Overview

The different sections of the **supplementary material** cover the following aspects of our `GradICON` approach.

- **Appendix A.1** provides a justification of our noise assumptions, details of the derivation of the regularization properties of `GradICON`, and additional insights on the convergence behavior of `GradICON`.
- **Appendix A.2** describes our affine data augmentation strategy in detail.
- **Appendix A.3** provides detailed comparisons of `GradICON` to other regularizers, including the loss curve and examples associated with the experiment in Sec. 5.3.
- **Appendix A.4** describes the details of the experiment in Sec. 5.4 with convergence speed demonstration on **OAI** dataset.
- **Appendix A.5** shows an expanded version of Table 2 from the main manuscript. In particular, this table provides information on the provenance of these validation results.
- **Appendix A.6** provides details on inference times, memory use, and numbers of parameters for some key learning-based registration approaches.
- **Appendix B** shows example registration results for the **OAI**, **HCP**, and **COPDGene** datasets.
- **Appendix C** discusses potentials for negative societal impacts of our work.

A. Supplementary material

A.1. Analysis details

Experiments on the main modeling hypothesis. The main modeling hypothesis in the implicit regularization analysis of Sec. 3.3 is that the noise term n can be neglected in the Taylor expansion $\nabla\Phi^{AB}(\Phi_\theta^{BA}) = \nabla\Phi^{AB}(\Phi^{BA}) + \varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})(n^{BA}) + o(\varepsilon)$. In this formula, we argue in the main text that the noise term n^{BA} can be neglected with respect to ε, which is the scale of the noise on the Jacobian. Indeed, only the low-frequency noise should appear since integration is a low-pass filter, but we expect this low-frequency noise to be dampened by the similarity measure, which is an L^2 norm on the images. On the synthetic dataset, we checked that our hypothesis is valid as a first approximation. From a given output of the network $\Phi_\theta^{AB}, \Phi_\theta^{BA}$, we estimated the closest Φ^{AB}, Φ^{BA} in L^2 norm to our data. Although this estimate is certainly biased, it is the first natural estimator to check our assumption. In Fig. 5, we plot the noise n^{AB} and its corresponding gradient ∇n^{AB} in one chosen *direction* (indicated by the red arrow in the plots on the right-hand side).

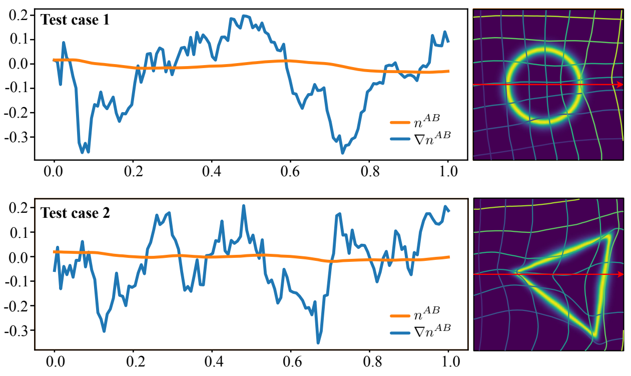

Figure 5. Two *finite difference estimations* of the noise n^{AB} and the gradients ∇n^{AB} on the synthetic dataset. The magnitude of the gradient is an order of magnitude higher which confirms our hypothesis.

Derivation details. Some steps that were omitted in the main text are explained hereafter. Our main object of interest is the `GradICON` regularizer

$$\mathcal{L}_{\text{reg}}^{\texttt{GradICON}} = \left\|\nabla_x[\Phi_\theta^{AB}(\Phi_\theta^{BA}(x))] - \mathbf{I}\right\|_F^2 \quad . \tag{14}$$

In what follows, we use ∇ instead of ∇_x. Making the assumption that the neural network will try to be perfectly inverse consistent in order to minimize Eq. (14), but will make some error due to limited capacity or imprecision in the training process or just because of trying to balance the matching term and the inverse consistency, we decompose each neural network output $\Phi_\theta^{AB}, \Phi_\theta^{BA}$ into two components, a perfectly inverse consistent component Φ^{AB}, and random noise εn^{AB}, *i.e.*,

$$\mathcal{L}_{\text{reg}}^{\texttt{GradICON}} = \left\|\nabla[\Phi^{AB}(\Phi_\theta^{BA}) + \varepsilon n^{AB}(\Phi_\theta^{BA})] - \mathbf{I}\right\|_F^2 \quad . \tag{15}$$

By applying the chain rule, this can be rewritten as

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\texttt{GradICON}} = \big\|&((\nabla\Phi^{AB})(\Phi_\theta^{BA}) + \\ &\varepsilon(\nabla n^{AB})(\Phi_\theta^{BA}))\cdot(\nabla\Phi^{BA} + \varepsilon\nabla n^{BA}) - \mathbf{I}\big\|_F^2 \quad .\end{aligned} \tag{16}$$

Next, we Taylor-expand the term $\nabla\Phi^{AB}(\Phi_\theta^{BA})$, *i.e.*, $\nabla\Phi^{AB}(\Phi^{BA} + \varepsilon n^{BA})$ with respect to ε, yielding

$$\begin{aligned}&\nabla\Phi^{AB}(\Phi_\theta^{BA}) = \\ &\nabla\Phi^{AB}(\Phi^{BA}) + \varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA} + o(\varepsilon) \quad ,\end{aligned} \tag{17}$$

where $\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA}$ is the appropriate tensor product. It is clear that the last term can be dropped in the limit of small ε. Plugging this approximation into Eq. (16), we get

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \big\|(&\nabla\Phi^{AB}(\Phi^{BA}) \\ &+ \varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA} \\ &+ \varepsilon(\nabla n^{AB})(\Phi_\theta^{BA})) \\ &\cdot(\nabla\Phi^{BA} + \varepsilon\nabla n^{BA}) - \mathbf{I}\big\|_F^2 \ .\end{aligned} \tag{18}$$

Upon distributing terms,

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \big\|&\nabla\Phi^{AB}(\Phi^{BA})\cdot\nabla\Phi^{BA} \\ &+ \varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA}\cdot\nabla\Phi^{BA} \\ &+ \varepsilon(\nabla n^{AB})(\Phi_\theta^{BA})\cdot\nabla\Phi^{BA} \\ &+ \nabla\Phi^{AB}(\Phi^{BA})\cdot\varepsilon\nabla n^{BA} \\ &+ \varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA}\cdot\varepsilon\nabla n^{BA} \\ &+ \varepsilon(\nabla n^{AB})(\Phi_\theta^{BA})\cdot\varepsilon\nabla n^{BA} \\ &-\mathbf{I}\big\|_F^2 \ .\end{aligned} \tag{19}$$

The first term equals $\mathbf{I}$ and so cancels with the last term. Further, by assuming small ε, we drop terms in ε^2, yielding

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \big\|&\varepsilon\nabla^2\Phi^{AB}(\Phi^{BA})n^{BA}\cdot\nabla\Phi^{BA} \\ &+\varepsilon(\nabla n^{AB})(\Phi_\theta^{BA})\cdot\nabla\Phi^{BA} \\ &+\nabla\Phi^{AB}(\Phi^{BA})\cdot\varepsilon\nabla n^{BA}\big\|_F^2 \ .\end{aligned} \tag{20}$$

Following arguments above regarding the relative magnitudes of n and ∇n, the first term in Eq. (20) can be neglected, and also ε can be factored out to obtain

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \varepsilon^2\|&\nabla n^{AB}(\Phi_\theta^{BA})\nabla\Phi^{BA} \\ &+ \nabla\Phi^{AB}(\Phi^{BA})\nabla n^{BA}\|_F^2 \ .\end{aligned} \tag{21}$$

Now, we justify and then use the approximation

$$\nabla n^{AB}(\Phi_\theta^{BA}) = \nabla n^{AB}(\Phi^{BA}) + O(\varepsilon) \ . \tag{22}$$

To proceed, we make the following remarks. Inversion of the map preserves a first-order expansion in ε, *i.e.*,

$$[\Phi_\theta^{AB}]^{-1} = \Phi^{BA} - \varepsilon\nabla\Phi^{BA}(n^{AB}(\Phi^{BA})) + o(\varepsilon) \ , \tag{23}$$

which can be checked by composition. A similar first-order expansion holds for the Jacobian determinant, *i.e.*,

$$\text{Det}(\nabla[\Phi_\theta^{AB}]^{-1}) = \text{Det}(\nabla[\Phi^{AB}]^{-1}) + O(\varepsilon) \ . \tag{24}$$

As a consequence, for a differentiable function (possibly vector valued) I, we have

$$I([\Phi_\theta^{BA}]^{-1}(x)) = I([\Phi^{BA}]^{-1}(x)) + O(\varepsilon)\,, \tag{25}$$

by Eq. (23) and first-order Taylor expansion of I. We combine the three above equations in what follows. Since ∇n^{AB} is a white noise, this formula is justified in the following sense. For a given vector-valued differentiable function I on the image domain Ω, we have

$$\begin{aligned}&\int_\Omega \nabla n^{AB}((\Phi_\theta^{BA}(x)))\cdot I(x)\,dx \\ &= \int_\Omega \nabla n^{AB}(x)\cdot I([\Phi_\theta^{BA}]^{-1}(x))\,\text{Det}(\nabla[\Phi_\theta^{BA}]^{-1}(x))\,\mathrm{d}x \\ &= \int_\Omega \nabla n^{AB}(x)\cdot I([\Phi_\theta^{BA}]^{-1}(x))\,\text{Det}(\nabla[\Phi^{BA}]^{-1}(x))\,\mathrm{d}x + O(\varepsilon) \\ &= \int_\Omega \nabla n^{AB}(\Phi^{BA})\cdot I(x)\,\mathrm{d}x + O(\varepsilon) \ ,\end{aligned} \tag{26}$$

where, in the first and the last equation, we used the change of variable formula. This brings us to

