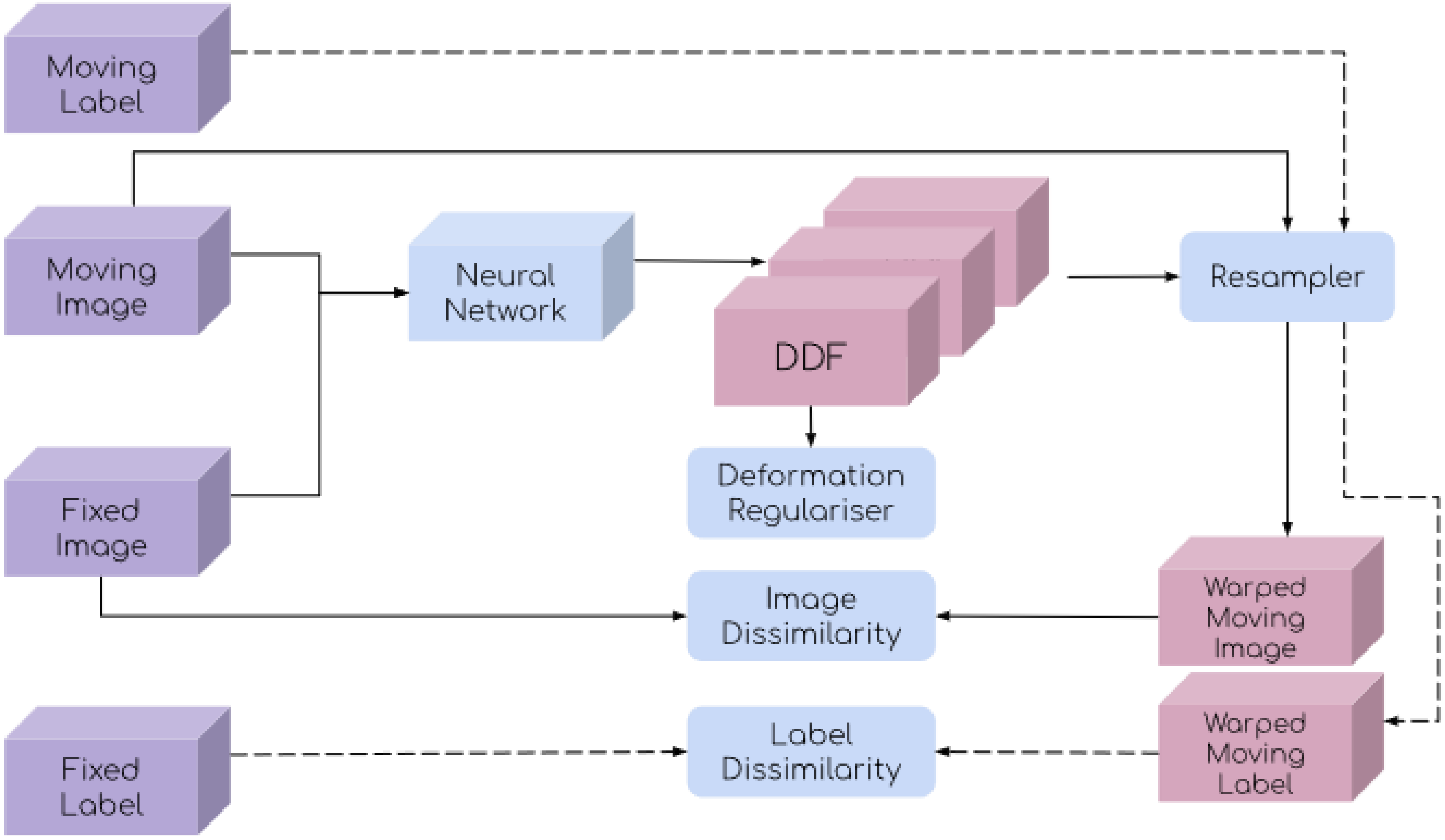


Moving Image

Fixed Image



Medical image registration using deep learning

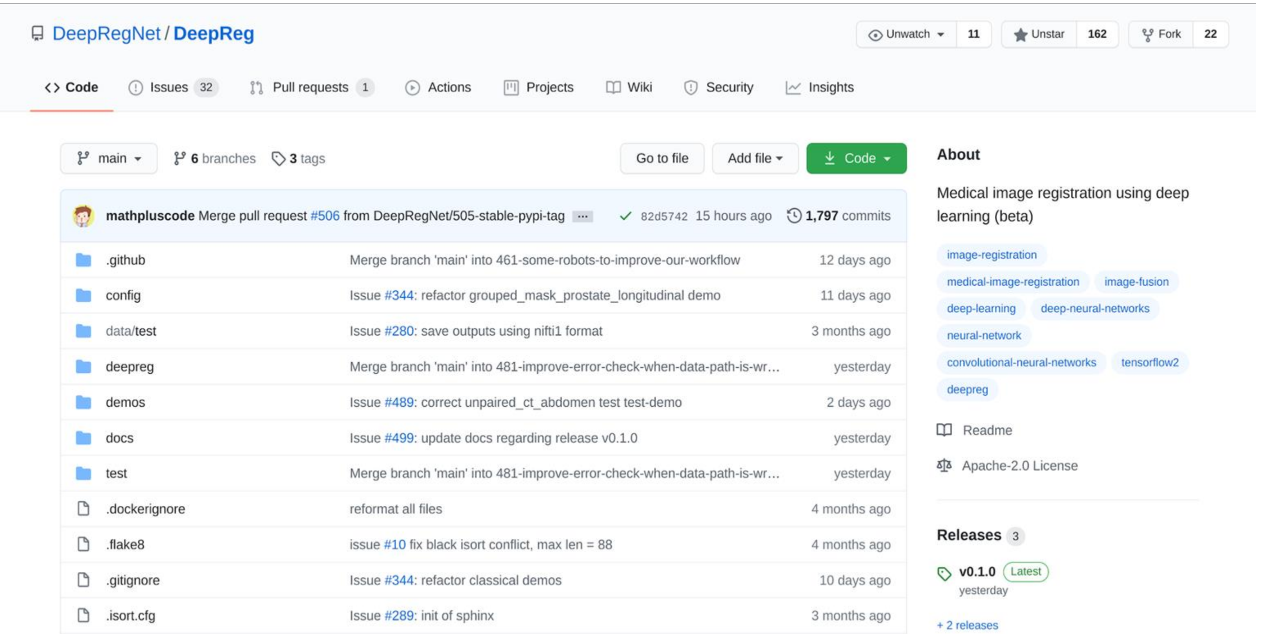


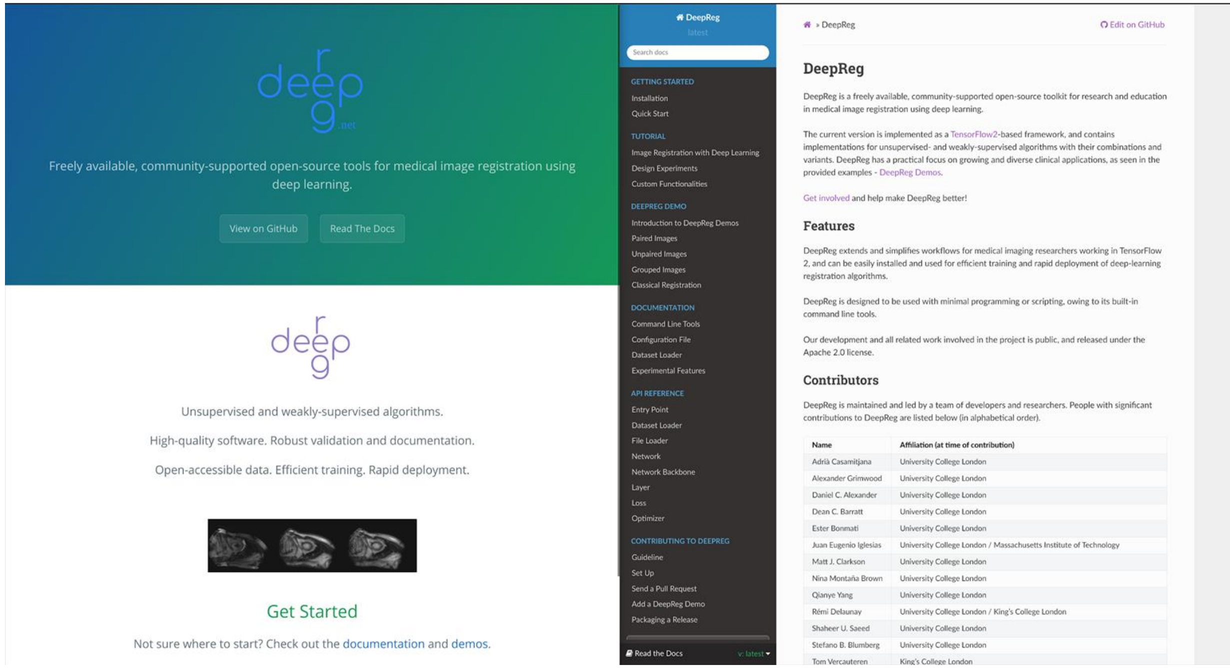
Freely available, community-supported open-source tools for medical image registration using deep learning

DeepReg.net

github.com/DeepRegNet/DeepReg

DeepReg.readthedocs.io





Readability – the original training.py script

```python
import tensorflow as tf
import sys
import random
import time

import labelreg.helpers as helper
import labelreg.networks as network
import labelreg.utils as util
import labelreg.losses as loss


# 0 - get configs
config = helper.ConfigParser(sys.argv, 'training')

# 1 - data
reader_moving_image, reader_fixed_image, reader_moving_label, reader_fixed_label = helper.get_data_readers(
    config['Data']['dir_moving_image'],
    config['Data']['dir_fixed_image'],
    config['Data']['dir_moving_label'],
    config['Data']['dir_fixed_label'])


# 2 - graph
ph_moving_image = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+reader_moving_image.data_shape+[1])
ph_fixed_image = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+reader_fixed_image.data_shape+[1])
ph_moving_affine = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+[1, 12])
ph_fixed_affine = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+[1, 12])
input_moving_image = util.warp_image_affine(ph_moving_image, ph_moving_affine)  # data augmentation
input_fixed_image = util.warp_image_affine(ph_fixed_image, ph_fixed_affine)  # data augmentation

# predicting ddf
reg_net = network.build_network(network_type=config['Network']['network_type'],
                                minibatch_size=config['Train']['minibatch_size'],
                                image_moving=input_moving_image,
                                image_fixed=input_fixed_image)
```

```python
# loss
ph_moving_label = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+reader_moving_image.data_shape+[1])
ph_fixed_label = tf.placeholder(tf.float32, [config['Train']['minibatch_size']]+reader_fixed_image.data_shape+[1])
input_moving_label = util.warp_image_affine(ph_moving_label, ph_moving_affine)  # data augmentation
input_fixed_label = util.warp_image_affine(ph_fixed_label, ph_fixed_affine)  # data augmentation

warped_moving_label = reg_net.warp_image(input_moving_label)  # warp the moving label with the predicted ddf

