GraDIRN: Learning Iterative Gradient Descent-based Energy Minimization for Deformable Image Registration

Huaqi Qiu
huaiqi.qiu15@imperial.ac.uk

Kerstin Hammernik
k.hammernik@imperial.ac.uk

Chen Qin
chen.qin@ed.ac.uk

Daniel Rueckert
d.rueckert@imperial.ac.uk

1BioMedIA, Imperial College London
2University of Edinburgh
3Technical University Munich

Abstract

We present a Gradient Descent-based Image Registration Network (GraDIRN) for learning deformable image registration by embedding gradient-based iterative energy minimization in a deep learning framework. Traditional image registration algorithms typically use iterative energy-minimization optimization to find the optimal transformation between a pair of images, which is time-consuming when many iterations are needed. In contrast, recent learning-based methods amortize this costly iterative optimization by training deep neural networks so that registration of one pair of images can be achieved by fast network forward pass after training. Motivated by successes in image reconstruction techniques that combine deep learning with the mathematical structure of iterative variational energy optimization, we formulate a novel registration network based on multi-resolution gradient descent energy minimization. The forward pass of the network takes explicit image dissimilarity gradient steps and generalized regularization steps parameterized by Convolutional Neural Networks (CNN) for a fixed number of iterations. We use auto-differentiation to derive the forward computational graph for the explicit image dissimilarity gradient w.r.t. the transformation, so arbitrary image dissimilarity metrics and transformation models can be used without complex and error-prone gradient derivations. We demonstrate that this approach achieves state-of-the-art registration performance while using fewer learnable parameters through extensive evaluations on registration tasks using 2D cardiac MR images and 3D brain MR images.

1. Introduction

Image registration is a task of finding the spatial alignment between matching structures from different images. Especially in medical imaging, it is one of the most fundamental tasks for applications such as the fusion of different images of the same patient (e.g. MR-CT or MR-PET), surgical intervention, cardiac motion analysis [6] and disease progression analysis [21].

Formally, registration of a pair of moving and fixed images $(I_m, I_f)$ aims to find a transformation $\phi$ that can align the images. Many traditional registration methods obtain this transformation by minimizing an energy functional $E(I_m, I_f, \phi)$ using iterative optimization algorithms for each pair of images [40]. The energy functional usually consists of a data-fidelity term and a regularization term. The data-fidelity term measures the alignment between the reference fixed image $I_f$ and the transformed moving image $I_m' = T(I_m, \phi)$ using (dis)-similarity metrics such as Sum of Squared Difference (SSD), Cross-Correlation (CC) or Mutual Information (MI). The regularization term imposes constraints on the property of the transformation such as smoothness and invertibility. The transformation model that describes $\phi$ can be rigid or affine transformations, which are useful for global alignment of images; or deformable transformation models with more degrees-of-freedom which can be modeled using lower-dimensional parameterization such as B-spline free-form-deformation [35] or non-parametric high-dimensional models such as dense displacement field or velocity field for diffeomorphic registration [1, 43].

However, iterative optimization techniques typically require many iteration steps and a significant amount of computational time, especially for high-dimensional transformation models. To circumvent this problem, data-driven
methods have been proposed recently to substantially speed up the registration process by training deep neural networks on groups of training data sets [5,45]. After training, the registration of a new image pair can be achieved by one fast forward pass of the network. To train the networks, early methods relied on transformation parameters derived from traditional registration tools or randomly generated transformation fields as ground truth and thus formulate the training as a supervised regression task [34,39,45]. This coincides with advances in optical flow estimation using deep learning, where large simulated data sets with ground truth optical flow became available [11,23,41]. To avoid dependency on ground truth deformations, other approaches rely on dissimilarity metrics based on image intensities [4,9,31] or semantic segmentation [5,20,29] as self-supervision signal for training. This is made possible by the differentiable spatial transformation module introduced in [24].

Intuitively, these DL registration methods use iterative optimization in the training process to amortize the iterative optimization in traditional registration, so registration becomes a one-step inference of the network once trained. Some recent advances in deep learning registration re-introduce some elements of iteration by extending the single-network approach (represented by [4]) via composition of predictions from multiple networks on single or multiple resolutions [19,30,32,47], or using a recurrent refinement network [36].

Meanwhile, several approaches based on the concept of unrolled optimization networks have been shown successful for inverse problems such as image reconstruction and image de-noising [16,27]. These approaches embed iterative optimization in a learning-based setting by designing networks that follow the mathematical structure of classical iterative optimization algorithms. While the optimization steps in the solution space are parameterized by more complex learned components such as Field-of-Expert model [8] or neural networks [14,15,37], explicit data-fidelity terms are used as the knowledge of maximizing data likelihood from the data space. The idea of learned unrolled optimization has been hardly explored in image registration. We are only aware of the work in [25], who uses a single-resolution optimization scheme based on an auxiliary variable with the linearization of the brightness consistency data-fidelity [12,46].

In this work, we design a Gradient Descent-based Image Registration Network (GraDIRN) for deformable image registration which unrolls a multi-resolution gradient descent algorithm to optimize the spatial transformation \( \phi \) by minimizing an energy functional. GraDIRN uses explicit gradient steps of the data-fidelity term and generalized regularization gradient steps parameterized by CNNs. The parameters of the CNNs and gradient step sizes are learned based on training data. In contrast to [25], the proposed method is based on a more parameter and memory-efficient multi-resolution gradient-based optimization. This enables us to incorporate the data-fidelity term in the network inference by automatically deriving the explicit gradient using auto-differentiation, making the use of various image dissimilarity metrics and transformation models in our framework considerably easier.

Our main contributions in this work are the following:

- We introduce a novel Gradient Descent-based Image Registration Network (GraDIRN) for deformable image registration, which is based on the mathematical structure of a multi-resolution gradient descent energy minimization.

- We use auto-differentiation to derive the computational graph of the explicit data-fidelity gradient steps. While avoiding error-prone manual derivation, this paves the way for using arbitrary data similarity functions and transformation models for image registration.

- We extensively investigate the efficacy of our framework, including the significance of the explicit data-fidelity and parameter efficiency, by ablation studies where variations of our method and baseline methods are evaluated.

- We demonstrate state-of-the-art performance for challenging registration tasks including 2D cardiac registration and 3D brain registration while being efficient in the number of learned parameters.

2. Method

2.1. Bi-level Optimization Formulation

**Overview** We formulate the training of the proposed Gradient Descent-based Image Registration Network (GraDIRN) for image registration under a bi-level optimization view. Mathematically, this can be written as:

\[
\min_{\theta_\hat{\phi}} \mathcal{L}(I_m, I_f, \hat{\phi}(\theta_\hat{\phi})) \\
\text{s.t. } \hat{\phi}(\theta_\hat{\phi}) \in \arg \min_{\phi} E(I_m, I_f, \phi, \theta_1)
\]

where \( \hat{\phi}(\theta_\hat{\phi}) \) is the solution to the lower-level optimization problem \( \arg \min_{\phi} E(I_m, I_f, \phi, \theta_1) \), which is an energy minimization problem that finds the optimal transformation \( \hat{\phi} \) depending on the trainable parameters \( \theta_1 \). The higher-level optimization problem finds these parameters by minimizing the loss function \( \mathcal{L}(I_m, I_f, \hat{\phi}(\theta_\hat{\phi})) \).

**Lower-level optimization** Under this view, the forward pass of the network, which contains the learnable parameters \( \theta_1 \), is the lower-level optimization. Specifically, the
Resolution 1 Resolution n Resolution N
GD 1 ... GD t ... ... GD T ...

(a) Lower-level optimization (inference)

Resolution 1 Resolution n Resolution N
GD 1 ... GD t ... ... GD T ...

(b) Higher-level optimization (training)

Figure 1. Illustration of the bi-level optimization framework. Fig. 1a shows the inference of GraDIRN as the lower-level optimization, which is an iterative multi-resolution gradient descent using explicit data-fidelity gradient steps and regularization steps parameterized by CNN. Starting from an identity transformation $\phi_0$ on the coarsest resolution, the optimization takes $T_n$ step in each resolution before reaching the final solution $\phi_T$. The blue blocks shows the inner structure of each gradient descent step. Fig. 1b shows the training process, or the higher-level optimization, which learns the parameters of the network using a loss function $L$ on the output of GraDIRN. The structure shown in Fig. 1a is illustrated in the dotted box.