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \varepsilon^2\|&\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA} \\ &+ \nabla\Phi^{AB}(\Phi^{BA})\nabla n^{BA}\|_F^2 \ ,\end{aligned} \tag{27}$$

which is Eq. (7) in the main text. We now expand the square to get

$$\begin{aligned}\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \varepsilon^2\big(\ &\|\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA}\|_F^2 \\ &+ \|\nabla\Phi^{AB}(\Phi^{BA})\nabla n^{BA}\|_F^2 \\ + 2\langle\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA},\ &\left[\nabla\Phi^{AB}\right](\Phi^{BA})\nabla n^{BA}\rangle_F\ \big)\end{aligned} \tag{28}$$

and an application of the fact that Φ^{AB} and Φ^{BA} are inverses of each other gives

$$\begin{aligned}&\mathcal{L}_{\text{reg}}^{\text{GradICON}} \approx \\ &\varepsilon^2\Big(\left\|\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA}\right\|_F^2 + \left\|\left[\nabla\Phi^{BA}\right]^{-1}\nabla n^{BA}\right\|_F^2 \\ &\quad + 2\langle\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA},\left[\nabla\Phi^{BA}\right]^{-1}\nabla n^{BA}\rangle_F\Big)\,.\end{aligned} \tag{29}$$

When *taking expectation* in Eq. (9) of the main text, the white noise independence assumption implies that

$$\mathbb{E}[\nabla n_i^{BA}(y)\nabla n_j^{AB}(x)] = 0 \tag{30}$$

for all coordinates i, j and $x, y \in \Omega$; this explains that the cross-term vanishes. Thus, we arrive at

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\text{reg}}^{\text{GradICON}}] \approx \mathbb{E}\Big[\varepsilon^2\Big(&\left\|\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA}\right\|_F^2 + \\ &\left\|\left[\nabla\Phi^{BA}\right]^{-1}\nabla n^{BA}\right\|_F^2\Big)\Big]\,.\end{aligned}$$

Now, to further simplify Eq. (9), we use the fact that $\mathbb{E}[\nabla n_i^{AB}(x)\nabla n_j^{AB}(x)] = \delta_{ij}$ where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$. A direct computation already gives the second term of Eq. (10) from the main text, *i.e.*,

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\text{reg}}^{\text{GradICON}}] \approx \mathbb{E}\Big[\varepsilon^2\Big(&\left\|\nabla n^{AB}(\Phi^{BA})\nabla\Phi^{BA}\right\|_F^2 + \\ &\left\|\left[\nabla\Phi^{BA}\right]^{-1}\right\|_F^2\Big)\Big]\,.\end{aligned} \tag{31}$$

In order to obtain the first term of Eq. (10) from Eq. (9) in the main text, one needs to use a change of variables $y = \Phi^{AB}(x)$ in space, which results in the appearance of the determinant of the Jacobian matrix, denoted by $\text{Det}(\nabla\Phi^{AB})$, and then similarly use the white noise assumption. The first term of Eq. (10) has a square root since it is put inside the squared Frobenius norm. Overall, we arrive at Eq. (10) from the main text, *i.e.*,

$$\mathbb{E}[\mathcal{L}_{\text{reg}}^{\texttt{GradICON}}] \approx \varepsilon^2 \Bigg(\Big\| \left[\nabla \Phi^{AB}\right]^{-1} \sqrt{\text{Det}(\nabla \Phi^{AB})} \Big\|_F^2 + \Big\| \left[\nabla \Phi^{BA}\right]^{-1} \Big\|_F^2 \Bigg) \ . \tag{32}$$

`GradICON` and preconditioning. Recall that in our notation $\psi(x) = \Phi^{AB}(\Phi^{BA}(x)) - \text{Id}$. The `ICON` formulation uses $\|\psi\|_{L^2}^2$, whereas the `GradICON` formulation uses $\|\nabla\psi\|_{L^2}^2$. Our goal is to understand the effect of this change on the optimization scheme. The parameters of the neural networks encoding the map from A, B to Φ^{AB} are optimized to minimize the overall loss but let us focus on the inverse consistency loss. For each pair A, B, automatic differentiation computes the gradient of the loss with respect to ψ, which is then backpropagated. Computing the gradient of the `GradICON` loss can be done by rewriting

$$\|\nabla\psi\|_{L^2}^2 = -\langle \psi, \Delta\psi\rangle_{L^2} \ , \tag{33}$$

where Δ is the Laplacian. Hence, the gradient is $-2\Delta\psi$, which is also called change of metric or *preconditioning*. This gradient has a particularly clear formulation in Fourier space (denoting by $\hat{f}(\omega)$ the Fourier transform of $f(x)$) since it reads as

$$\widehat{\Delta\psi}(\omega) = -|\omega|^2 \hat{\psi}(\omega) \tag{34}$$

and has to be compared with the gradient of the `ICON` loss which is $\hat{\psi}$. Low frequencies are thus damped in comparison to high frequencies. For instance, the gradient flows (steepest descent in cont. time) of the two regularizers are

$$\begin{aligned} \partial_t \psi(x) &= -\psi(x) \ \text{ and} \\ \partial_t \psi(x) &= \Delta\psi(x) \ . \end{aligned} \tag{35}$$

Please note $\psi(x)$ on the top is the ψ in `ICON` loss while the one on the bottom corresponds to the ψ in `GradICON` loss. In Fourier space, these flows become

$$\begin{aligned} \partial_t \hat{\psi}(\omega) &= -\hat{\psi}(\omega) \ \text{ and} \\ \partial_t \hat{\psi}(\omega) &= -|\omega|^2 \hat{\psi}(\omega) \ . \end{aligned} \tag{36}$$

Here, the main difference is exponential convergence of the first gradient flow while for the second one (related to `GradICON`), the rate of exponential convergence depends on $|\omega|^2$, *i.e.*, faster convergence occurring for high spatial frequencies ω. We now still need to understand why such a preconditioning is beneficial for the overall goal of the diffeomorphic registration problem. From the discussion above, it is clear that low-frequency perturbations are less penalized than high-frequency perturbations. The simplest example is the case of constant perturbations which are not penalized at all by the penalty on the gradient.

The purpose of inverse consistency is to encourage each one of the transformations Φ^{AB} and Φ^{BA} to be a bijective map. However, a relaxation of this loss (in the sense of a relaxation of a constraint) which still encourages invertible maps can be beneficial in the context of diffeomorphic registration. A possible idea consists in relaxing the constraint of ψ to be a deformation close to identity, while still being diffeomorphic. As a consequence, if both $\Phi^{AB}(\Phi^{BA})$ and $\Phi^{BA}(\Phi^{AB})$ are invertible, then both transformation are also invertible. In fact, constraining these two compositions to be any diffeomorphism also leads to the same conclusion.

Further, it is well-known that perturbation of identity by a map with a small Lipschitz constant remains a diffeomorphism. It is the result of the inverse function theorem, explained in more detail below. Small variations around identity by a Lipschitz map can be written as $\text{Id} + v$ with $v : \mathbb{R}^d \to \mathbb{R}^d$ Lipschitz. We want these perturbations of identity to remain diffeomorphic and, in particular, injective (equivalent to asking for no foldings). A sufficient condition to satisfy injectivity is that $\|v(x) - v(y)\| \leq \varepsilon\|x - y\|$ (*i.e.*, v is ε-Lipschitz) with $\varepsilon < 1$. When v is C^1, the Lipschitz inequality reduces to saying that $\|\nabla v(x)\| \leq \varepsilon$ for every point x in the domain. In fact, as shown by the inverse function theorem, this condition is also sufficient for $\text{Id} + \varepsilon v$ to be a diffeomorphism. Recall in our case, the deviation to identity is denoted by ψ. In view of this sufficient condition, one would ideally penalize the maximum value of $\|\nabla\psi(x)\|$.

To sum up, in order to control the invertibility of $\Phi^{AB}(\Phi^{BA}(x))$, it is better to control $\nabla\psi$ rather than ψ itself. Last, let us show on a concrete example that the `ICON` regularizer can be more constrained than the `GradICON` regularizer. To this end, note that *constant* shifts around identity are still invertible maps but `ICON` penalizes too large constant shifts, while `GradICON` does not at all. Also, having small `ICON` loss does not guarantee invertibility of the resulting maps. Indeed, there are maps v with a small L^2 norm for which the magnitude of the gradient may be larger than 1, thereby potentially leading to folds. Penalizing the maximum value of the norm of the gradient of ψ is better suited to guarantee invertibility when the loss is less than 1. However, in practice, this loss has a lack of differentiability; using an L^2 loss is much simpler, more convenient, and retains some of the nice properties mentioned above.

A.2. Affine data augmentation details

When we train using affine augmentation, first, we sample a new pair of images from the dataset. Then, we randomly choose whether the image is flipped along each axis: these choices are shared between images in the pair. Finally, independently for each image in the pair, we sample a 3×4 matrix (with each entry i.i.d. from a standard Gaussian, denoted as $\mathcal{N}^{(3,4)}(0, 1)$) that represents an affine warp using homogeneous coordinates. This produces an augmented image $\hat{I}$, *i.e.*,

$$\hat{I}(\vec{x}) = I\left(\left(\begin{bmatrix} u_1 & 0 & 0 & 0 \\ 0 & u_2 & 0 & 0 \\ 0 & 0 & u_3 & 0 \end{bmatrix} + \gamma \cdot \mathcal{N}^{(3,4)}(0,1)\right)\begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}\right)$$

where $u_i \sim \text{Uniform}\{\pm 1\}$ and $\gamma = 0.05$.