loss_similarity, loss_regulariser = loss.build_loss(similarity_type=config['Loss']['similarity_type'],
                                                    similarity_scales=config['Loss']['similarity_scales'],
                                                    regulariser_type=config['Loss']['regulariser_type'],
                                                    regulariser_weight=config['Loss']['regulariser_weight'],
                                                    label_moving=warped_moving_label,
                                                    label_fixed=input_fixed_label,
                                                    network_type=config['Network']['network_type'],
                                                    ddf=reg_net.ddf)

train_op = tf.train.AdamOptimizer(config['Train']['learning_rate']).minimize(loss_similarity+loss_regulariser)

# utility nodes - for information only
dice = util.compute_binary_dice(warped_moving_label, input_fixed_label)
dist = util.compute_centroid_distance(warped_moving_label, input_fixed_label)
```

```python
# 3 - training
num_minibatch = int(reader_moving_label.num_data/config['Train']['minibatch_size'])
train_indices = [i for i in range(reader_moving_label.num_data)]

saver = tf.train.Saver(max_to_keep=1)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for step in range(config['Train']['total_iterations']):

    if step in range(0, config['Train']['total_iterations'], num_minibatch):
        random.shuffle(train_indices)

    minibatch_idx = step % num_minibatch
    case_indices = train_indices[
                   minibatch_idx*config['Train']['minibatch_size']:(minibatch_idx+1)*config['Train']['minibatch_size']]
    label_indices = [random.randrange(reader_moving_label.num_labels[i]) for i in case_indices]

    trainFeed = {ph_moving_image: reader_moving_image.get_data(case_indices),
                 ph_fixed_image: reader_fixed_image.get_data(case_indices),
                 ph_moving_label: reader_moving_label.get_data(case_indices, label_indices),
                 ph_fixed_label: reader_fixed_label.get_data(case_indices, label_indices),
                 ph_moving_affine: helper.random_transform_generator(config['Train']['minibatch_size']),
                 ph_fixed_affine: helper.random_transform_generator(config['Train']['minibatch_size'])}

    sess.run(train_op, feed_dict=trainFeed)
```