Lower-level optimization aims to find the optimal transformation $\phi$ that minimizes the energy functional $E$. The energy function consists of a data-fidelity term $D$ which measures the dissimilarity between the registered images, and a regularization term $R$:

$$E(I_m, I_f, \phi, \theta) = D(T(I_m, \phi), I_f) + R(\phi, \theta; I_m, I_f)$$

where again $T(I_m, \phi)$ denotes the transformed moving image. We use a simple gradient descent with adaptive step sizes $\{\tau_t\}$ as the base algorithm for the lower-level optimization, where the update to $\phi_t$ at step $t$ is given by:

$$\phi_{t+1} = \phi_t - \tau_t (\nabla D(T(I_m, \phi_t), I_f) + \nabla R(\phi_t, \theta; I_m, I_f))$$

(2)

where $\nabla D$ and $\nabla R$ are the gradient of the data-fidelity and regularization term w.r.t. the transformation $\phi_t$. We design the forward computation of GraDIRN to follow the lower-level optimization steps mathematically with a multi-resolution approach (detailed in Sec. 2.2), which yields the structure illustrated in Fig. 1a.

To incorporate the data-fidelity optimization in the network inference, the gradient of the fidelity w.r.t. the transformation $\nabla D(T(I_m, \phi_t), I_f)$ is explicitly computed as part of the forward computation of the network (detailed in Sec. 2.3). To incorporate rich image information and enable more complex local regularization to efficiently optimize the transformation, we generalize the regularization term in the lower-level optimization with learnable modules using a Convolutional Neural Networks (CNN) with parameters $\theta$. At each step, the CNN takes the current solution $\phi_t$ and the image pair $(I_m, I_f)$ as input, and outputs the gradient step corresponding to $\nabla R$ in Eq. (2). The parameters $\theta$ of the CNNs are shared within each resolution. The formulation of GraDIRN is not limited to any specific CNN architecture. In this paper, we use a simple 5-layer CNN with kernel size of 3 in each dimension and LeakyReLU with negative slope of 0.2 as the activation function, except the last layer where no activation function is used.

Higher-level optimization The parameters of the CNNs $\theta$ and the step sizes of the gradient descent $\{\tau_t\}$ are learned in the higher-level optimization, i.e. $\theta_t = \{\theta, \{\tau_t\}\}$. As shown in Fig. 1b, this is achieved by minimizing the expected loss $L$ over a training dataset $X$, given the solution of the lower-level optimization $\bar{\phi} = \phi_T$:

$$L(I_m, I_f, \bar{\phi}(\theta_t)) = L_D(T(I_m, \bar{\phi}(\theta_t)), I_f) + \lambda L_R(\bar{\phi}(\theta_t))$$

(3)

$$\tilde{\theta}_t = \arg \min \mathbb{E}_{(I_m, I_f) \sim X} [L(I_m, I_f, \bar{\phi}(\theta_t))]$$

(4)

where $\tilde{\theta}_t$ denotes the optimal parameters, $L_D$ is an image dissimilarity loss and $L_R$ is a regularization loss. The dissimilarity loss we use in this paper is a self-supervised loss which has the same form as the data-fidelity term in the energy function of the lower-level optimization, i.e. $L_D = D$. A diffusion-based regularization loss $L_R$ is used to constrain spatial variations of the output transformation in the higher-level optimization, i.e. $L_R = \| \delta \|$ with $\delta$ denoting a point in the spatial domain $\Omega$. We use stochastic gradient descent for this higher-level optimization. Note that the gradient here is computed w.r.t. the network parameters $\theta_t$, not to be confused with the gradients w.r.t. the transformation in Eq. (2).
2.2. Multi-resolution Optimization

Multi-resolution optimization has been shown to be effective in capturing both local deformation and global deformation while being computationally efficient in traditional iterative registration methods [3, 35, 43]. Recent deep learning registration methods also make use of the multi-resolution concept in their design to improve performance [19, 32].

GraDIRN can naturally incorporates multi-resolution optimization in its forward pass iterations similar to traditional methods. Specifically, we initialize the transformation \( \phi_0 \) with identity transformation, and optimize the transformation in a coarse-to-fine fashion following the gradient descent steps in Eq. (2). For resolution \( n \), the gradient descent update steps are applied \( T_n \) times. A total of \( T \) gradient descent steps are taken to give the final solution \( \tilde{\phi} = \phi_T \), where \( T = \sum_{n=1}^{N} T_n \). At the end of each resolution, the spatial resolution of the transformation is increased to be refined on the next (finer) resolution. To achieve continuous optimization between resolutions, the exact technique used to increase the resolution of the transformation depends on the transformation model of choice. In this paper, we use a dense displacement field as the transformation model thus linear interpolation is employed to up-sample the transformation, followed by scaling of the displacement fields to correct for the change of spatial domain.

To facilitate the multi-resolution optimization steps, we create a coarse-to-fine images pyramid of the moving and fixed images \( \{(I_m^{(n)}, I_f^{(n)})\}_{n=1,2,...N} \) to compute \( \nabla D \) and \( \nabla R \) at multiple resolutions.

2.3. Gradient of the Data-Fidelity term

The data-fidelity term in the energy functional Eq. (1) measures the dissimilarity between the images being registered. In this paper, we demonstrate the use of two image dissimilarity metrics as the data-fidelity term, namely the Sum of Squared Difference (SSD), which is defined as:

\[
D_{\text{SSD}}(I_m, I_f) = \sum_{x \in \Omega} (I_m(x) - I_f(x))^2 \tag{5}
\]

where \( I'_m = T(I_m, \tilde{\phi}) \) is the transformed moving image, and (negative) Normalized Cross-Correlation (NCC), which is defined as:

\[
D_{\text{NCC}}(I'_m, I_f) = \frac{-\sum_{x \in \Omega} [(I'_m(x) - \bar{I'_m})(I_f(x) - \bar{I_f})]^2}{\sum_{x \in \Omega} (I'_m(x) - \bar{I'_m})^2 \sum_{x \in \Omega} (I_f(x) - \bar{I_f})^2} \tag{6}
\]

where \( \bar{I} = \frac{1}{|\Omega|} \sum_{x \in \Omega} I(x) \) is the mean intensity of the image. Here, \( \Omega \) denotes the spatial domain of the images and \(|\Omega|\) denotes number of pixels/voxels in the domain. NCC is designed to be robust to additive and multiplicative intensity changes [10].

The gradient of the data term \( \nabla D(T(I_m, \phi_t), I_f) \) needs to be computed explicitly in the gradient descent steps as part of the forward computation of the network. This gradient computation also needs to be differentiable, since the stochastic gradient optimization in the higher-level optimization needs to compute the partial derivatives w.r.t. the learned parameters \( \theta \) and \( \{\tau\} \) via back-propagation.

Instead of implementing this gradient computation from its analytical form, we leverage the power of modern auto-differentiation engines [33] to create the forward graph of this explicit gradient computation. While avoiding inefficient code and human error in symbolic differentiation, this enables us to easily use other image dissimilarity functions as the data-fidelity term for multi-modal registration, as well as the use of other transformation models such as B-spline free-form deformation [35] for smooth and efficient parameterization or Stationary Velocity Field [1] for diffeomorphic registration.

3. Experiments

3.1. Tasks and Datasets

We extensively evaluate the proposed GraDIRN on 2D as well as 3D medical image registration tasks. We first evaluate on the task of intra-subject registration for cardiac motion estimation. For this, we use 220 cardiac cine-MR sequences from the UK Biobank study\(^1\). We split the sequences into 100/20/100 for training/validation/testing purposes. 2D registration is performed between the end-diastolic (ED) frame and end-systolic (ES) frame of the cine-MR sequence for each subject. The in-plane resolution is 1.8mm × 1.8mm per-pixel and the out-of-plane resolution is 10mm per pixel. Min-max intensity normalization to \([0, 1]\) is applied to both the ED and the ES frame with the same scaling factor. All images are cropped to 160 × 160 pixels in our experiments. For evaluation purpose only, we also generate a semantic segmentation of the left ventricle cavity (LV), myocardium (Myo) and right ventricle cavity (RV) using an automatic CNN-based segmentation model provided by [2].