A.3. Comparison to other regularizers

In Sec. 5.3 of our main text, we conducted a comparison between different regularizers using displacement vector fields (DVF). However, Fig. 3 only shows *aggregated* results. In Fig. 6 and Fig. 7, we present one registration example from the **Triangles and Circles** and **DRIVE** data, respectively. Fig. 8 and Fig. 9 show the image similarity measure, the number of folds and the mean of the squared L_2-norm on the displacement vector field plotted over training iterations and with varying regularization weight.

A.4. Convergence of Gradient Inverse Consistency

We conduct the experiment in Sec. 5.4 based on our experimental result of Fig. 3. In particular, we draw a horizontal line in Fig. 3 at $\%|J| = 10^{-2}$ and find the closest point to the line on the `GradICON` and `ICON` curve. We then plot the loss curves associated with these points in Fig. 4.

As can be seen from Fig. 4, there is a clear convergence speed difference between `GradICON` and `ICON` on the two 2D datasets. While a similar study on actual 3D data would be interesting, training 13 models with varying λ is computationally challenging. Nevertheless, to obtain some intuition about convergence speed differences on 3D data, we present in Fig. 10 the training loss curve, as well as the transform magnitude and the (log) number of folds, of `GradICON` and `ICON` corresponding to the **OAI** dataset in Tab. 2. Figure 10 clearly supports our claim of faster convergence of models trained with `GradICON` regularization over models trained with `ICON` regularization. Further, it can be seen that `GradICON` exhibits lower similarity loss and shows larger transform magnitudes because it better captures the large deformations in the **OAI** dataset.

A.5. Details for comparisons in Table 2

In this section, we 1) present Table 3, which is a complete version of Table 2, and 2) describe the experimental details for comparisons in Table 2.

sm-shapes and sm-brains. We evaluate the SynthMorph [31] model with pre-trained weights from its official repository[15] on the same **HCP** test set we use for `GradICON` and follow the suggested testing protocol from the repository. We first run **FreeSurfer-Affine** (see the following Freesurfer-Affine paragraph) to align the source and target image to the reference image provided in the repository. Then, we run *SynthMorph-shapes* (`sm-shapes`) and *SynthMorph-brains* (`sm-brains`) models to obtain the deformation between the pre-aligned source and target images. To compute the final transformation field used to warp the original source label map to the target label map, we first generate an identity map in the target image space and transform it via the target-to-reference affine matrix. Then, we compose the transformed map with the deformation field computed by SynthMorph. Lastly, we transform the composed deformation field via the reference-to-source affine matrix (obtained by inverting the source-to-reference affine matrix). Eventually, we use the final computed deformation field to warp the original source label map and then compute the DICE between the warped label map and the target label map in the *original* target space.

FreeSurfer-Affine. We report the affine pre-alignment result from our SynthMorph experiment and label it as FreeSurfer-Affine [48] in Table 2. FreeSurfer is run with the configuration recommended in the SynthMorph repository. For evaluation purposes, we compose the target-to-reference affine matrix and reference-to-source affine matrix the same way as we did in the SynthMorph experiment except that we skip the step to compose the deformation field computed by SynthMorph. Essentially, we simply assume that the non-parametric part that would have been obtained by SynthMorph is set to the identity transform thereby only leaving the affine registrations. This experiment differs from directly obtaining an affine transformation between the source and the target spaces as instead we go through the template space and compute two affine transformations. However, this choice of affine transform composition allows a more direct assessment of the improvements obtained by SynthMorph.

`ICON`. We follow a similar design as described in [23] and, in particular, adopt the `tallUNet2` architecture as the backbone network. Specifically, a composition of two such UNet's is initially trained on half-resolution image pairs. This network is then composed with a third UNet, trained on full-resolution image pairs.

VoxelMorph. We use the official code from the VoxelMorph repository[15] and train on **COPDGene**. As VoxelMorph requires pre-registration, we train another neural network for affine pre-registration. Table 4 shows the registration accuracy of this affine registration network. For VoxelMorph, we use NCC as the similarity measure, set the learning rate to 1e-3, and the regularizer weight to 5. We keep all other settings at the provided default values. Since the official code provides the inverse transformation, the bi-directional registration result is obtained with the forward map and its inverse map is computed by the official code.

LapIRN. We obtain the network from the official reposi-

[15]https://github.com/voxelmorph/voxelmorph

tory[16] of LapIRN and train it on **COPDGene**. In particular, we train using the training script provided by the official repository with hyperparameter tuning for **COPDGene** data. We switched from LNCC to NCC because we observed unstable training with the LNCC implementation provided in the official LapIRN repository. For each resolution, we adjust the learning rate and λ to assure that the training converges. Table 5 provides the hyperparameters we used to obtain the results in Table 2. We randomly swap the source and target images during training so that the trained network can work for bi-directional registration.

A.6. Model statistics

We compute model statistics regarding the number of parameters, peak memory use, FLOPs, and inference time using built-in PyTorch[17] functions and the `thop`[18] package. This experiment is conducted using an NVIDIA GeForce RTX 3090 GPU with a batch size of 1 and randomly generated image pairs of size $175 \times 175 \times 175$. We run the model 10 times and take the average of the elapsed time as the final measurement. In addition, the peak GPU memory usage is reported for each model. Table 6 and Table 7 list the statistics of models evaluated in Table 2 and the UNet used in our ablation study of Sec. 5.2. Working on 2-D convolutional networks builds a strong intuition that if, in a downsampling step, we double the number of channels and cut in half the resolution, the amount of computation stays roughly constant. This intuition is not correct for 3-D networks. In fact, for 3-D networks, doubling the number of channels and halving the resolution cuts the amount of computation by 1/2. As a result, large channel counts deep in the network are, from a computation time and VRAM perspective, free. The UNet from `ICON` approach [23] takes advantage of this effect to boost performance using a large parameter count while reducing runtime and VRAM usage compared to the standard VoxelMorph channel counts. We used the same approach for our registration networks using `GradICON`. Note that Table 6 illustrates that even though `ICON` and `GradICON` use about 50 times more parameters than LapIRN and roughly 150 times more parameters than VoxelMorph, memory consumption and inference times are in fact lower.

B. Visualizations

In Fig. 11 and Fig. 12, we show two example registration cases from **OAI**, Fig. 13 and Fig. 14 show two example registration cases on **DirLab**, and Fig. 15 and Fig. 16 show two example registration cases on **HCP**.

In Fig. 17 we show a block diagram of the network structure described in Sec. 4, that is more detailed.

[16]https://github.com/cwmok/LapIRN
[17]https://pytorch.org/
[18]https://github.com/Lyken17/pytorch-OpCounter

C. Potential negative societal impacts

Image registration results might not be accurate or might even fail for certain image pairs in practice. Hence, careful quality control of the results should be performed when registrations are used for decision-support systems in a medical context.

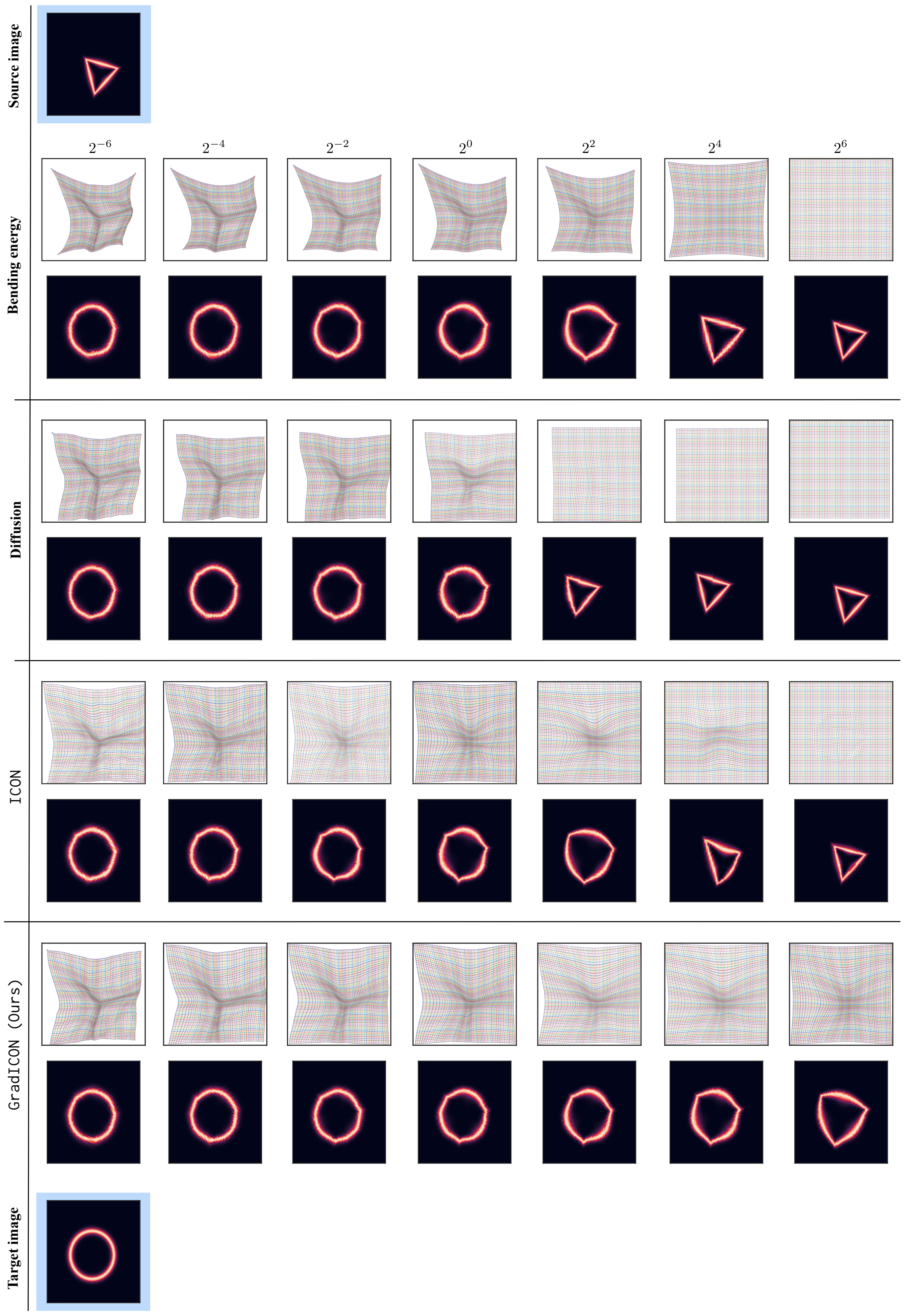

Figure 6. Illustration of one *warped source image* and the corresponding transformation maps for different regularizers across varying regularization strengths on **Triangles and Circles**. *Best-viewed in color.*