Readability – the original training.py script

```python
import tensorflow as tf


def build_loss(similarity_type, similarity_scales, regulariser_type, regulariser_weight,
               label_moving, label_fixed, network_type, ddf):
    label_similarity = multi_scale_loss(label_fixed, label_moving, similarity_type.lower(), similarity_scales)
    if network_type.lower() == 'global':
        ddf_regularisation = tf.constant(0.0)
    else:
        ddf_regularisation = tf.reduce_mean(local_displacement_energy(ddf, regulariser_type, regulariser_weight))
    return tf.reduce_mean(label_similarity), ddf_regularisation


def weighted_binary_cross_entropy(ts, ps, pw=1, eps=1e-6):
    ps = tf.clip_by_value(ps, eps, 1-eps)
    return -tf.reduce_sum(
        tf.concat([ts*pw, 1-ts], axis=4)*tf.log(tf.concat([ps, 1-ps], axis=4)),
        axis=4, keep_dims=True)


def dice_simple(ts, ps, eps_vol=1e-6):
    numerator = tf.reduce_sum(ts*ps, axis=[1, 2, 3, 4]) * 2
    denominator = tf.reduce_sum(ts, axis=[1, 2, 3, 4]) + tf.reduce_sum(ps, axis=[1, 2, 3, 4])+eps_vol
    return numerator/denominator
```

Readability – the DeepReg training.py script

```python
import argparse
from datetime import datetime

from deepreg.train import train

name = "paired_mrus_prostate"


# parser is used to simplify testing
# please run the script with --full flag to ensure non-testing mode
# for instance:
# python script.py --full
parser = argparse.ArgumentParser()
parser.add_argument(
    "--test",
    help="Execute the script with reduced image size for test purpose.",
    dest="test",
    action="store_true",
)
parser.add_argument(
    "--full",
    help="Execute the script with full configuration.",
    dest="test",
    action="store_false",
)
parser.set_defaults(test=True)
args = parser.parse_args()


print(
    "\n\n\n\n\n"
    "=========================================================\n"
    "The training can also be launched using the following command.\n"
    "deepreg_train --gpu '0' "
    f"--config_path demos/{name}/{name}.yaml "
    f"--log_dir demos/{name} "
    "--exp_name logs_train\n"
    "=========================================================\n"
    "\n\n\n\n\n"
)

log_dir = f"demos/{name}"
exp_name = "logs_train/" + datetime.now().strftime("%Y%m%d-%H%M%S")
config_path = [f"demos/{name}/{name}.yaml"]
if args.test:
    config_path.append("config/test/demo_paired.yaml")
```

```python
train(
    gpu="0",
    config_path=config_path,
    gpu_allow_growth=True,
    ckpt_path="",
    log_dir=log_dir,
    exp_name=exp_name,
)
```

Readability – the DeepReg training.py script

```python
train(
    gpu="0",
    config_path=config_path,
    gpu_allow_growth=True,
    ckpt_path="",
    log_dir=log_dir,
    exp_name=exp_name,
)
```

```yaml
dataset:
  train:
    dir:
      - demos/paired_mrus_prostate/dataset/part00
      - demos/paired_mrus_prostate/dataset/part01
      - demos/paired_mrus_prostate/dataset/part02
      - demos/paired_mrus_prostate/dataset/part03
      - demos/paired_mrus_prostate/dataset/part04
      - demos/paired_mrus_prostate/dataset/part05
      - demos/paired_mrus_prostate/dataset/part06
      - demos/paired_mrus_prostate/dataset/part07
    format: nifti
    labeled: true
  valid:
    dir:
      - demos/paired_mrus_prostate/dataset/part08
    format: nifti
    labeled: true
  test:
    dir:
      - demos/paired_mrus_prostate/dataset/part09
      - demos/paired_mrus_prostate/dataset/part10
    format: nifti
    labeled: true
  type: paired
  moving_image_shape: [69, 69, 64]
  fixed_image_shape: [47, 63, 44]
train:
  # define neural network structure
  method: "ddf" # the registration method, value should be ddf / dvf / conditional
  backbone:
    name: "local" # value should be local / global / unet
    num_channel_initial: 16 # number of initial channel in local net, controls the size of the network
    extract_levels: [0, 1, 2, 3]

  # define the loss function for training
  loss:
    image:
      name: "ssd"
      weight: 0
    label:
      weight: 1.0
      name: "dice"
      scales: [0, 1, 2, 4, 8, 16]
    regularization:
      weight: 0.125 # weight of regularization loss
      name: "bending" # options include "bending", "gradient"

  # define the optimizer
  optimizer:
    name: "Adam"
    learning_rate: 1.0e-5

  # define the hyper-parameters for preprocessing
  preprocess:
    data_augmentation:
      name: "affine"
    batch_size: 4
    shuffle_buffer_num_batch: 1 # shuffle_buffer_size = batch_size * shuffle_buffer_num_batch

  # other training hyper-parameters
  epochs: 5000 # number of training epochs
  save_period: 1000 # the model will be saved every `save_period` epochs.
```

...

Package or tool

-> For users who do not code

-> command line tool

-> custom config file

-> REGISTRY

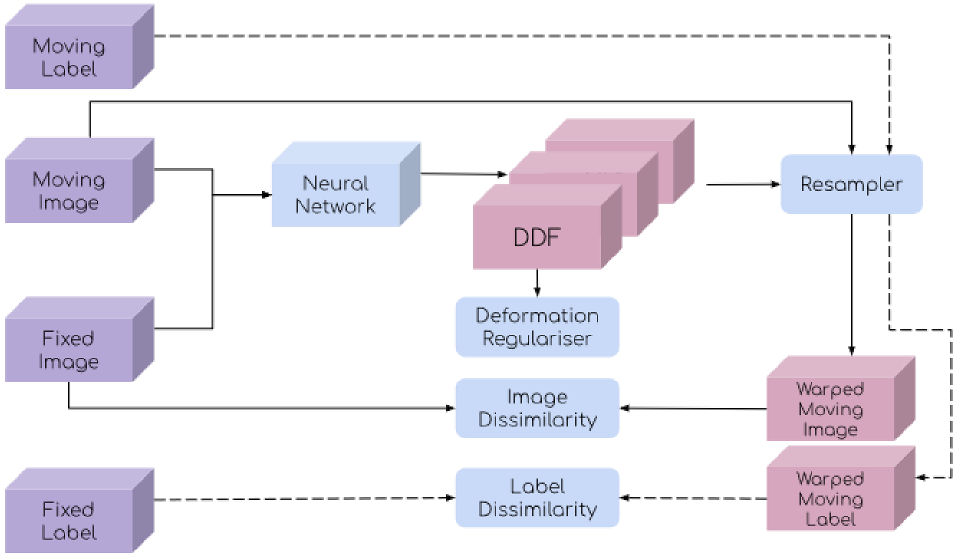
Feedback from developers, testers, and users

```
@REGISTRY.register_model(name="ddf")
class DDFModel(RegistrationModel):
    """
    A registration model predicts DDF.

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

    name = "DDFModel"

    def _resize_interpolate(self, field, control_points):
        resize = layer.ResizeCPTransform(control_points)
        field = resize(field)

        interpolate = layer.BSplines3DTransform(control_points, self.fixed_image_size)
        field = interpolate(field)

        return field

    def build_model(self):
        """Build the model to be saved as self._model."""
        # build inputs
        self._inputs = self.build_inputs()
        moving_image = self._inputs["moving_image"]  # (batch, m_dim1, m_dim2, m_dim3)
        fixed_image = self._inputs["fixed_image"]  # (batch, f_dim1, f_dim2, f_dim3)

        # build ddf
        control_points = self.config["backbone"].pop("control_points", False)
        backbone_inputs = self.concat_images(moving_image, fixed_image)
        backbone = REGISTRY.build_backbone(
            config=self.config["backbone"],
            default_args=dict(
                image_size=self.fixed_image_size,
                out_channels=3,
                out_kernel_initializer="zeros",
                out_activation=None,
            ),
        )
```