We also evaluate on a more challenging inter-subject 3D brain MRI registration task using T1-weighted brain MR images from the Cambridge Centre for Ageing and Neuroscience (CamCAN) project open data inventory [38, 42]. The dataset contains structural MR scans of subjects aged 18-90 exhibiting large structural variations, making the inter-subject registration a challenging task. We randomly selected 310 image volumes out of the 652 scans available and split into 200/10/100 for training/validation/testing. As pre-processing, we first applied affine registration to spatially normalize all volumes to a common MNI space. Then,

\(^1\)http://imaging.ukbiobank.ac.uk
Table 1. Quantitative results of the metrics evaluated for all competing methods on both cardiac and brain MRI registration tasks whilst using SSD and NCC image (dis-)similarity, averaged across testing subjects and anatomical structures (with standard deviation in parenthesis). Dice (higher the better) and Hausdorff Distances (HD, lower the better) measures the registration accuracy. The determinant of Jacobian (|J|) metrics measures the spatial regularity of the resulting transformations (both the lower the better). The number of learnable parameters are listed in the “#Parameters” column. Metrics are measured after the initial affine registration as reference, shown in the “Affine” column.

| (Dis-)similarity | Method  | Dice | HD | | | #Parameters | Dice | HD | | | #Parameters |
|------------------|---------|------|----|-----------------|-----------------|------------------|------|----|-----------------|-----------------|
| SSD              | Affine  | 0.500(0.058) | 16.091(2.625) | n/a | n/a | n/a | 0.621(0.043) | 6.354(0.959) | n/a | n/a | n/a |
| NCC              | FFD     | 0.787(0.057) | 10.790(2.885) | 0.057(0.059) | 0.407(0.222) | n/a | 0.785(0.026) | 5.124(0.947) | 0.085(0.093) | 0.638(0.224) | n/a |
|                  | VOXelmorph | 0.806(0.055) | 10.235(2.764) | 0.084(0.054) | 0.549(0.192) | 106K | 0.777(0.029) | 5.430(0.840) | 0.084(0.064) | 0.625(0.177) | 320K |
|                  | GraDIRN | 0.812(0.052) | 10.101(2.738) | 0.099(0.058) | 0.581(0.190) | 954K | 0.793(0.025) | 5.213(0.836) | 0.074(0.059) | 0.589(0.170) | 2.9M |
|                  | VOXelmorph×9 | 0.821(0.052) | 9.729(2.531) | 0.188(0.076) | 0.866(0.182) | 88.5K | 0.794(0.026) | 5.166(0.892) | 0.078(0.059) | 0.603(0.171) | 269K |

Figure 2. Boxplots of Dice scores of competing methods measured on different anatomical structures (groups). In each box, the black line marks the median and the red line marks the mean.

To comprehensively evaluate the registration performances, we measure both the accuracy and the regularity of the transformation fields produced by registration. In-lieu of ground truth transformation, we follow standard prac-

---

2 https://www.nitrc.org/projects/robex
3 https://simpleitk.org
4 https://github.com/ledigchr/MALPEM
tice [5, 47] and measure registration accuracy based on the agreement of anatomical structures under the transformation from the registration of images. Concretely, given the segmentation masks extracted from the fixed images $S_f$, and the transformed segmentation masks extracted from the moving image $S'_m = T(S_m, \hat{\phi})$, we measure the Dice overlap for all anatomical structures by:

$$\text{Dice}(S'_m, S_f) = \frac{2|S'_m \cap S_f|}{|S'_m| + |S_f|}. \tag{7}$$

To quantify the regularity of the transformed masks, we also measure the Hausdorff distance (HD) \cite{7} which is given by:

$$\text{HD}(S'_m, S_f) = \max \left\{ \sup_{x_m \in \Omega_{S'_m}} d(x_m, S_f), \sup_{x_f \in \Omega_{S_f}} d(x_f, S'_m) \right\} \tag{8}$$

where $x_m, x_f$ are locations of points on the surfaces $\Omega_{S'_m}, \Omega_{S_f}$ of the segmentation masks, and $d(x, S)$ denotes the set of shortest distances from point $x$ to the surface of $S$, and $\sup$ denotes the supremum of the set.

In addition to accuracy, we also evaluate the local property of the transformation by measuring metrics based on the determinant of the Jacobian of the transformation $|J| = |\nabla \phi|$. We measure the percentage of the points in the image domain around which the space is folded under the deformation, reflected by negative $|J|$, denoted by $|J|_{<c\%}$, and the smoothness of the transformation reflected by standard deviation of the log-determinant of Jacobian, denoted by $\text{std log } |J|$.

The registration accuracy and the regularity of the transformation are often a trade-off in image registration. We take this into account as much as possible when selecting the hyper-parameters in our evaluation to compare the methods fairly. We do this by finding settings that lead to similar Jacobian metrics within reasonable computation cost of the parameter search.

### 3.3. Baseline Methods

We first compare the proposed method to a conventional registration method which uses B-spline free-form deformation as transformation (FFD). An energy functional consisting of the same data-fidelity terms as in Sec. 2.3 and a Bending Energy regularization \cite{35} was used. Registration using this method is achieved by iterative multi-resolution gradient-based optimization. We also compare our proposed method to two deep learning based registration methods. One is the widely used VoxelMorph framework \cite{4}. The other is a recursive-cascaded method introduced in \cite{47} as a representative method for a pure network-based iterative approach. Since this method composes cascades of base networks, we simply refer to it as multiples of the base network (e.g. VoxelMorph$\times9$).

### 3.4. Implementation

We evaluated the B-spline FFD algorithm implemented in the Medical Image Registration Toolkit \cite{5}, using three resolution levels and the corresponding image dissimilarity metrics in comparisons (i.e. SSD or NCC). We reimplemented the full-resolution version of VoxelMorph (VoxelMorph-2) and the recursive cascade baseline method \cite{47} with 9 cascades using VoxelMorph-2 as the base network. Correspondingly, we use 9 iterations ($T_N=9$) in our own GraDIRN in a 3 resolution setting, with 3 iterations in each resolution ($T_n=3$). Both deep learning baselines are trained using the same loss function as the ones used in the higher-level optimization of the proposed method. The weighting of the regularization $\lambda$ are chosen using the validation dataset with the trade-off consideration mentioned in Sec. 3.2.

We implemented our framework and deep learning baselines in Pytorch \cite{33} v1.8 and Pytorch Lightning \cite{13} v1.2. All models are trained and tested with mixed-precision on NVIDIA RTX A5000 and A6000 GPUs. Batch size of 10 was used for 2D models and batch size of 4 was used for 3D models. We use the Adam \cite{26} optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning rate of $10^{-4}$ for the higher-level optimization. The core part of our implementation is included in the supplementary material.

### 3.5. Results

Table 1 shows the quantitative results of the evaluation metrics of all baseline methods and our proposed GraDIRN on both 2D cardiac MR registration and 3D brain MR registration tasks. The results are shown separately for methods using SSD and NCC as the dissimilarity metric. The Dice and HD shown are the average of all the anatomical structures, with Dice results of separate structures shown in the boxplots in Fig. 2. The metrics evaluated on the initial alignment by affine registration are shown as a reference. Examples of the registered images and resulting transformations are shown in Fig. 3. The results demonstrate that the proposed GraDIRN outperforms the traditional B-spline FFD method and VoxelMorph in the cardiac MR registration task. B-spline FFD method is more competitive in the more challenging 3D brain registration task, with our proposed method achieving comparable accuracy. Similar conclusions can be reached across anatomical structures as shown in Fig. 2. Recursive cascade of VoxelMorph is able to achieve comparable accuracy on all tasks. However, it can only reach such a performance using significantly more learned parameters as shown in the #Parameters column in Tab. 1. It can also be noticed that GraDIRN achieved significantly higher accuracy than VoxelMorph while using fewer learnable parameters, demonstrating the potential benefit of using learnable parameters.
parameter efficiency by including explicit data-fidelity optimization in network inference. We investigate these aspects further in Sec. 3.6.