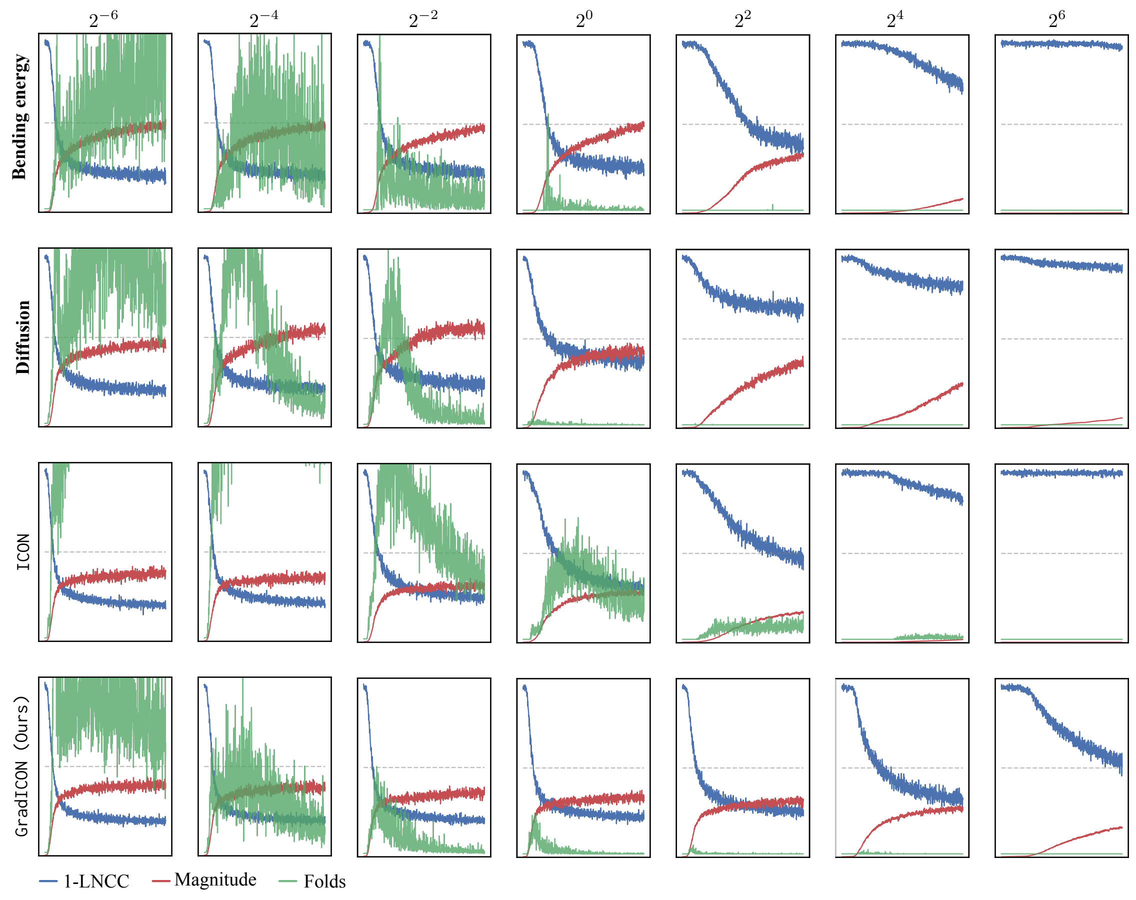

Figure 7. Illustration of image (dis)similarity (*i.e.*, $1 - \text{LNCC}$), the number of folds (Folds), and the mean of the squared L^2 norm of the displacement vector field (Magnitude) for different regularizers and across varying regularization strengths on **Triangles and Circles**. *Best-viewed in color.*

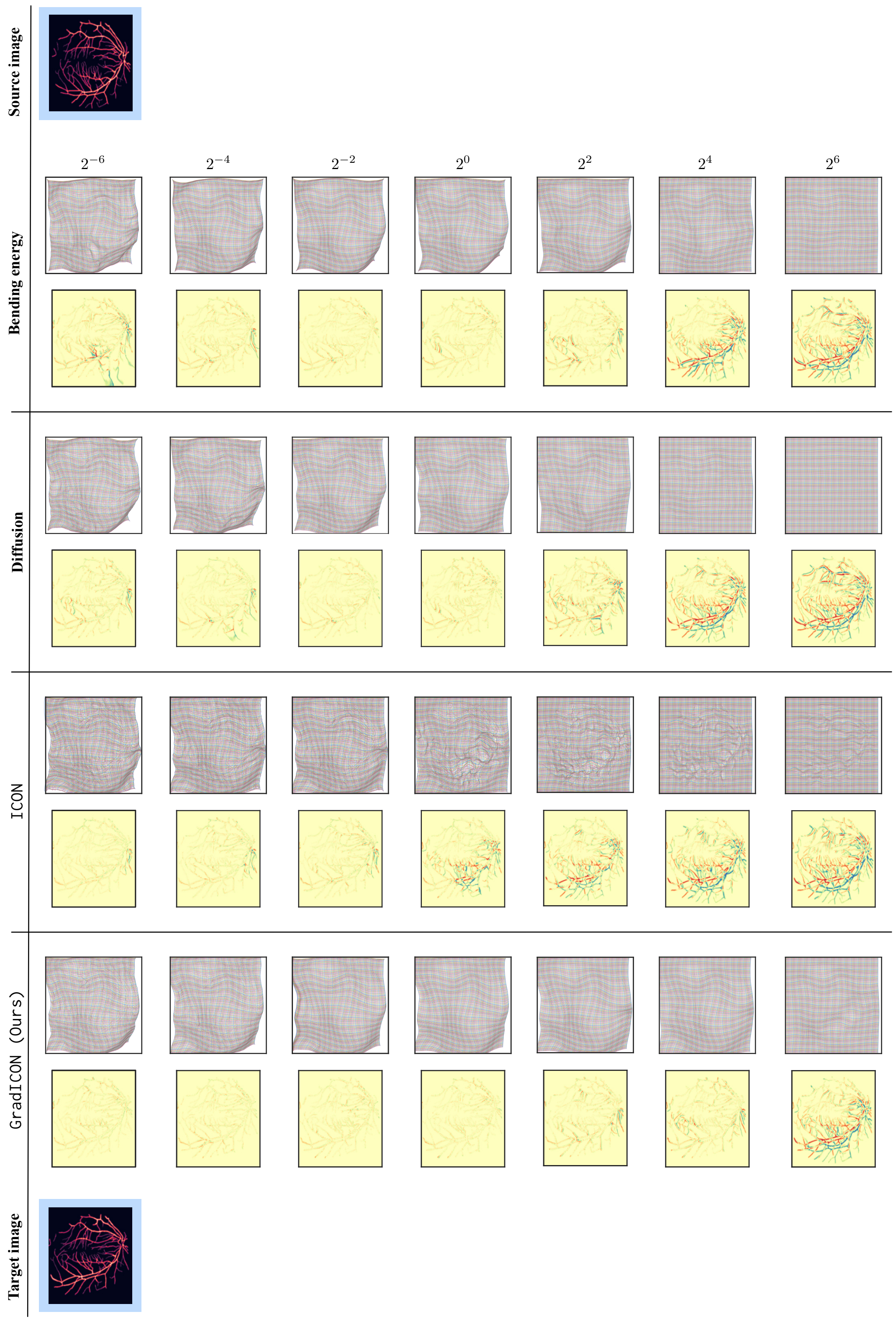

Figure 8. Illustration of the *residual error* and the corresponding transformation maps between the *warped source image* and the *target image* for different regularizers across varying regularization strengths on **DRIVE**. *Best-viewed in color.*