Pre-defined workflow

Data loader script – for a two-stage, image/label feeding

```python
class DataReader:

    def __init__(self, dir_name):
        self.dir_name = dir_name
        self.files = os.listdir(dir_name)
        self.files.sort()
        self.num_data = len(self.files)

        self.file_objects = [nib.load(os.path.join(dir_name, self.files[i])) for i in range(self.num_data)]
        self.num_labels = [self.file_objects[i].shape[3] if len(self.file_objects[i].shape) == 4
                           else 1
                           for i in range(self.num_data)]

        self.data_shape = list(self.file_objects[0].shape[0:3])

    def get_num_labels(self, case_indices):
        return [self.num_labels[i] for i in case_indices]

    def get_data(self, case_indices=None, label_indices=None):
        if case_indices is None:
            case_indices = range(self.num_data)
        # todo: check the supplied label_indices smaller than num_labels
        if label_indices is None:  # e.g. images only
            data = [np.asarray(self.file_objects[i].dataobj) for i in case_indices]
        else:
            if len(label_indices) == 1:
                label_indices *= self.num_data
            data = [self.file_objects[i].dataobj[..., j] if self.num_labels[i] > 1
                    else np.asarray(self.file_objects[i].dataobj)
                    for (i, j) in zip(case_indices, label_indices)]
        return np.expand_dims(np.stack(data, axis=0), axis=4)
```



Pre-defined workflow

DeepReg data loaders

Possibilities and permutations

Compromises

Readability

Configuration

An example configuration for paired dataset is provided as follows.

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

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

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

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

```
dataset:
  train:
    dir: # folders containing data
      - "data/test/h5/paired/train1"
      - "data/test/h5/paired/train2"
```

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

CLT user manual

(for a standalone software)

Required arguments

- **GPU**:

  `--gpu` or `-g`, specifies the index or indices of GPUs for training.

  Example usage:

  - `--gpu ""` for CPU only
  - `--gpu "0"` for using only GPU 0
  - `--gpu "0,1"` for using GPU 0 and 1.

- **Configuration**:

  `--config_path` or `-c`, specifies the configuration file for training.

  The path must end with `.yaml`.

  Optionally, multiple paths can be specified, and the configuration will be merged. In case of conflicts, values are overwritten by the last config file defining them.

  Example usage:

  - `--config_path config1.yaml` for using one single configuration file.
  - `--config_path config1.yaml config2.yaml` for using multiple configuration files.

Optional arguments

- **CPU allocation**:

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

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

  Example usage:

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

- **GPU memory allocation**:

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

  By default, it allocates all available GPU memory.

  Example usage:

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

- **Load checkpoint**:

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

Tutorials

(for research, education)

Cluster)

DEEPREG DEMO

- Introduction to DeepReg Demos
- Paired Images
- Unpaired Images
- Grouped Images
- Classical Registration

DOCUMENTATION

- Command Line Tools
- Logging
- Configuration File
- Dataset Loader
- Registry
- Experimental Features

API REFERENCE

- Entry Point
- Dataset Loader
- File Loader
- Registry
- Network
- Backbone
- Layer
- Loss
- Optimizer

CONTRIBUTING TO DEEPREG

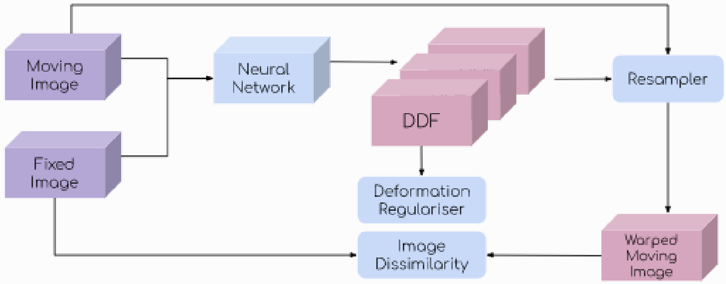
- Guidelines
- Unit Test
- DeepReg demo
- Documentation
- Release

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).

Tutorials

(for research, education)

1st Place Winner in MICCAI Educational Challenge 2020

MEC 2020 Materials

View Teaser Videos For 2020 Finalists

- **1st Place!** Introduction to Medical Image Registration with DeepReg, Between Old and New (Teaser Video)
  Nina Montana Brown, Yunguan Fu, Shaheer Saeed, Adrià Casamitjana, Zachary Baum, Rémi Delaunay, Qianye Yang, Alexander Grimwood, Zhe Min, Ester Bonmati, Tom Vercauteren, Matthew J. Clarkson and Yipeng Hu - University College London, InstaDeep & King's College London
- **2nd Place!** An Introduction to SimpleITK (Teaser Video)
  Ziv Yaniv, Bradley C. Lowekamp and David T. Chen - National Institute of Allergy and Infectious Diseases, National Institutes of Health & Medical Science and Computing LLC.
- **3rd Place!** How to Organize a Challenge –- Behind the Scenes (Teaser Video)
  Annika Reinke, Matthias Eisenmann, Laura Aguilera Saiz, Sinan Onogur, Leonardo Antonio Ayala Menjivar and Peter Maximilian Full - German Cancer Research Center (DKFZ) & University of Heidelberg
- **Finalist** Guitar and Musical MRI Concepts (Part 2) (Part 3) (Teaser Video)
  Jerry J. Battista - Western University
- **Finalist** Hands on Machine Learning Training -- HaMLeT (Teaser Video)
  Leon Weninger, Laxmi Gupta and Philipp Gräbel - RWTH Aachen University
- **Finalist** The Basic Augmented Reality Demo -- BARD (Teaser Video)
  Matt Clarkson, Steve Thompson, Ester Bonmati, Tom Dowrick, Yipeng Hu and Ann Blandford - University College of London

Intro_to_Medical_Image_Regsitration.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text Copy to Drive

Table of contents

Introduction to Medical Image Registration with DeepReg, between Old and New

Authors:

- Nina Montana Brown [1, 2]
- Yunguan Fu [1, 2, 3]
- Shaheer Saeed [1, 2]
- Adrià Casamitjana [2]
- Zachary Baum [1, 2]
- Rémi Delaunay [1, 2, 4]
- Qianye Yang [1, 2]
- Alexander Grimwood [1, 2]
- Zhe Min [1]
- Ester Bonmati [1, 2]
- Tom Vercauteren [4]
- Matthew J. Clarkson [1, 2]
- Yipeng Hu [1, 2]

Affiliations:

- [1] Wellcome/EPSRC Centre for Surgical and Interventional Sciences, University College London
- [2] Centre for Medical Image Computing, University College London
- [3] InstaDeep
- [4] Department of Surgical & Interventional Engineering, King's College London

Table of Contents

Objective of the Tutorial

This tutorial introduces a new open-source project DeepReg, currently based on the latest release of TensorFlow 2. This package is designed to accelerate research in image registration using parallel computing and deep learning by providing simple, tested entry points to pre-designed networks for users to get a head start with. Additionally, DeepReg provides more basic functionalities such as custom TensorFlow layers which allow more seasoned researchers to build more complex functionalities.