3.6. Ablation study

We extensively study the efficacy of the proposed framework by comparing variations of the framework with variations of the baseline method. For simplicity, we conduct these studies on 2D cardiac MR registration task using the SSD image dissimilarity setting.

**Parameter Efficiency** First, we investigate the parameter efficiency aspect of our approach in a fair comparison with the recursive cascaded network baseline [47]. The results are shown in Tab. 2. We compare our methods to variations of the cascaded network by switching the base network to the same light-weight network used in GraDIRN (denoted by “CNN$^G \times 9$”). To match the number of parameters used in GraDIRN, we also evaluate a variation of this where the parameters of the base networks are shared in the same way as in GraDIRN (denoted by “CNN$^G \times 3 \times 3$”). The results of this experiment indicate that the performance of purely cascading networks depends on the number of learnable parameters, where higher performance is only achieved with larger number of parameters.

**Multi-resolution** We investigated the effect of using multi-resolution scheme in our framework by comparing to a variant of our network that does not use multi-resolution. The single-resolution variant, denoted by “GraDIRN*” in Tab. 2, performs only marginally worse than its multi-resolution counterpart. However, the multi-resolution approach is found to be more memory efficient in our experiments.

**Explicit Data-fidelity** It is of great interest for our framework to study the significance of the explicit data-fidelity gradient in the network. We experimented with a variation of GraDIRN where the data-fidelity gradient step in Fig. 1a is removed, and the network reduces to a structure similar to residual network [17, 27]. As shown in Tab. 2, the registration performance without using explicit data-fidelity gradient notably decreases, which signals the significance and benefit of including data-fidelity gradient steps in the network.

Table 2. Study of parameter efficiency, resolutions and the effect of using explicit data-fidelity term using variations of the proposed method and baseline methods.

| Method       | Dice     | HD       | $	ext{J}(|\mathcal{J}|)$ | #Parameters |
|--------------|----------|----------|--------------------------|-------------|
| CNN$^G \times 9$ | 0.778(0.054) | 10.96(7.271) | 0.127(0.047) | 0.059(0.191) | 265K |
| CNN$^G \times 3 \times 1$ | 0.774(0.054) | 11.075(2.674) | 0.129(0.059) | 0.704(1.180) | 88.5K |
| GraDIRN*     | 0.799(0.052) | 10.48(2.738) | 0.082(0.050) | 0.547(0.182) | 88.5K |
| GraDIRN      | 0.811(0.054) | 10.094(2.763) | 0.078(0.045) | 0.528(0.165) | 88.5K |
| GraDIRN*     | 0.818(0.055) | 9.943(2.766) | 0.073(0.041) | 0.510(0.163) | 88.5K |

Table 3. Runtime measured for all competing methods for both CPU and GPU. All times are reported in seconds unless marked otherwise.

| Method       | 2D CPU | 2D GPU | 3D CPU | 3D GPU |
|--------------|--------|--------|--------|--------|
| SSD (CNN)    | 1.291(0.024) | 30.851(0.89) | n/a | n/a |
| NCC (CNN)    | 1.534(0.088) | 16.435(1.016) | n/a | n/a |
| Voxelmorph   | 0.011(0.005) | 0.858(0.053) | 2.796(0.636) | 0.003(0.002) |
| GraDIRN (SSD)| 0.065(0.006) | 4.726(0.343) | 42.031(1.506) | 0.214(0.002) |
| GraDIRN (NCC)| 0.075(0.007) | 4.589(0.333) | 59.502(2.283) | 0.220(0.003) |

3.7. Runtime

We measured the runtime required for each method to register a pair of 2D or 3D images on both CPU and GPU and report the results in Tab. 3. CPU runtimes are measured on a workstation with two Intel Xeon Silver 4214 CPUs. GPUs runtimes are measured on the same workstation using a NVIDIA RTX A6000 GPU. We observe that on both CPU and GPU the deep learning methods are orders of magnitude faster than the conventional B-spline FFD algorithm, especially for 3D registrations. Among the deep learning methods, the more accurate iterative methods can be seen noticeably slower than Voxelmorph [4], with the proposed GraDIRN slightly faster than the recursive cascaded method [4] overall.

4. Discussion

The structure of the network we proposed is designed by following classical energy minimization optimization steps. Intuitively, this design explicitly minimize the image dissimilarity as part of its forward pass thus constrains the solution space of the network parameters under the bi-level optimization view. This leads to efficient use of learned parameters while achieving state-of-the-art registration performance. As shown in our ablation study in Sec. 3.6, the image dissimilarity gradient steps are found essential in achieving competitive performance, which is in-line with the principles in unrolled optimization approaches in related fields [16, 37]. By unrolling the gradient-based optimization and implementing the data-fidelity gradient forward steps with auto-differentiation, a feature widely available in modern deep learning software frameworks, we make our framework more accessible to the community and easier to be extended to arbitrary differentiable data-fidelity terms and transformations for general image registration.

5. Limitations and Future Works

As we have shown, it seems while introducing iteration into deep learning registration with unrolled optimization improves registration performance, this comes with a computational cost, both in memory usage and speed. Although our network-based approach is still significantly faster than
traditional iterative algorithms, especially so when GPU and deep learning frameworks are used, slow-down from a simple one-network method such as VoxelMorph is noticeable. The memory efficiency is improved by using a multi-resolution optimization scheme in our method. And the overall efficiency could be improved if the transformation is optimized in a lower-dimensional parameterization such as B-spline FFD or in a band-limited space [44]. We will explore these avenues in the future.

We have demonstrated that our framework can be adapted to Normalized Cross-Correlation dissimilarity metric which can be used for some multi-modal registration tasks. While we only demonstrated this on mono-modal registration tasks in this paper, we will experiment on more image dissimilarity metrics such as MI [22] or MIND [18] for challenging multi-modal registration tasks in our future work.

6. Conclusion

In this paper, we present a novel framework for learning-based deformable image registration that embeds deep neural networks in the theoretical formulation of iterative gradient-based energy minimization. Using extensive evaluations, we show that our approach obtains state-of-the-art registration accuracy while retaining parameter efficiency. Further analysis show that using explicit data-fidelity gradient as part of the forward pass is vital for our framework to achieve competitive performance. Using a gradient-based optimization network allows us to use SSD as well as the more robust NCC as the data-fidelity term easily, facilitating the use of arbitrary image dissimilarity and more transformation models for general deformable image registration.

References

[1] John Ashburner. A fast diffeomorphic image registration algorithm. *NeuroImage*, 38(1):95–113, 2007.
[2] Wenjia Bai, Matthew Sinclair, Giacomo Tarroni, Ozan Oktay, Martin Rajchl, Ghislain Vaillant, Aaron M Lee, Nay Aung, Elena Lukaschuk, Mihir M Sanghvi, Filip Zemrak, Kenneth Fung, Jose Miguel Paiva, Valentina Carpella, Young Jin Kim, Hideaki Suzuki, Bernhard Kainz, Paul M Matthews, Steffen E Petersen, Stefan K Piechnik, Stefan Neubauer, Ben Glocker, and Daniel Rueckert. Automated cardiovascular magnetic resonance image analysis with fully convolutional networks. *J. Cardiovasc. Magn. Reson.*, 20(1):65, Sept. 2018.
[3] Ruzena Bajcsy and Stanislav Kovacic. Multiresolution elastic matching. *Comput. Vis. Graph. Image Process.*, 46(1):1–21, 1989.
[4] Guha Balakrishnan, Amy Zhao, Mert R. Sabuncu, John V. Guttag, and Adrian V. Dalca. An unsupervised learning model for deformable medical image registration. In *CVPR*, pages 9252–9260. Computer Vision Foundation / IEEE Computer Society, 2018.
[5] Guha Balakrishnan, Amy Zhao, Mert R. Sabuncu, John V. Guttag, and Adrian V. Dalca. VoxelMorph: A learning framework for deformable medical image registration. *IEEE Trans. Medical Imaging*, 38(8):1788–1800, 2019.
[6] Ghalib A Bello, Timothy J W Dawes, Jimming Duan, Carlo Biffi, Antonio de Marvao, Luke S G E Howard, J Simon R Gibbs, Martin R Wilkins, Stuart A Cook, Daniel Rueckert,
and Declan P O’Regan. Deep learning cardiac motion analysis for human survival prediction. *Nat Mach Intell.*, 1(2):95–104, Feb. 2019.