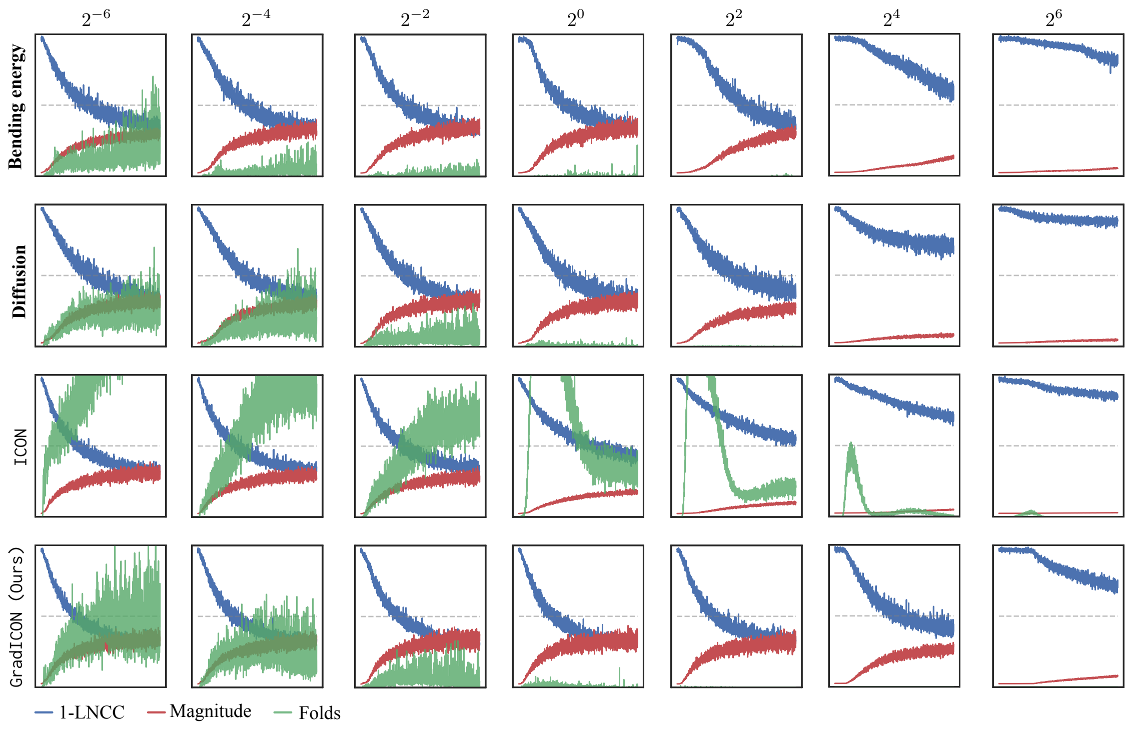

Figure 9. Illustration of image (dis)similarity (*i.e.*, $1 - \text{LNCC}$), the number of folds (Folds), and the mean of the squared L^2 norm of the displacement vector field (Magnitude) for different regularizers and across varying regularization strengths on **DRIVE**. *Best-viewed in color.*

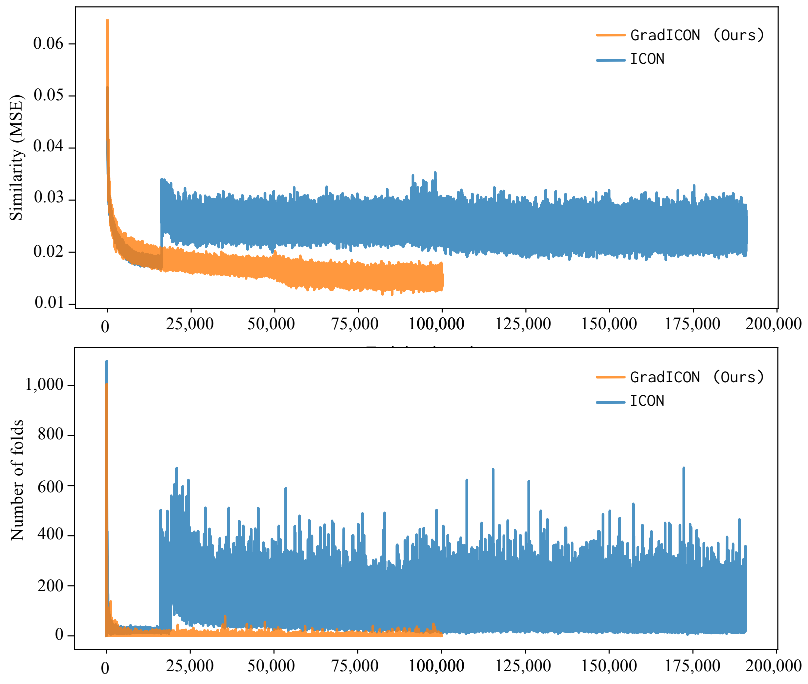

Figure 10. Image similarity and the number of folds, plotted over training iterations for the `ICON` and `GradICON` (MSE, $\lambda = 0.2$) entries in the **OAI** section of Tab. 3. `ICON`'s parity in similarity loss early in training is illusory, as unlike our approach it is trained progressively, and so during these iterations, it is still being trained at low resolution. This leads to a lower MSE during this phase, as the MSE demands more precise alignment on higher resolution (and hence not low-pass filtered) input images. Once both networks are training at the final resolution, the values are directly comparable. These results demonstrate the faster convergence rate, regularity, and final performance of `GradICON`.

Method	**Trans.**	$\mathcal{L}_{\text{reg}}$	$\mathcal{L}_{\text{sim}}$	**DICE** $\uparrow$	$\%\lvert J\rvert \downarrow$	Reported by
			OAI			
Initial				7.6		
Demons [62]	A,DVF	Gaussian	MSE	63.5	0.0006	[52]
SyN [3]	A,VF	Gaussian	LNCC	65.7	0.0000	[52]
NiftyReg [43]	A,B-Spline	BE	NMI	59.7	0.0000	[52]
NiftyReg [43]	A,B-Spline	BE	LNCC	**67.9**	0.0068	[52]
vSVF-opt [52]	A,vSVF	m-Gauss	LNCC	67.4	0.0000	[52]
VM [4]	SVF	Diff.	MSE	46.1	0.0028	[52]
VM [4]	A,SVF	Diff.	MSE	66.1	0.0013	[52]
AVSM [52]	A,vSVF	m-Gauss	LNCC	68.4	0.0005	[52]
`ICON` [23]	DVF	`ICON`	MSE	65.1	0.0040	$*$
Ours (MSE, λ=0.2)	DVF	`GradICON`	MSE	69.5	0.0000	$*$
Ours (MSE, λ=0.2, Opt.)	DVF	`GradICON`	MSE	70.5	0.0001	$*$
Ours *(std. protocol)*	DVF	`GradICON`	LNCC	70.1†	0.0261	$*$
	DVF	`GradICON`	LNCC	**71.2**‡	0.0042	$*$
			HCP			
Initial				53.4		
FreeSurfer-Affine [48]	A	—	TB	62.1	0.0000	$*$
SyN [3]	A,VF	Gaussian	MI	**75.8**	0.0000	$*$
sm-shapes [31]	A,SVF	Diff.	DICE	79.8	0.2981	$*$
sm-brains [31]	A,SVF	Diff.	DICE	78.4	0.0364	$*$
Ours *(std. protocol)*	DVF	`GradICON`	LNCC	78.7†	0.0012	$*$
	DVF	`GradICON`	LNCC	**80.5**‡	0.0004	$*$
			DirLab			
Method	**Trans.**	$\mathcal{L}_{\text{reg}}$	$\mathcal{L}_{\text{sim}}$	**mTRE** $\downarrow$ [mm]	$\%\lvert J\rvert \downarrow$	
Initial				23.36		
SyN [3]	A,VF	Gaussian	LNCC	1.79	—	[26]
Elastix [38]	A,B-Spline	BE	MSE	1.32	—	[26]
NiftyReg [43]	A,B-Spline	BE	MI	2.19	—	[26]
PTVReg [65]	DVF	TV	LNCC	0.96	—	[65]
RRN [28]	DVF	TV	LNCC	**0.83**	—	[28]
VM [4]	A,SVF	Diff.	NCC	9.88	0	$*$
LapIRN [45]	SVF	Diff.	NCC	2.92	0	$*$
LapIRN [45]	DVF	Diff.	NCC	4.24	0.0105	$*$
Hering et al. [30]	DVF	Curv+VCC	DICE+KP+NGF	2.00	0.0600	[30]
GraphRegNet [26]	DV	—	MSE	1.34	—	[26]
PLOSL [66]	DVF	Diff.	TVD+VMD	3.84	0	[66]
PLOSL$_{50}$ [66]	DVF	Diff.	TVD+VMD	1.53	0	[66]
`ICON` [23]	DVF	`ICON`	LNCC	7.04	0.3792	$*$
Ours *(std. protocol)*	DVF	`GradICON`	LNCC	1.26†	0.0003	$*$
	DVF	`GradICON`	LNCC	**0.96**‡	0.0002	$*$

Table 3. Full comparison on **OAI**, **HCP** and **DirLab**. † and ‡ indicate results from our standard training protocol, without (†) and with (‡) instance optimization (Sec. 4.2). Only when `GradICON` is trained with MSE, we set $\lambda = 0.2$. *Top* and *bottom* table parts denote non-learning and learning-based methods, resp. For **DirLab**, results are shown in the common *inspiration→expiration* direction. Results marked with * are reported by us using code from the official repository. A: affine pre-registration, BE: bending energy, MI: mutual information, DV: displacement vector of sparse key points, TV: total variation, Curv: curvature regularizer, VCC: volume change control, NGF: normalized gradient flow, TVD: sum of squared tissue volume difference, VMD: sum of squared vesselness measure difference, Diff: diffusion, VF: velocity field, SVF: stationary VF, DVF: displacement vector field. PLOSL$_{50}$: 50 iterations of instance optimization with PLOSL.

	mTRE ↓	**DICE** ↑
Affine	13.715	80.23

Table 4. Registration performance measures of the pre-registration *affine* network for our VoxelMorph comparison on **DirLab**.

Resolution	LapIRN (disp)		LapIRN (sVF)	
	LR	**reg. weight**	**LR**	**reg. weight**
1	$1e^{-4}$	0.1	$1e^{-4}$	0.1
$1/2$	$5e^{-5}$	0.1	$1e^{-4}$	0.1
$1/4$	$1e^{-5}$	1	$5e^{-5}$	1

Table 5. Learning rate (LR) and regularization weight (reg. weight) hyperparameters of LapIRN per resolution.