A previous MICCAI workshop learn2reg provided an excelent example of novel algorithms and interesting approaches in this active research area, whilst this tutorial explores the strength of the simple, yet generalisable design of DeepReg.

- Explain basic concepts in medical image registration;
- Explore the links between the modern algorithms using neural networks and the classical iterative algorithms (also using DeepReg);
- Introduce the versatile capability of DeepReg, with diverse examples in real clinical challenges.

Since DeepReg has a pre-packaged command line interface, minimum scripting and coding expereince with DeepReg is necessary to follow along this tutorial. Accompanying the tutorial is the DeepReg documentation and a growing number of demos using real, open-accesible clinical data. This tutorial will get you started with DeepReg by illustrating a number of examples with step-by-step instructions.

Set-up

This tutorial depends on the package DeepReg, which in turn has external dependencies which are managed by `pip`. The current version is implemented as a TensorFlow 2 and Python>=3.7.

Papers

(for research)

The Journal of Open Source Software

DeepReg: a deep learning toolkit for medical image registration

Yunguan Fu[1, 2, 3], Nina Montaña Brown[1, 2], Shaheer U. Saeed[1, 2], Adrià Casamitjana[2], Zachary M. C. Baum[1, 2], Rémi Delaunay[1, 4], Qianye Yang[1, 2], Alexander Grimwood[1, 2], Zhe Min[1], Stefano B. Blumberg[2], Juan Eugenio Iglesias[2, 5, 6], Dean C. Barratt[1, 2], Ester Bonmati[1, 2], Daniel C. Alexander[2], Matthew J. Clarkson[1, 2], Tom Vercauteren[4], and Yipeng Hu[1, 2]

1 Wellcome/EPSRC Centre for Surgical and Interventional Sciences, University College London, London, UK **2** Centre for Medical Image Computing, University College London, London, UK **3** InstaDeep, London, UK **4** Department of Surgical & Interventional Engineering, King's College London, London, UK **5** Martinos Center for Biomedical Imaging, Massachusetts General Hospital and Harvard Medical School, Boston, USA **6** Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Boston, USA

DOI: 10.21105/joss.02705

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Summary

Image fusion is a fundamental task in medical image analysis and computer-assisted intervention. Medical image registration, computational algorithms that align different images together (Hill, Batchelor, Holden, & Hawkes, 2001), has in recent years turned the research attention towards deep learning. Indeed, the representation ability to learn from population

Docstring, API documentation

» Entry Point Edit on GitHub

Entry Point

Train

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

`deepreg.train.build_config`*(config_path: Union[str, List[str]], log_dir: str, exp_name: str, ckpt_path: str, max_epochs: int = - 1)* → Tuple[Dict, str, str]

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

Parameters:
- **config_path** – list of str, path to config file
- **log_dir** – path of the log directory
- **exp_name** – name of the experiment
- **ckpt_path** – path where model is stored.
- **max_epochs** – if max_epochs > 0, use it to overwrite the configuration

Returns:
- config: a dictionary saving configuration
- exp_name: the path of directory to save logs

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

Entry point for train script.

Parameters: **args** – arguments

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

Function to train a model.

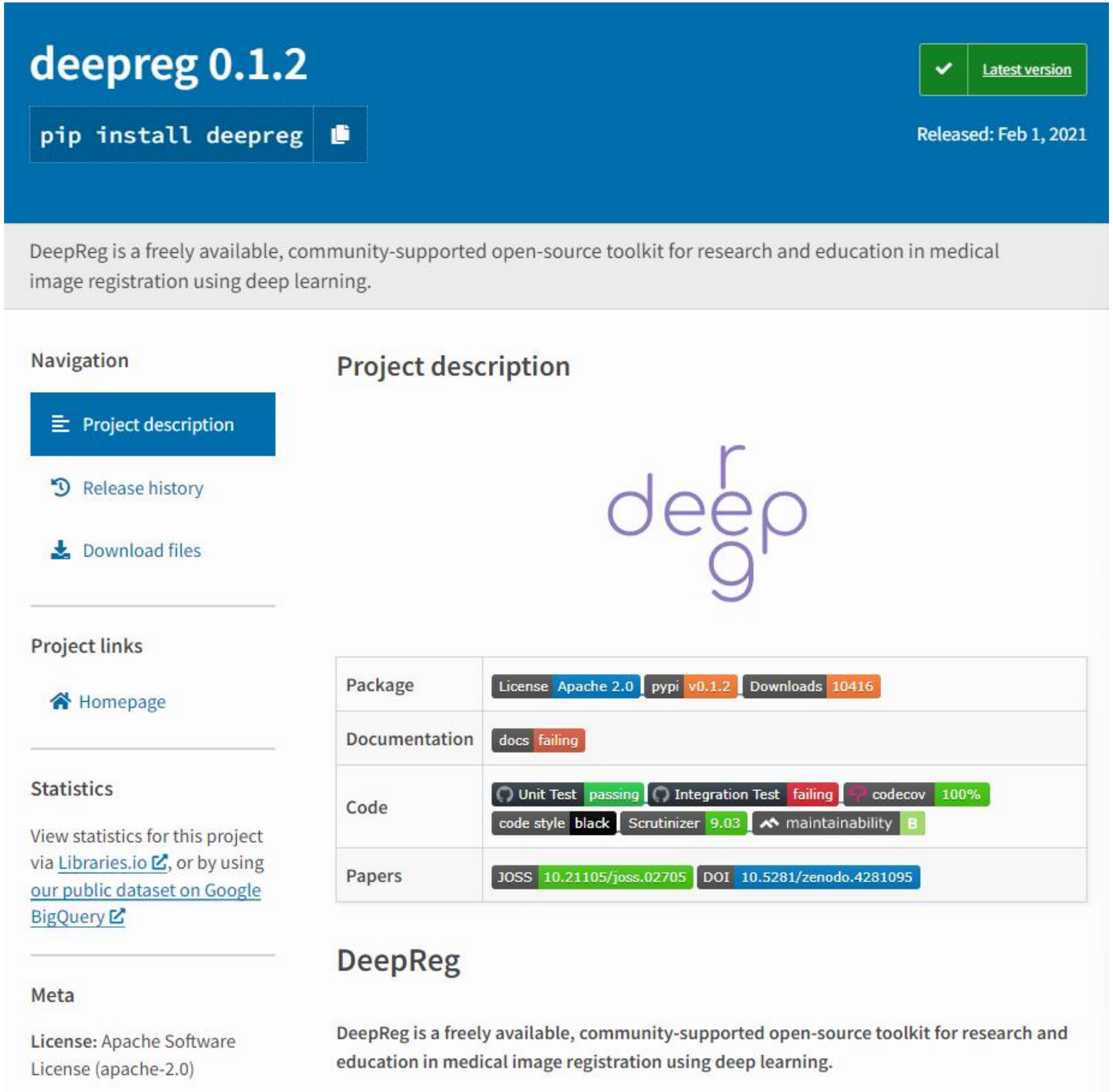
Parameters:
- **gpu** – which local gpu to use to train.
- **config_path** – path to configuration set up.
- **ckpt_path** – where to store training checkpoints.
- **num_workers** – number of cpu cores to be used.