[7] T. Birsan and Dan Tiba. One hundred years since the introduction of the set distance by dimitrie Pompeu. In *System Modelling and Optimization*, volume 199 of *IFIP International Federation for Information Processing*, pages 35–39. Springer, 2018.

[8] Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1256–1272, 2017.

[9] Bob D. de Vos, Floris F. Berendsen, Max A. Viergever, Hessel Sokooti, Marius Staring, and Ivana Isgum. A deep learning framework for unsupervised affine and deformable image registration. *Medical Image Anal.*, 52:128–143, 2019.

[10] Luigi di Stefano, Stefano Mattoccia, and Martino Mola. An efficient algorithm for exhaustive template matching based on normalized cross correlation. In *ICIP*, pages 322–327. IEEE Computer Society, 2003.

[11] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, pages 2758–2766. IEEE Computer Society, 2015.

[12] Jiming Duan, Jo Schlemper, Chen Qin, Cheng Ouyang, Wenjia Bai, Carlo Biffi, Ghaleb Bello, Ben Statton, Declan P. O’Regan, and Daniel Rueckert. Vs-net: Variable splitting network for accelerated parallel MRI reconstruction. In *MICCAI* (4), volume 11767 of *Lecture Notes in Computer Science*, pages 713–722. Springer, 2019.

[13] William Falcon et al. Pytorch lightning. *GitHub*. Note: https://github.com/PyTorchLightning/pytorch-lightning, 3:6, 2019.

[14] Davis Gilton, Greg Ongie, and Rebecca Willett. Neumann networks for linear inverse problems in imaging. *IEEE Trans. Computational Imaging*, 6:328–343, 2020.

[15] Dong Gong, Zhen Zhang, Qinfeng Shi, Anton van den Hengel, Chunhua Shen, and Yanning Zhang. Learning deep gradient descent optimization for image deconvolution. *IEEE Trans. Neural Networks Learn. Syst.*, 31(12):5468–5482, 2020.

[16] Kerstin Hammernik, Teresa Klatzer, Erich Kohler, Michael P. Recht, Daniel K. Sodickson, Thomas Pock, and Florian Knoll. Learning a variational network for reconstruction of accelerated MRI data. *CoRR*, abs/1704.00447, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[18] Mattias P. Heinrich, Mark Jenkinson, Manav Bhushan, Tahreema N. Matin, Fergus Gleeson, Michael Brady, and Julia A. Schnabel. MIND: modality independent neighbourhood descriptor for multi-modal deformable registration. *Medical Image Anal.*, 16(7):1423–1435, 2012.

[19] Alessa Hering, Bram van Ginneken, and Stefan Heldmann. mlvirmnet: Multilevel variational image registration network.

In *MICCAI* (6), volume 11769 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 2019.

[20] Yipeng Hu, Marc Modat, Eli Gibson, Wenzhen Ghami, Ester Bonmati, Guotai Wang, Steven Bandula, Caroline M. Moore, Mark Emberton, Sébastien Ourm精灵, J. Alison Noble, Dean C. Barratt, and Tom Vercauteren. Weakly-supervised convolutional neural networks for multimodal image registration. *Medical Image Anal.*, 49:1–13, 2018.

[21] Xia Hua, Alex D. Leow, Neelooop Parikshak, Suh Lee, Ming-Chang Chiang, Arthur W. Toga, Clifford R. Jack Jr., Michael W. Weiner, and Paul M. Thompson. Tensor-based morphometry as a neuroimaging biomarker for Alzheimer’s disease: An MRI study of 676 AD, MCI, and normal subjects. *NeuroImage*, 43(3):458–469, 2008.

[22] William M. Wells III, Paul A. Viola, Hideki Atsumi, Shin Nakajima, and Ron Kikinis. Multi-modal volume registration by maximization of mutual information. *Medical Image Anal.*, 1(1):35–51, 1996.

[23] Eddy Ilg, Nikolaus Mayer, Tommo Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*, pages 1647–1655. IEEE Computer Society, 2017.

[24] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[25] Xi Jia, Alexander Thorley, Wei Chen, Huaiqi Qiu, Linlin Shen, Iain B Styles, Hyung Jin Chang, Ales Leonardis, Antonio De Marvao, Declan P O’Regan, Daniel Rueckert, and Jiming Duan. Learning a model-driven variational network for deformable image registration. *IEEE Trans. Med. Imaging*, pages 1–1, 2021.

[26] Diederek P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[27] Erich Kohler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: Connecting variational methods and deep learning. In *GCPR*, volume 10496 of *Lecture Notes in Computer Science*, pages 281–293. Springer, 2017.

[28] Christian Ledig, Rolf A. Heckemann, Alexander Hammers, Juan Carlos López, Virginia F. J. Newcombe, Antonios Makropoulos, Jyrki Löötjönen, David K. Menon, and Daniel Rueckert. Robust whole-brain segmentation: Application to traumatic brain injury. *Medical Image Anal.*, 21(1):40–58, 2015.

[29] Matthew C. H. Lee, Ozan Oktay, Andreas Schuh, Michiel Schaal, and Ben Glocker. Image-and-spatial transformer networks for structure-guided image registration. In *MICCAI* (2), volume 11765 of *Lecture Notes in Computer Science*, pages 337–345. Springer, 2019.

[30] Risheng Liu, Zi Li, Xin Fan, Chenying Zhao, Hao Huang, and Zhongxuan Luo. Learning deformable image registration from optimization: Perspective, modules, bilevel training and beyond. *IEEE Trans. Pattern Anal. Mach. Intell.*, PP, Sept. 2021.

[31] Tony C. W. Mok and Albert C. S. Chung. Fast symmetric diffeomorphic image registration with convolutional neu-
ral networks. In CVPR, pages 4643–4652. Computer Vision Foundation / IEEE, 2020.

[32] Tony C. W. Mok and Albert C. S. Chung. Large deformation diffeomorphic image registration with laplacian pyramid networks. In MICCAI (3), volume 12263 of Lecture Notes in Computer Science, pages 211–221. Springer, 2020.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasanak Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In NeurIPS, pages 8024–8035, 2019.

[34] Marc-Michel Rohé, Manasi Datar, Tobias Heimann, Maxime Sermesant, and Xavier Pennec. Svf-net: Learning deformable image registration using shape matching. In MICCAI (1), volume 10433 of Lecture Notes in Computer Science, pages 266–274. Springer, 2017.

[35] Daniel Rueckert, Luke I. Sonoda, Carmel Hayes, Derek L. G. Hill, Martin O. Leach, and David J. Hawkes. Non-rigid registration using free-form deformations: Application to breast MR images. IEEE Trans. Medical Imaging, 18(8):712–721, 1999.

[36] Robin Sandkühler, Simon Andermatt, Grzegorz Bauman, Sylvia Nyilas, Christoph Jud, and Philippe C. Cattin. Recurrent registration neural networks for deformable image registration. In NeurIPS, pages 8755–8765, 2019.

[37] Jo Schlemper, Jose Caballero, Joseph V. Hajnal, Anthony N. Price, and Daniel Rueckert. A deep cascade of convolutional neural networks for dynamic MR image reconstruction. IEEE Trans. Medical Imaging, 37(2):491–503, 2018.

[38] Meredith A Shafto, Lorraine K Tyler, Marie Dixon, Jason R Taylor, James B Rowe, Rhodri Cusack, Andrew J Calder, William D Marslen-Wilson, John Duncan, Tim Dalgleish, Richard N Henson, Carol Brayne, Fiona E Matthews, and Cam-CAN. The cambridge centre for ageing and neuroscience (Cam-CAN) study protocol: a cross-sectional, lifespan, multidisciplinary examination of healthy cognitive ageing. BMC Neurol., 14:204, Oct. 2014.