	#Params	Inference			Train	
		Peak Mem. (MB)	**FLOPs**	**Time** (ms)	**Peak Mem.** (MB)	**Time** (ms)
VM (SVF)	327,331	4548	397.878G	190.10	—	—
LapIRN (SVF)	923,748	5578	652.310G	253.87	—	—
LapIRN (DV)	923.748	5576	652.310G	235.30	—	—
`ICON`	53,010,687	2918	678.513G	96.36	8082	573.57
`GradICON-Stage1`	53,010,687	2934	618.592G	88.24	9384	727.77
`GradICON-Stage1&Stage2`	70,680,916	3122	1.159T	160.59	13482	1162.54

Table 6. Model statistics at inference (test) time. G denotes gigaFLOPS, T denotes teraFLOPS.

	#Params	**Peak Mem.** (MB)	**FLOPs**
UNet from [4]	327,331	4182.0	397.878G
UNet from [23]	17,670,229	2244	540.084G

Table 7. Model statistics of the UNets used in our ablation study. G denotes gigaFLOPS.

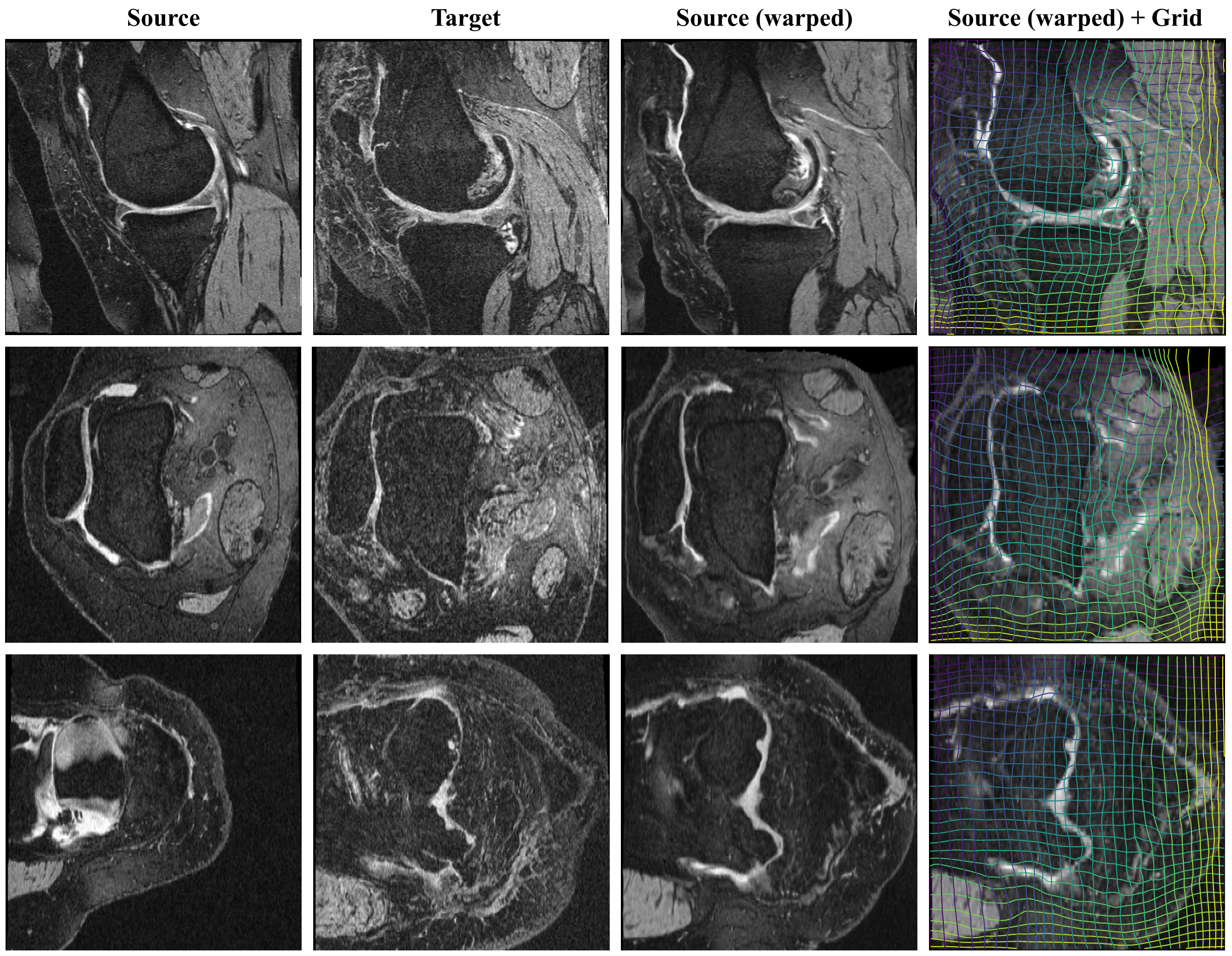

Figure 11. Example registration case **A** (from test set instances) performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **OAI** dataset. *Best-viewed in color.*

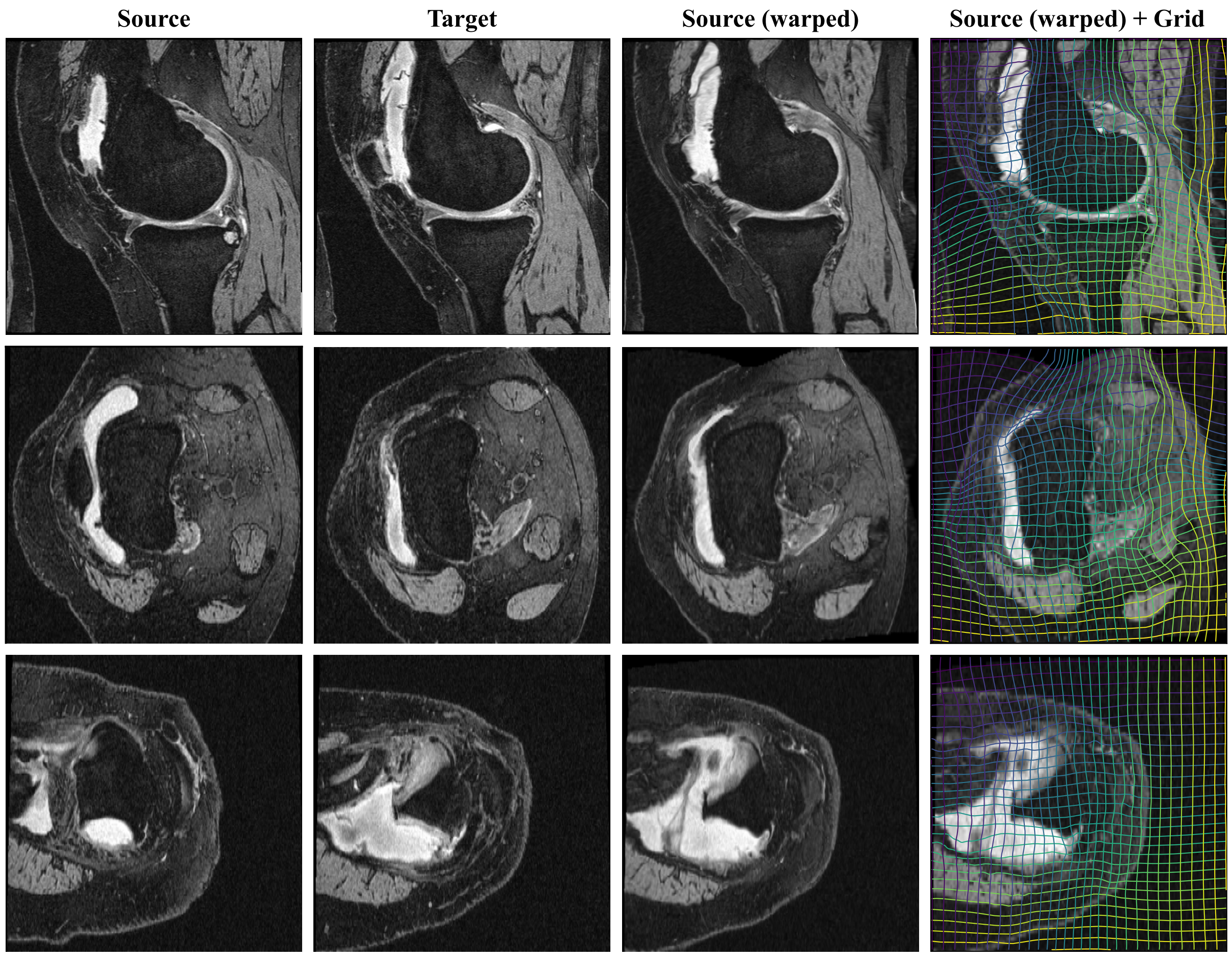

Figure 12. Example registration case **B** (from test set instances) performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **OAI** dataset. *Best-viewed in color.*