Installation

Conda

PYPI packaging

Docker



deepreg 0.1.2

```
pip install deepreg
```

Latest version

Released: Feb 1, 2021

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

Navigation

Project description

Release history

Download files

Project links

Homepage

Statistics

View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery

Meta

License: Apache Software License (apache-2.0)

Project description

Package	License Apache 2.0 · pypi v0.1.2 · Downloads 10416
Documentation	docs failing
Code	Unit Test passing · Integration Test failing · codecov 100% · code style black · Scrutinizer 9.03 · maintainability B
Papers	JOSS 10.21105/joss.02705 · DOI 10.5281/zenodo.4281095

DeepReg

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

DeepReg Demos

Reference

Paired brain MR-ultrasound registration

Paired prostate MR-ultrasound registration

Unpaired Images

Grouped Images

Classical Registration

DOCUMENTATION

Command Line Tools

Logging

Configuration File

Dataset Loader

Registry

Experimental Features

API REFERENCE

Entry Point

Dataset Loader

File Loader

Registry

Network

Backbone

Layer

Loss

Optimizer

CONTRIBUTING TO DEEPREG

Guidelines

Unit Test

DeepReg demo

Documentation

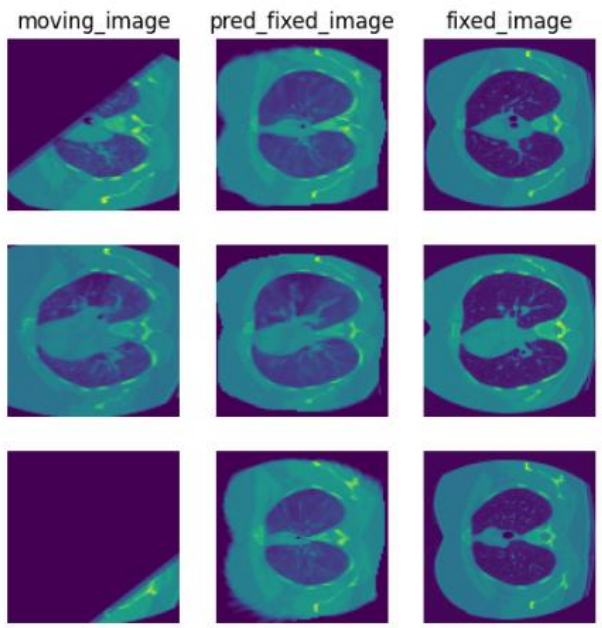
Release

Visualise

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

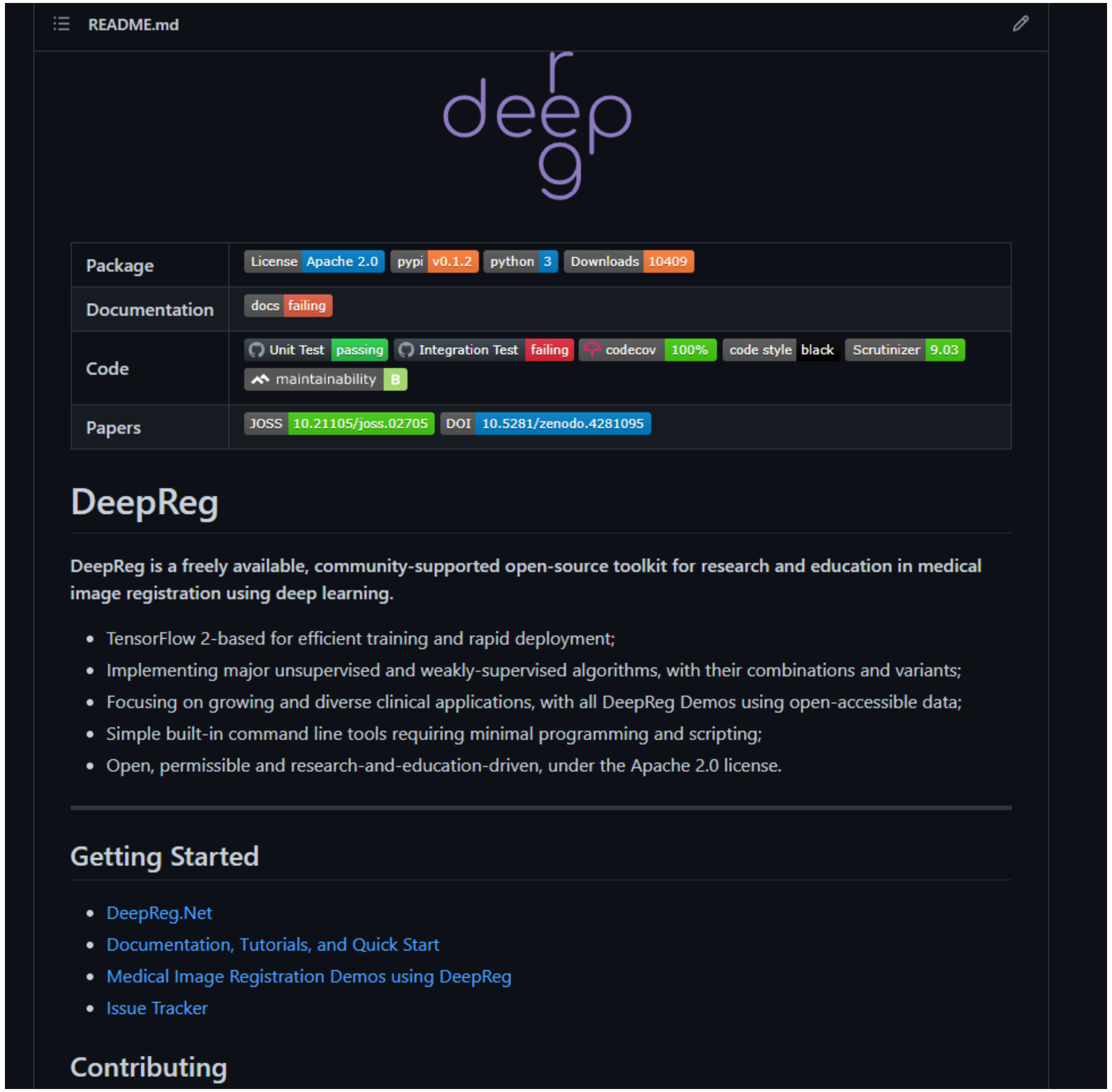
```
deepreg_vis -m 2 -i 'demos/paired_ct_lung/logs_predict/<time-stamp>/test/<pair-
```