[39] Hessam Sokooti, Bob D. de Vos, Floris F. Berendsen, Boudewijn P. F. Lelieveldt, Ivana Isgum, and Marius Staring. Nonrigid image registration using multi-scale 3d convolutional neural networks. In MICCAI (1), volume 10433 of Lecture Notes in Computer Science, pages 232–239. Springer, 2017.

[40] Aristeidis Sotiras, Christos Davatzikos, and Nikos Paragios. Deformable medical image registration: A survey. IEEE Trans. Medical Imaging, 32(7):1153–1190, 2013.

[41] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In CVPR, pages 8934–8943. Computer Vision Foundation / IEEE Computer Society, 2018.

[42] Jason R. Taylor, Nitin Williams, Rhodri Cusack, Tibor Auer, Meredith A. Shafto, Marie Dixon, Lorraine K. Tyler, Cam-CAN Group, and Richard N. Henson. The cambridge centre for ageing and neuroscience (cam-can) data repository: Structural and functional mri, meg, and cognitive data from a cross-sectional adult lifespan sample. NeuroImage, 144:262–269, 2017.

[43] Tom Vercauteren, Xavier Pennec, Aymeric Perchant, and Nicholas Ayache. Diffeomorphic demons: Efficient non-parametric image registration. NeuroImage, 45(1):S61–S72, 2009.

[44] Jian Wang and Miaomiao Zhang. Deepflash: An efficient network for learning-based medical image registration. In CVPR, pages 4443–4451. Computer Vision Foundation / IEEE, 2020.

[45] Xiao Yang, Roland Kwitt, Martin Styner, and Marc Niethammer. Quicksilver: Fast predictive image registration - A deep learning approach. NeuroImage, 158:378–396, 2017.

[46] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-L 1 optical flow. In DAGM-Symposium, volume 4713 of Lecture Notes in Computer Science, pages 214–223. Springer, 2007.

[47] Shengyu Zhao, Yue Dong, Eric I-Chao Chang, and Yan Xu. Recursive cascaded networks for unsupervised medical image registration. In ICCV, pages 10599–10609. IEEE, 2019.
Supplementary Material

Contents

1. Code 1
2. GraDIRN Iterative Optimization Visual Demonstration 11
3. Additional Visual Examples of Registration Results 14

1. Code

We provide here the code for the core part of GraDIRN implemented in Pytorch v1.8.1. Code for the main model, key components of the model and loss functions are provided so the forward pass pipeline can be constructed. The data and code for data loading are not provided since we are not allowed to distribute the medical imaging data ourselves. Training logic is also not provided since we use standard Adam optimizer with parameters provided in the main paper.

GraDIRN core module code including the data fidelity terms (or dissimilarity) and the CNN-parameterized regularization terms:

```python
class GradIRN(torch.nn.Module):
    ""
    'num_blocks' is a list of number of blocks in each resolution.
    The similarity and regulariser blocks are each a nn.ModuleList of nn.ModuleList(s),
    each list for one resolution:
    '{[block1, block2, ..., block_<num_blocks[0]>>,
    [block1, block2, ..., block_<num_blocks[1]>>,
    ...]}'
    ""
    def __init__(self,
                 ndim,
                 num_blocks=(1, 1, 1),
                 num_repeat=(3, 3, 3),
                 scale_step_size=True,
                 tau_config=None,
                 similarity='ssd',
                 similarity_config=None,
                 regulariser_config=None,
                 return_series=True,
                 ):
        super().__init__()
        self.ndim = ndim
        self.num_blocks = num_blocks  # number of blocks per resolution
        self.num_repeat = num_repeat  # number of times blocks are repeated per resolution
        self.num_resolutions = len(num_blocks)
        self.return_series = return_series
        
        if scale_step_size:
            # scale the initial tau with 1/(number_or_blocks)
            step_size_scale = 1 / (sum([a*b for a, b in zip(num_blocks, num_repeat)]) + 1)
            tau_config.init = step_size_scale * tau_config.init
```

arXiv:2112.03915v1  [eess.IV]  7 Dec 2021
# configure similarity term
SIMS = {'ssd': SSDSimilarity, 'ncc': NCCSimilarity}
SIMILARITY = SIMS[similarity]

# configure regulariser term
REGULARISER = CNNGRegulariser
self.reg_input_images = regulariser_config.input_images

# main blocks
self.sim_blocks = torch.nn.ModuleList()
self.reg_blocks = torch.nn.ModuleList()
for nb in self.num_blocks:
    self.sim_blocks.append(torch.nn.ModuleList([SIMILARITY(tau_config, **similarity_config) for _ in range(nb)]))
    self.reg_blocks.append(torch.nn.ModuleList([REGULARISER(config=regulariser_config) for _ in range(nb)]))

@staticmethod
def get_norm_grid(size, device):
    grid = torch.meshgrid([torch.linspace(-1, 1, s, device=device) for s in size])
    grid = torch.stack(grid, 0).requires_grad_(False) # (ndims, *size)
    return grid

def forward(self, tars, srcs):
    # Input 'tars' and 'srcs' are list of images with increasing resolution

    # initialise disp
    device = tars[0].device
    disp = torch.zeros(tars[0].shape[0], self.ndim, *tars[0].shape[2:], device=device)
    disps = []
    for lvl in range(self.num_resolutions):
        self.grid = self.get_norm_grid(tars[lvl].shape[2:], device)
        for sim_block, reg_block in zip(self.sim_blocks[lvl], self.reg_blocks[lvl]):
            for _ in range(self.num_repeat[lvl]): # repeating blocks
                tar, src = tars[lvl], srcs[lvl]
                with torch.set_grad_enabled(True):
                    # compute explicit gradient step w.r.t. disp with autograd
                    disp.requires_grad_()
                    warped_src = warp_fn(src, disp, self.grid, backwards_grad=True)
                    sim_step = sim_block(tar, warped_src, disp)
                    # regulariser step
                    if self.reg_input_images == 'explicit':
                        reg_input = torch.cat((disp, tar, warped_src), dim=1)
                    elif self.reg_input_images == 'implicit':
                        reg_input = torch.cat((disp, tar, src), dim=1)
                    else:
                        reg_input = disp
                    reg_step = reg_block(reg_input)
                    disp = disp - sim_step - reg_step
                    disps.append(disp)
            if lvl < self.num_resolutions - 1:
                disp = interpolate_nd(disp, scale_factor=2.) * 2.
        if self.return_series:
            return disps
    else:
        return [disp]

Listing 1. GraDIRN module

class SimilarityTerm(torch.nn.Module):

def __init__(self, tau_config, ndim, dtype=torch.float32):
    super(SimilarityTerm, self).__init__()
    self.ndim = ndim
    self._tau_ = torch.nn.Parameter(torch.tensor(1, dtype=dtype))
    reset_scalar(self._tau_, **tau_config)

@property
def tau_(self):
    return self._tau_ * self._tau_.train_scale

def energy(self, tar, warped_src):
    return NotImplementedError

def grad(self, tar, warped_src, disp):
    return NotImplementedError

def forward(self, tar, warped_src, disp):
    return self.grad(tar, warped_src, disp) * self.tau_

class SSDSimilarity(SimilarityTerm):
    def __init__(self, tau_config, ndim, dtype=torch.float32):
        super().__init__(tau_config, ndim, dtype=dtype)

    def energy(self, tar, warped_src):
        """ SSD similarity energy """
        return F.mse_loss(tar, warped_src)

    def grad(self, tar, warped_src, disp):
        """ Gradient step of the SSD dis-similarity"""
        e = self.energy(tar, warped_src)
        grad = torch.autograd.grad(e, disp, create_graph=self.training)[0]
        # correct the magnitude by number of points because of the averaging in F.mse_loss()
        grad = grad * torch.tensor(tar.shape).prod()
        return grad

class NCCSimilarity(SimilarityTerm):
    def __init__(self, tau_config, ndim, dtype=torch.float32):
        super().__init__(tau_config, ndim, dtype=dtype)

    def energy(self, tar, warped_src):
        return ncc(tar, warped_src)

    def grad(self, tar, warped_src, disp):
        e = self.energy(tar, warped_src)
        grad = torch.autograd.grad(e, disp, create_graph=self.training)[0]
        grad = grad * torch.tensor(tar.shape).prod()
        return grad

def ncc(x, y):
    """ Global Normalised Cross Correlation """
    x = x.view(x.shape[0], -1)
    y = y.view(y.shape[0], -1)

    x_bar = x - x.mean(dim=1, keepdim=True) # (N, 1)
    y_bar = y - y.mean(dim=1, keepdim=True) # (N, 1)

    with torch.cuda.amp.autocast(enabled=False): # compute in float32 to avoid overflow
        x_var = x_bar.square().sum(dim=1)