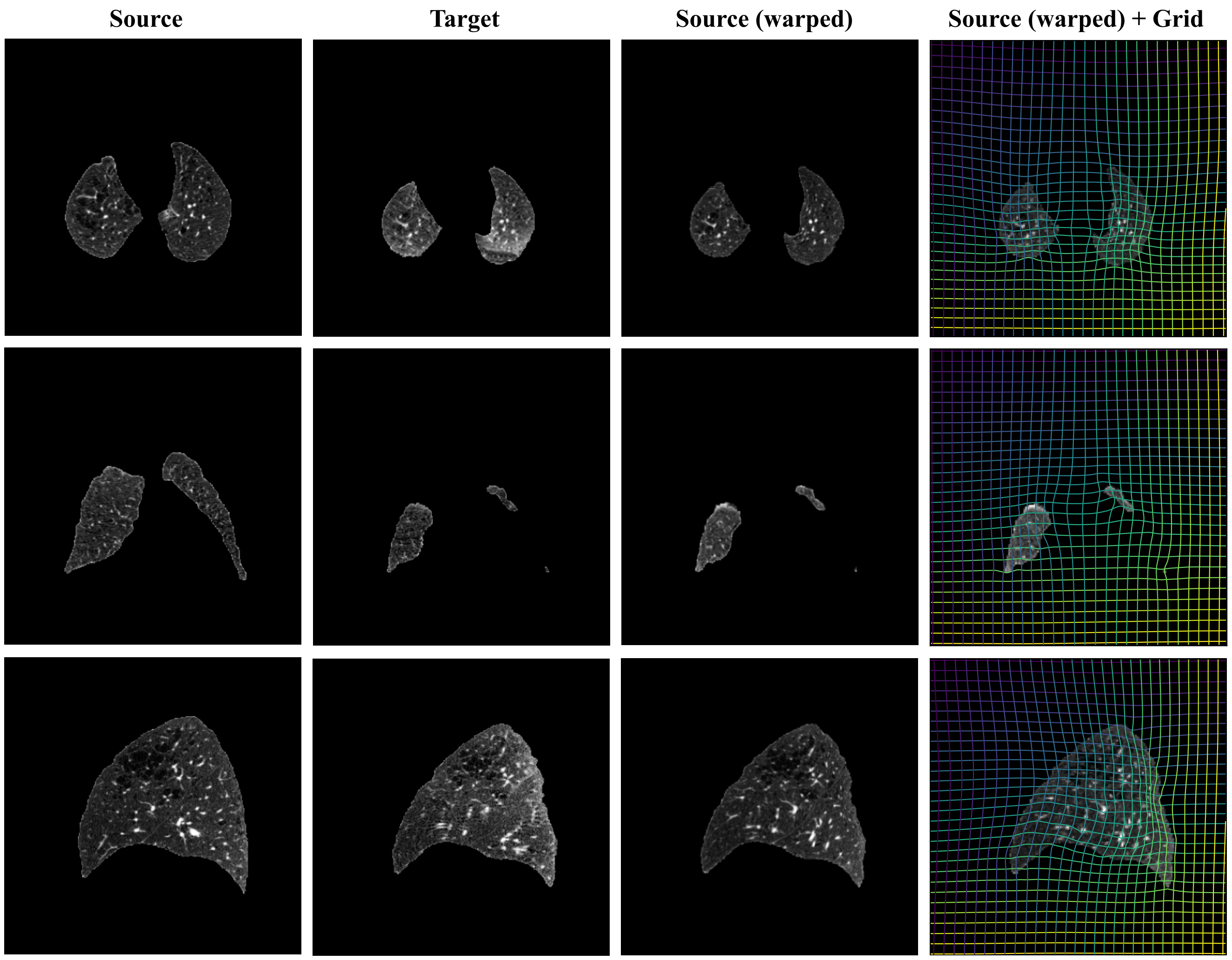

Figure 13. Example registrations case **A** performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **Dirlab** dataset. *Best-viewed in color.*

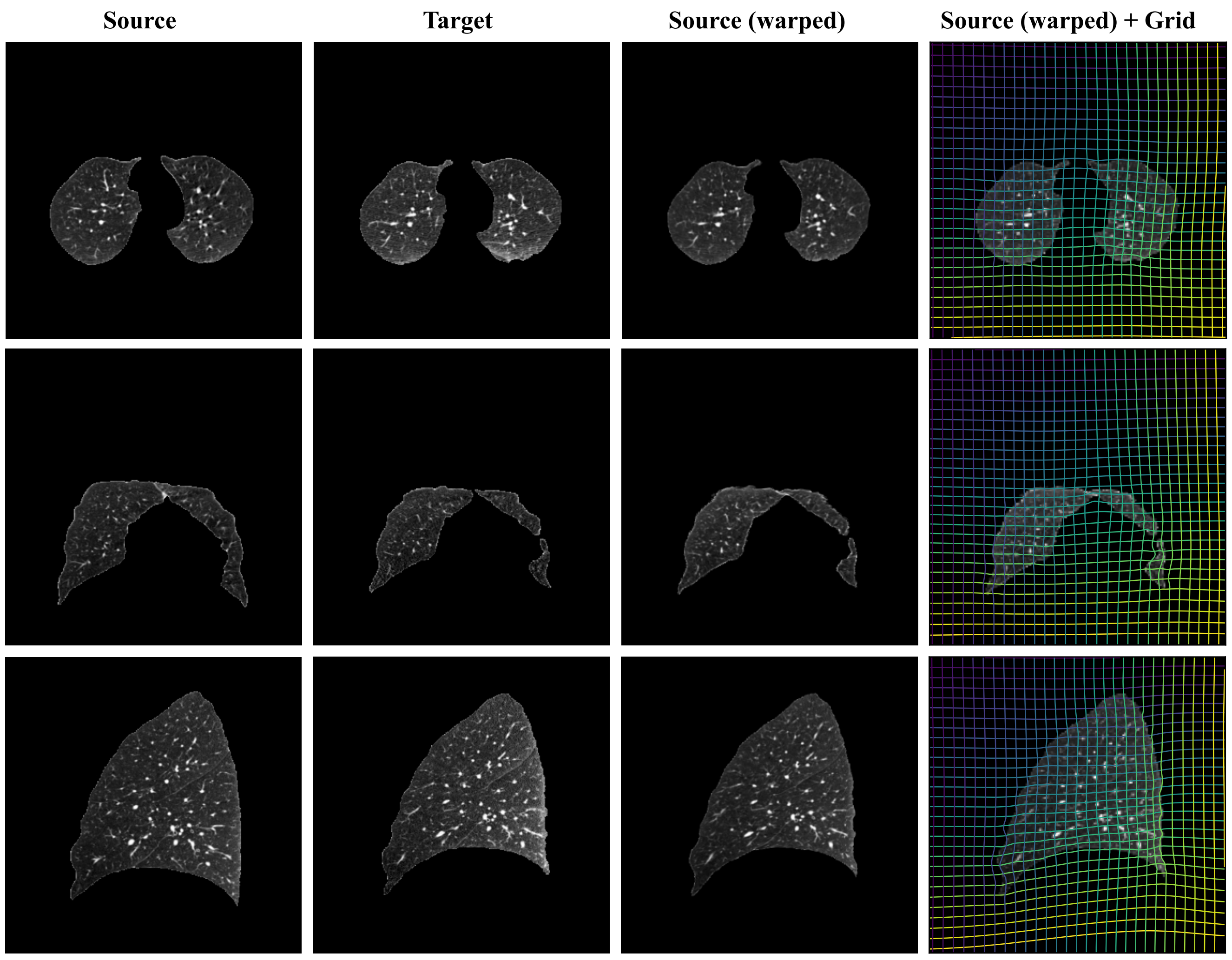

Figure 14. Example registrations case **B** performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **Dirlab** dataset. *Best-viewed in color.*

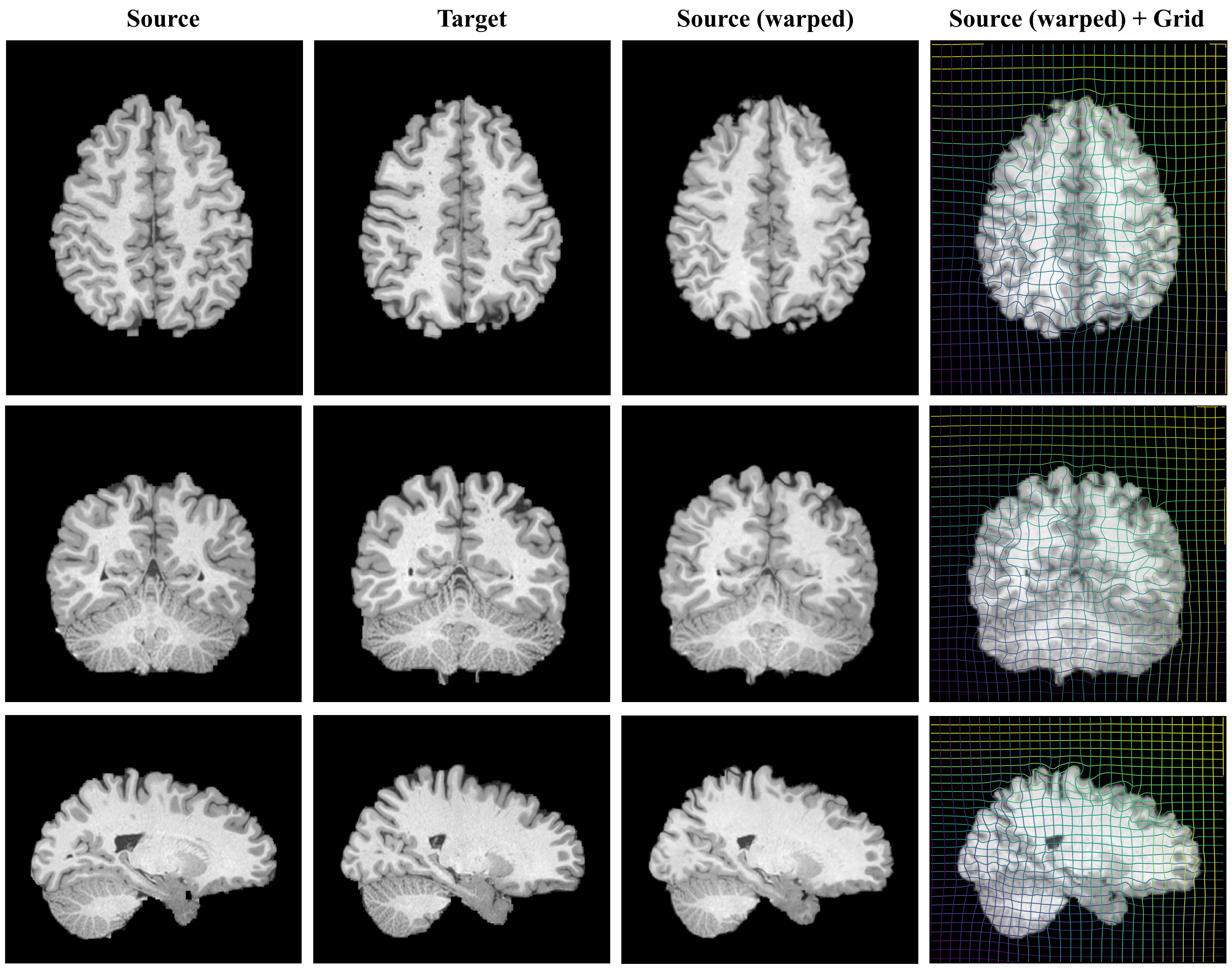

Figure 15. Example registration case **A** (from test set instances) performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **HCP** dataset. *Best-viewed in color.*