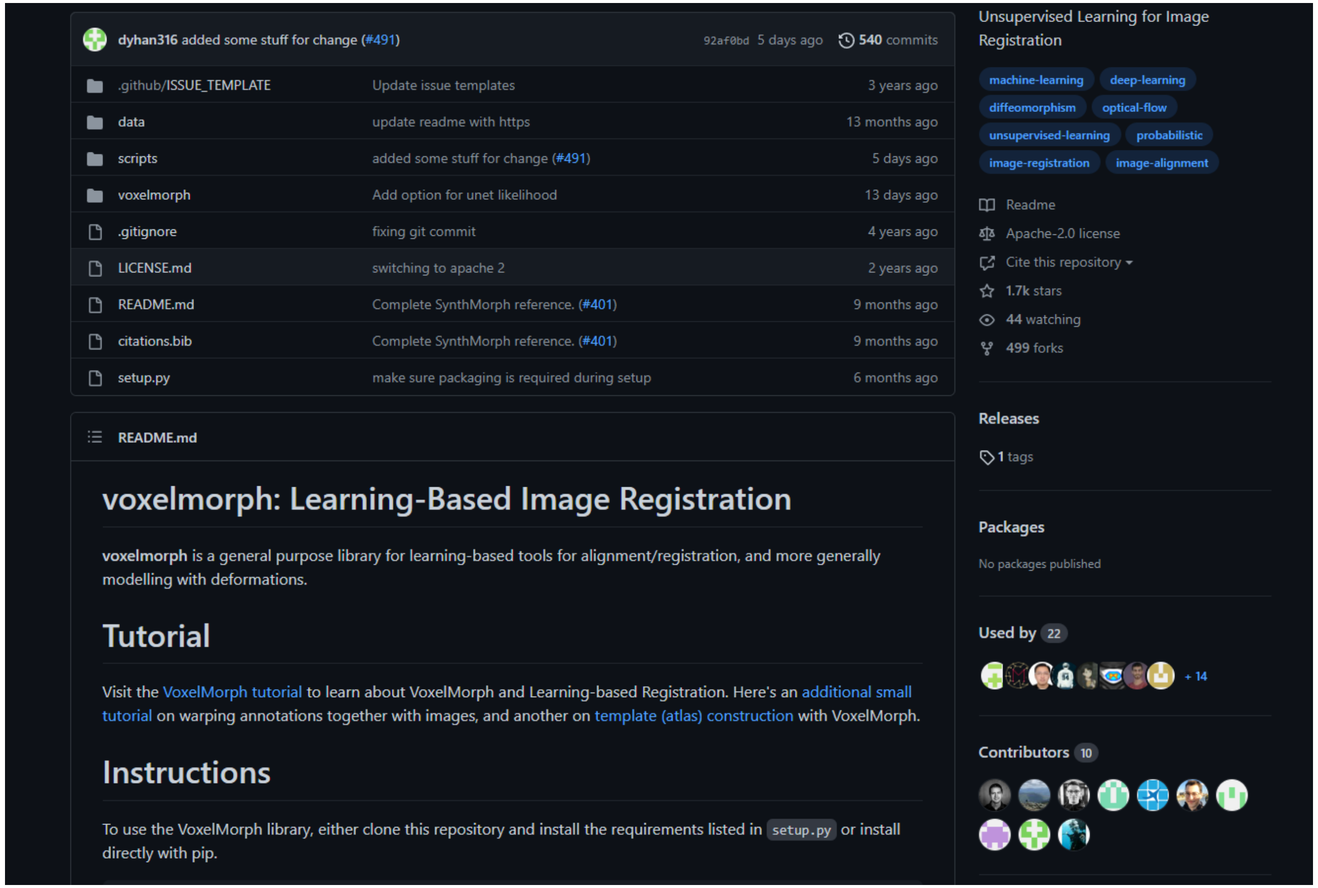
Note: The prediction must be run before running the command to generate the visualisation. The `<time-stamp>` and `<pair-number>` must be entered by the user.



Contact

Tests


README.md
deep reg
Package
License Apache 2.0
pypi v0.1.2
python 3
Downloads 10409
Documentation
docs failing
Code
Unit Test passing
Integration Test failing
codecov 100%
code style black
Scrutinizer 9.03
maintainability B
Papers
JOSS 10.21105/joss.02705
DOI 10.5281/zenodo.4281095
DeepReg
DeepReg is a freely available, community-supported open-source toolkit for research and education in medical image registration using deep learning.
TensorFlow 2-based for efficient training and rapid deployment;
Implementing major unsupervised and weakly-supervised algorithms, with their combinations and variants;
Focusing on growing and diverse clinical applications, with all DeepReg Demos using open-accessible data;
Simple built-in command line tools requiring minimal programming and scripting;
Open, permissible and research-and-education-driven, under the Apache 2.0 license.
Getting Started
DeepReg.Net
Documentation, Tutorials, and Quick Start
Medical Image Registration Demos using DeepReg
Issue Tracker
Contributing


dyhan316 added some stuff for change (#491)
92af0bd 5 days ago
540 commits
.github/ISSUE_TEMPLATE
Update issue templates
3 years ago
data
update readme with https
13 months ago
scripts
added some stuff for change (#491)
5 days ago
voxelmorph
Add option for unet likelihood
13 days ago
.gitignore
fixing git commit
4 years ago
LICENSE.md
switching to apache 2
2 years ago
README.md
Complete SynthMorph reference. (#401)
9 months ago
citations.bib
Complete SynthMorph reference. (#401)
9 months ago
setup.py
make sure packaging is required during setup
6 months ago
README.md
voxelmorph: Learning-Based Image Registration
voxelmorph is a general purpose library for learning-based tools for alignment/registration, and more generally modelling with deformations.
Tutorial
Visit the VoxelMorph tutorial to learn about VoxelMorph and Learning-based Registration. Here's an additional small tutorial on warping annotations together with images, and another on template (atlas) construction with VoxelMorph.
Instructions
To use the VoxelMorph library, either clone this repository and install the requirements listed in setup.py or install directly with pip.
Unsupervised Learning for Image Registration
machine-learning
deep-learning
diffeomorphism
optical-flow
unsupervised-learning
probabilistic
image-registration
image-alignment
Readme
Apache-2.0 license
Cite this repository
1.7k stars
44 watching
499 forks
Releases
1 tags
Packages
No packages published
Used by 22
+ 14
Contributors 10