3
\[
y_{\text{var}} = y_.\text{square()}.\text{sum(dim=1)} \\
cov2 = (x_ * y_).\text{sum(dim=1).square()} \\
ncc = cov2 / (x_{\text{var}} * y_{\text{var}} + 1e-5) \\
\text{return} -\text{ncc.mean()}\]

def reset_scalar(scalar, init=1., min=0, max=1000, requires_grad=True, train_scale=1):
    """ Reset scalar parameters: set attributes, attach projection ""
    scalar.data = torch.tensor(init, dtype=scalar.dtype)
    scalar.proj = lambda: scalar.data.clamp_(min, max)
    scalar.requires_grad = requires_grad
    scalar.train_scale = train_scale

class Regulariser(torch.nn.Module):
    """Base regulariser term/step class""
    def __init__(self, dtype=torch.float32):
        super(Regulariser, self).__init__()
        self.dtype = dtype

    def energy(self, x):
        raise NotImplementedError

    def grad(self, x):
        raise NotImplementedError

    def forward(self, x):
        return self.grad(x)

class CNNRegulariser(Regulariser):
    def __init__(self, config=None):
        super(CNNRegulariser, self).__init__()
        self.layers = nn.ModuleList()

        ndim = config.ndim
        num_layers = config.num_layers
        num_channels = config.num_channels

        indim = ndim + 2 if config.input_images else ndim
        for i in range(num_layers-2):
            self.layers.append(nn.Sequential(convNd(ndim, indim, num_channels, a=0.2), nn.LeakyReLU(0.2)))
        self.layers.append(nn.Sequential(convNd(ndim, num_channels, ndim))

        if config.input_images:
            self.layers.append(nn.LeakyReLU(0.2))

        def grad(self, x):
            x = layer(x)
            return x

def convNd(ndim,
           in_channels,
           out_channels,
           kernel_size=3,
           stride=1,
           padding=1,
           a=0.):
    """Wrapper to instantiate convolution module of generic dimension""
    Args:
        in_channels: (int) number of input channels
        out_channels: (int) number of output channels
        kernel_size: (int) size of the convolution kernel
stride: (int) convolution stride (step size)
padding: (int) outer padding
ndim: (int) model dimension
a: (float) leaky-relu negative slope for He initialisation

Returns:
(nn.Module instance) Instance of convolution module of the specified dimension

assert ndim > 0
conv_nd = [nn.Conv2d, nn.Conv3d][ndim-2](in_channels, out_channels, kernel_size, stride=stride,
padding=padding)
nn.init.kaiming_uniform_(conv_nd.weight, a=a)
return conv_nd

Listing 3. CNN-parameterized regularization steps

Twice differentiable spatial transformation function, critical for using auto-differentiation to compute explicit data fidelity gradient:

def warp_fn(x, disp, grid, interp_mode="bilinear", backwards_grad=False):
    ""
    Warping function that takes grid as input (avoids recreating grid)
    disp and grid both shaped: (N, ndims, *size), i-j order
    ""
    disp = normalise_disp(disp)
    warped_grid = grid + disp
    ndims = x.ndim - 2
    warped_grid = torch.movedim(warped_grid, 1, -1)[..., list(range(ndims))[::-1]]
    align_corners = None if interp_mode == 'nearest' else True
    if backwards_grad: # use twice differentiable grid sample
        assert interp_mode == 'bilinear', 'Only linear interpolation allowed.'
        return grid_sample(x, warped_grid)
    else:
        return F.grid_sample(x, warped_grid, mode=interp_mode, align_corners=align_corners)

def grid_sample(x, grid):
    ""
    Twice differentiable N-D grid sample with bi/tri-linear interpolation using Tensor operations
to replace the functional 'F.grid_sample()'
    ""
    Args:
        x: (torch.Tensor) shape (N, C, *size)
        grid: (torch.Tensor) shape (N, *size, ndims), locations to sample, normalised, x-y order
    Returns:
        out: x sampled at grid via linear interpolation
    ""
    assert x.shape[2:] == grid.shape[1:-1]
    shape = x.shape
    size = shape[2:]
    ndims = len(size)

    grid = unnormalise_grid(grid, size)  # (N, *size, ndims: (x, y, (z)))
    grid_floor = grid.floor()  # (N, *size, ndims: (x0, y0, (z0)))
    grid ceil = grid floor + 1  # (N, *size, ndims: (x1, y1, (z1)))

    # calculate offsets (normalised) for interpolation weights
    offset = grid - grid floor  # (N, *size, ndims: (x-x0, y-y0, (z-z0)))
    offset_inv = 1 - offset  # (N, *size, ndims: (1-x-x0, 1-y-y0, (1-(z-z0)))
    offsets = ([offset[..., i] for i in range(ndims)], [offset_inv[..., i] for i in range(ndims)])

    # border handling (replicate border)
    locs = [grid_floor[..., i].clamp(0, size[i] - 1) for i in range(ndims)],
           [grid ceil[..., i].clamp(0, size[i] - 1) for i in range(ndims)]

    # flatten data for torch.gather()
    x = x.view(*shape[2:], -1)
out = []
for corner_point in itertools.product([0, 1], repeat=ndims):
    # build the weight of the corner
    corner_weight = tensor_prod([offsets[1 - corner_point[i]] for i in range(ndims)])
    # (N, 1, *size)

    # find the corner locations
    corner_locs = [locs[corner_point[i]] for i in range(ndims)]

    # gather the corner values at the corner locations
    corner_idx = loc_to_idx(corner_locs, size)
    corner_idx = corner_idx.view(shape[0], 1, -1).repeat(1, shape[1], 1).long()
    corner_val = x.gather(dim=2, index=corner_idx)
    corner_val = corner_val.view(*shape) # (N, C, *size)

    out_corner = corner_weight * corner_val
    out.append(out_corner)
out = tensor_sum(out)
return out

def loc_to_idx(locs, size):
    
    Convert spatial locations to indices in flattened vector (ravel index)
    Formula for x-y order:
    - 2D: idx = x + y * W
    - 3D: idx = x + y * W + z * W * H
    
    Args:
    locs: a list of coordinates [x, y, (z)] each of shape (N, *size)
    size: (H, W) or (D, H, W)
    
    Returns:
    idx: (torch.Tensor) index of locs in ravel tensor shape (N, prod(size))
    
    assert len(locs) == len(size)
    csize = np.cumprod(size[::-1]) # (W, H*W) or (W, H*W, H*W*D)
    idx = locs[0]
    for i, loc in enumerate(locs[1:]):
        idx = idx + loc * csize[i]
    return idx

def unnormalise_grid(grid, size):
    
    spatially normalise displacement to [-1, 1] coordinate system used by Pytorch 'grid_sample()'
    Assumes disp size is the same as the corresponding image.
    