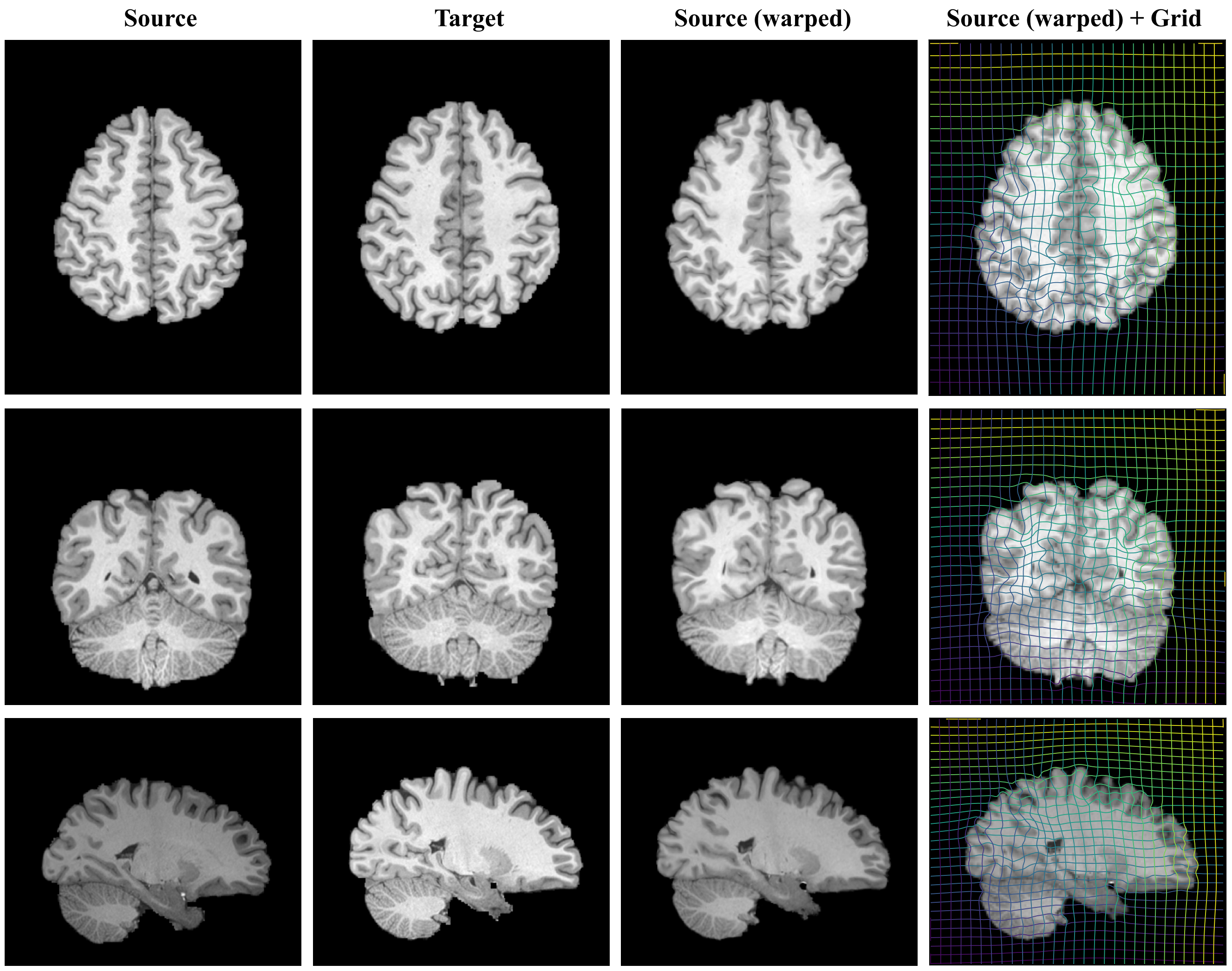

Figure 16. Example registration case **B** (from test set instances) performed using `GradICON` and our standard training protocol (†) w/o instance optimization on the **HCP** dataset. *Best-viewed in color.*

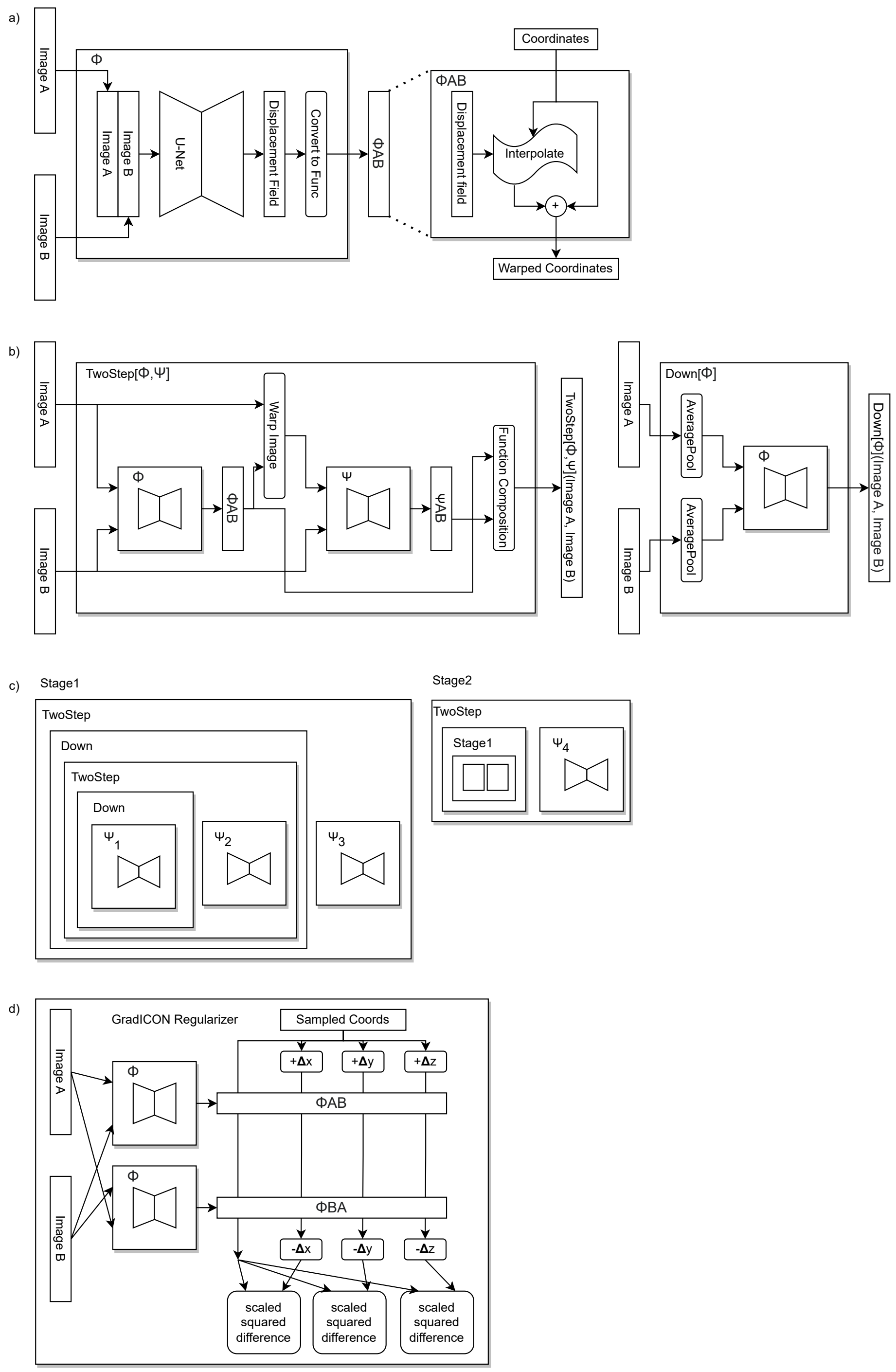

Figure 17. Our approach is most succinctly described using equations, as done in Sec. 4 , but we also desire to respect the convention that neural network papers include a representation of the network as a block diagram. Our "atomic," or simplest component registration network is a U-Net outputting a deformation (a). Φ^{AB}, the output of this component, is a *python function* that may be called on a tensor of coordinates. Components can be combined using the `TwoStep` and `Down` operators (b). The 'function composition' block in this row is implemented by the python code `lambda coords: phi_AB(psi_AB(coords))` , which is pleasing enough to justify our decision to represent deformations as functions. These parts are combined into the Stage1 and Stage2 networks we use for our general purpose registration approach (c). Finally, this network is regularized by a finite difference approximation of the gradient of the inverse consistency error (d)