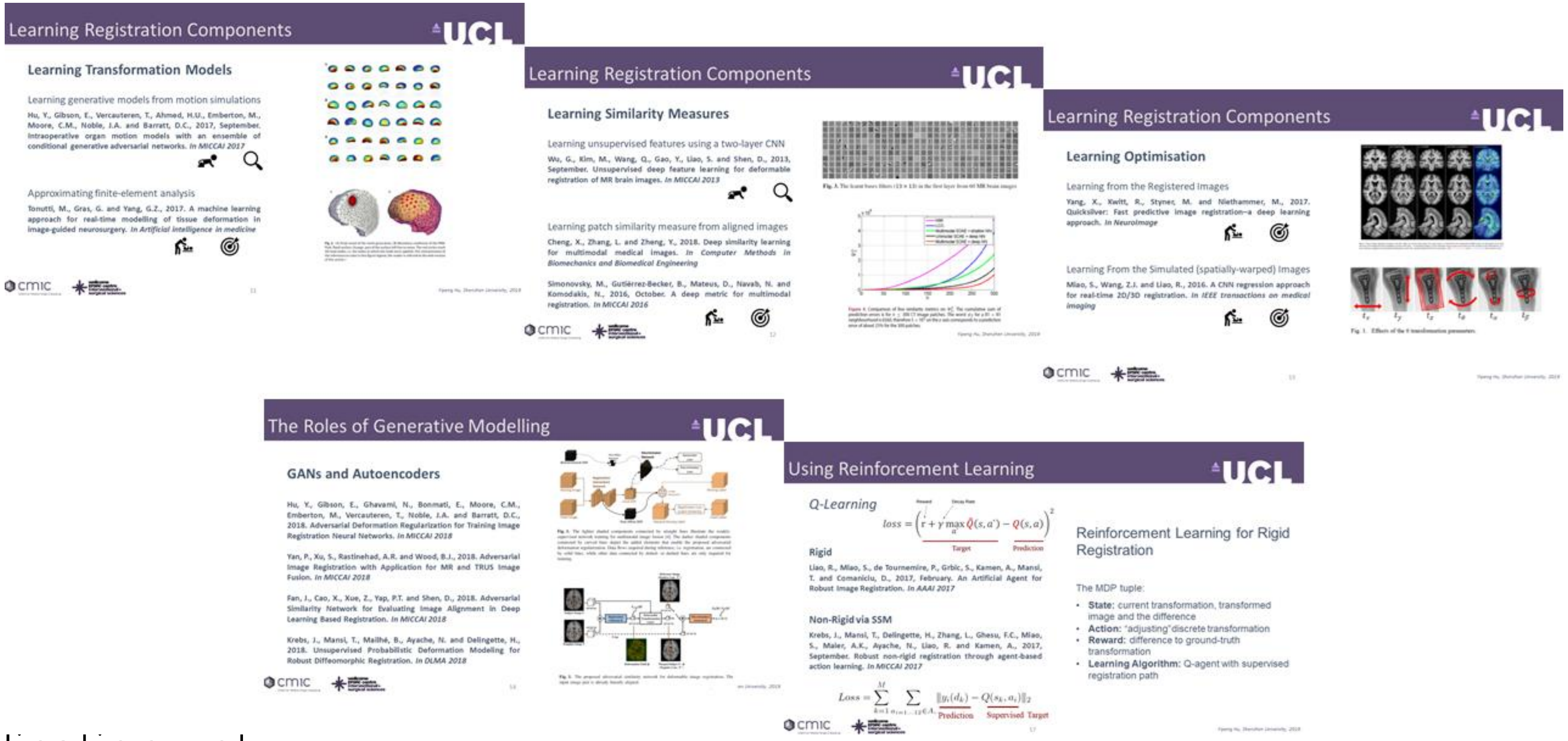
Promotion

We have no clue…!

Trained models

Open data sets, e.g. learn2reg challenges

Clinical data pre-processing and curation



Plenty research, no clinical application yet

As is and achieved?

Next interested PhD students?

MONAI?

Paper-based? e.g.

Yang 2019 MICCAI (DeepReg branch)

Yang 2021 ISBI (using DeepReg)

Yang 2022 TMI (PyTorch)

Baum 2021 MedIA (standalone repo)

Baum 2022 ASMUS (using DeepReg)

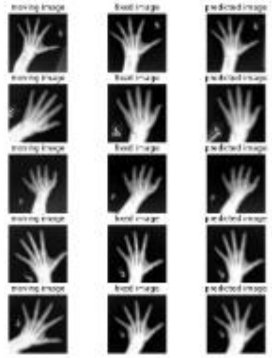
Baum 2022 TMI (using DeepReg)

Shen 2022 MICCAI (PyTorch)

UCL Modules? e.g.

MPHY0041 (DeepReg code)

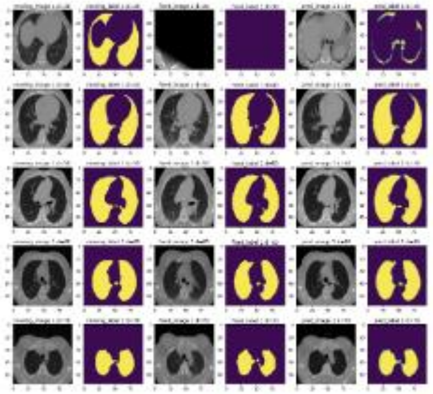
MPHY0043 (MONAI code)



3D intra-subject lung CT registration

The 3D intra-subject lung CT registration tutorial is an example of registration between 3D lung CT images acquired at inspiration and expiration from a single patient. This type of intra-subject registration is helpful in tracking anatomical features of interest like airways in airflow analysis or compensating motion during radiotherapy.

The tutorial showcases several features described above, including unsupervised and weakly-supervised losses, deformation regulariser, non-rigid transformation based on DDFs, and 3D volumetric registration using real clinical images. To learn more, check out the provided notebook and the original DeepReg demo using the same open-accessible data set.



MONAI Medical Open Network for AI Follow

Apr 23, 2021 · 4 min read · Listen

Monai Explores Learning-Based Medical Image Registration

MONAI has been working closely with DeepReg on learning-based medical image registration using PyTorch. In the latest release, MONAI v0.5.0, we are delighted to provide a set of essential tools for developing registration pipelines.

Medical Image Registration

Image registration, a process that spatially aligns one image with another, is important to many medical imaging applications. The two images that register are often referred to as moving and fixed images. An image registration algorithm generates a spatial transformation that can transform, or "warp", the moving image onto the fixed image coordinates.

The ability to align two or more images together allows the complementary information acquired from different imag (multimodal registration, e.g., MR and ultrasound), at diffe (longitudinal registration), or from different patients (inter registration). For example, track 33 owth from a