    Args:
    disp: (torch.Tensor, shape (N, ndim, *size)) Displacement field
    
    Returns:
    disp: (normalised disp)
    
    ndim = disp.ndim - 2
    norm_factors = torch.tensor(2.) / torch.tensor(disp.size()[2:], dtype=disp.dtype, device=disp.device)
    norm_factors = norm_factors.view(1, ndim, *(1,) * ndim)
def tensor_prod(tensor_list):
    """ Element-wise product of a list of tensors ""
    out = tensor_list[0]
    for x in tensor_list[1:]:
        out = out * x
    return out

def tensor_sum(tensor_list):
    """ Element-wise sum of a list of tensors ""
    out = tensor_list[0]
    for x in tensor_list[1:]:
        out = out + x
    return out

Listing 4. Spatial transformation function and some utility functions

Diffusion-based regularization loss function:

def diffusion_reg_loss(u):
    """ Diffusion-based regularisation loss""
    derives = []
    ndim = u.size()[1]
    for i in range(ndim):
        derives += [diff(u, dim=i)]
    loss = torch.cat(derives, dim=1).pow(2).sum(dim=1).mean()
    return loss

def diff(x, dim, mode="forward", boundary="Neumann"):
    """ Finite difference operator
    Args:
    x: (torch.Tensor, shape (N, ndim, *sizes))
    dim: (int) the dimension along which to compute finite difference
    mode: (str) Finite difference direction 'forward', 'backward' or 'central'
    boundary: (str) Boundary handling method, 'Neumann' or 'Dirichlet'
    ""
    ndim = x.ndim - 2
    sizes = x.shape[2:]

    # paddings
    paddings = [[0, 0] for _ in range(ndim)]
    if mode == "forward":
        # forward difference: pad after
        paddings[dim][1] = 1
    elif mode == "backward":
        # backward difference: pad before
        paddings[dim][0] = 1
    else:
        raise ValueError(f'Mode {mode} not recognised')

    # reverse and join sublists into a flat list
    # (Pytorch uses last -> first dim order when padding)
    paddings.reverse()
    paddings = [p for ppair in paddings for p in ppair]

    # pad data
    if boundary == "Neumann":
        # Neumann boundary condition
        x_pad = F.pad(x, paddings, mode='replicate')
    elif boundary == "Dirichlet":
        # Dirichlet boundary condition
        x_pad = F.pad(x, paddings, mode='constant')
    else:
Listing 5. Diffusion-based regularization loss function and finite-difference operator used for the loss

```python
def create_img_pyramid(x, lvls=1, label=False):
    """ Create image pyramid, low-resolution to high-resolution""
    x_pyr = [x]
    interp_mode = 'nearest' if label else None
    for l in range(lvls-1):
        x_pyr.append(interpolate_nd(x, scale_factor=0.5 ** (l + 1), mode=interp_mode))
    x_pyr.reverse()  # low resolution to high resolution
    return x_pyr

def interpolate_nd(x, scale_factor=None, size=None, mode=None):
    """ Wrapper for torch.nn.functional.interpolate """
    if mode == 'nearest':
        mode = mode
        align_corners = None
    else:
        ndims = x.ndim - 2
        align_corners = True
        if ndims == 1:
            mode = 'linear'
        elif ndims == 2:
            mode = 'bilinear'
        elif ndims == 3:
            mode = 'trilinear'
        else:
            raise ValueError(f'Data dimension ({ndims}) must be 2 or 3')
    y = F.interpolate(x,
                     scale_factor=scale_factor,
                     size=size,
                     mode=mode,
                     recompute_scale_factor=False,
                     align_corners=align_corners)
    return y
```

Listing 6. Image pyramid

Putting everything together, we provide the following demo code to run the network forward pass and loss computation for both 2D and 3D settings (requires installing the omegaconf package):

```python
import torch.nn.functional as F

from omegaconf import OmegaConf

def get_network(ndim):
    # config
    tau_config = ""
    init: 0.5
    requires_grad: True
    train_scale: 10.0
    min: 0.001
    max: 10000
    ""
    tau_config = OmegaConf.create(tau_config)
    similarity_config = OmegaConf.create({'ndim': ndim})
    regulariser_config = f""
    ndim: {ndim}
    num_layers: 5
```

8
num_channels: 32
input_images: 'implicit'

regulariser_config = OmegaConf.create(regulariser_config)

# instantiate the network
gradin = GradIRN(ndim,
    init_mode='identity',
    num_blocks=(1, 1, 1),
    num_repeat=(3, 3, 3),
    scale_step_size=True,
    tau_config=tau_config,
    similarity='ssd',
    similarity_config=similarity_config,
    regulariser_config=regulariser_config,
    return_series=True)

return gradin

# -- 2d demo -- #
# generate psuedo data
size_2d = 160
batch_size_2d = 10
I_fixed_2d = torch.randn(batch_size_2d, 1, size_2d, size_2d)
I_moving_2d = torch.randn(batch_size_2d, 1, size_2d, size_2d)

I_fixed_pyramid_2d = create_img_pyramid(I_fixed_2d, lvls=3)
I_moving_pyramid_2d = create_img_pyramid(I_moving_2d, lvls=3)

# network inference / forward pass
gradin_2d = get_network(ndim=2)
disps_2d = gradin_2d(I_fixed_pyramid_2d, I_moving_pyramid_2d)

for x in disps_2d:
    print(x.shape)

# loss
disp2d = disps_2d[-1]
lambda_ = 0.1
grid_2d = gradin_2d.get_norm_grid(disp2d.shape[2:], device=disp2d.device)
I_moving_2d_warped = warp_fn(I_moving_2d, disp2d, grid_2d)
loss_mse_2d = F.mse_loss(I_fixed_2d, I_moving_2d_warped) + lambda_ * diffusion_reg_loss(disp2d)
loss_ncc_2d = ncc(I_fixed_2d, I_moving_2d_warped) + lambda_ * diffusion_reg_loss(disp2d)

# -- 3d demo -- #
# generate psuedo data
size_3d = 128
batch_size_3d = 4
I_fixed_3d = torch.randn(batch_size_3d, 1, size_3d, size_3d, size_3d)
I_moving_3d = torch.randn(batch_size_3d, 1, size_3d, size_3d, size_3d)

I_fixed_pyramid_3d = create_img_pyramid(I_fixed_3d, lvls=3)
I_moving_pyramid_3d = create_img_pyramid(I_moving_3d, lvls=3)

# network inference / forward pass
gradin_3d = get_network(ndim=3)
disps_3d = gradin_3d(I_fixed_pyramid_3d, I_moving_pyramid_3d)

for x in disps_3d:
    print(x.shape)

# loss
disp3d = disps_3d[-1]
lambda_ = 0.1
grid_3d = gradin_3d.get_norm_grid(disp3d.shape[2:], device=disp3d.device)
I_moving_3d_warped = warp_fn(I_moving_3d, disp3d, grid_3d)
loss_mse_3d = F.mse_loss(I_fixed_3d, I_moving_3d_warped) + lambda_ * diffusion_reg_loss(disp3d)
loss_ncc_3d = ncc(I_fixed_3d, I_moving_3d_warped) + lambda_ * diffusion_reg_loss(disp3d)

Listing 7. Demo network forward pass and loss computation
2. GraDIRN Iterative Optimization Visual Demonstration

Here, we visualize the iterative optimization in the forward pass of GraDIRN, which shows the multi-step multi-resolution progressive optimization of the transformation. This is shown for both the cardiac MRI registration and brain MRI registration with SSD and NCC settings in Fig. 1, Fig. 2, Fig. 3 and Fig. 4. For visualization purpose, the deformed grids $\phi$ in multiple resolutions are drawn with increasing spacing (2, 4, 8) before interpolating to the same image size so the "observation density" of the transformation is the same across resolutions.

Figure 1. Illustration of iterative optimization in forward pass of GraDIRN, $t$ denotes step number, $n$ denotes the resolution number. Cardiac SSD setting.
Figure 2. Illustration of iterative optimization in forward pass of GraDIRN, $t$ denotes step number, $n$ denotes the resolution number. Cardiac NCC setting.

Figure 3. Illustration of iterative optimization in forward pass of GraDIRN, $t$ denotes step number, $n$ denotes the resolution number. Brain SSD setting.
Figure 4. Illustration of iterative optimization in forward pass of GraDIRN, $t$ denotes step number, $n$ denotes the resolution number. Brain NCC setting.
3. Additional Visual Examples of Registration Results

Fig. 5 and Fig. 6 are provided as additional visual examples of registration results comparing competing methods under different data and data fidelity / image dissimilarity settings, supplementing results shown Figure 3 in the main paper.

Figure 5. Cardiac MR registration results comparing competing methods with NCC as data fidelity and/or dis-similarity loss.
Figure 6. Brain MR registration results comparing competing methods with SSD as data fidelity and dis-similarity loss.