ICON: Supplementary Material

Hastings Greer
Department of Computer Science
UNC Chapel Hill, USA
tgreer@cs.unc.edu

Roland Kwitt
Department of Computer Science
University of Salzburg, Austria
Roland.Kwitt@sbg.ac.at

François-Xavier Vialard
LIGM, Université Gustave Eiffel, France
francois-xavier.vialard@u-pem.fr

Marc Niethammer
Department of Computer Science
UNC Chapel Hill, USA
mn@cs.unc.edu

Introduction

We cover these topics in the supplementary material that did not fit into the main manuscript:

- We retrace the proof of the H^1 regularizing property of approximate inverse consistency in greater detail.
- We specify all details of the neural network architectures used in the manuscript's experiments, including number of features, number of layers, training proceedure, etc.

1. Detailed proof of the regularizing effect of inverse consistency

This section details our derivation for the smoothness properties emerging from approximate inverse consistency.

Denote by $\Phi_\theta^{AB}(x)$ the output map of a network for images (I^A, I^B) and by $\Phi_\theta^{BA}(x)$ the output map between (I^B, I^A) Recall that we add two independent spatial white noises $n_1(x),\ n_2(x) \in \mathbb{R}^N$ $(x \in [0,1]^N$ with $N=2$ or $N=3$ the dimension of the image) of variance 1 for each spatial location to the two output maps and define $\Phi_{\theta\varepsilon}^{AB}(x) := \Phi_\theta^{AB}(x) + \varepsilon n_1(\Phi_\theta^{AB}(x))$ and $\Phi_{\theta\varepsilon}^{BA}(x) := \Phi_\theta^{BA}(x) + \varepsilon n_2(\Phi_\theta^{BA}(x))$ with ε a positive parameter. We consider the following loss

$$\mathcal{L} \;=\; \lambda\left(\|\Phi_{\theta\varepsilon}^{AB} \circ \Phi_{\theta\varepsilon}^{BA} - \mathrm{Id}\,\|_2^2 + \|\Phi_{\theta\varepsilon}^{BA} \circ \Phi_{\theta\varepsilon}^{AB} - \mathrm{Id}\,\|_2^2\right) \;+\; \|I^A \;\circ\; \Phi_\theta^{AB} \;-\; I^B\|_2^2 \;+\; \|I^B \;\circ\; \Phi_\theta^{BA} \;-\; I^A\|_2^2\,. \quad (1)$$

Throughout this section, we give the details of the expansion in ε of the loss, thus we use the standard notations o and O w.r.t $\varepsilon \to 0$. We focus on the first two terms (that we denote by $\lambda\mathcal{L}_{\text{inv}}$) since the regularizing property comes from the inverse consistency. We expand one of the first two terms of (1) since by symmetry the other is similar. If the noise is bounded (or with high probability in the case of Gaussian noise), we have

$$\|\Phi_{\theta\varepsilon}^{AB} \circ \Phi_{\theta\varepsilon}^{BA} - \mathrm{Id}\,\|_2^2 = \|\Phi_\theta^{AB} \circ \Phi_\theta^{BA} + \varepsilon n_1(\Phi_\theta^{AB} \circ \Phi_\theta^{BA}) + d\Phi_{\theta\varepsilon}^{AB}(\varepsilon n_2(\Phi_\theta^{BA})) - \mathrm{Id}\,\|_2^2 + o(\varepsilon^2)\,, \quad (2)$$

where $d\Phi$ denotes the Jacobian of Φ. By developing the squares and taking expectation, we get

$$\mathbb{E}[\|\Phi_{\theta\varepsilon}^{AB} \circ \Phi_{\theta\varepsilon}^{BA} - \mathrm{Id}\,\|_2^2] = \|\Phi_\theta^{AB} \circ \Phi_\theta^{BA} - \mathrm{Id}\,\|_2^2 + \varepsilon^2\mathbb{E}[\|n_1 \circ (\Phi_{\theta\varepsilon}^{AB} \circ \Phi_{\theta\varepsilon}^{BA})\|_2^2] + \varepsilon^2\mathbb{E}[\|d\Phi_{\theta\varepsilon}^{AB}(n_2) \circ \Phi_\theta^{BA}\|_2^2] + o(\varepsilon^2)\,, \quad (3)$$

since by independence all the cross-terms vanish. Indeed, the noise terms have 0 mean value. The second term is constant:

$$\begin{aligned}\mathbb{E}[\|n_1 \circ (\Phi_{\theta\varepsilon}^{AB} \circ \Phi_{\theta\varepsilon}^{BA})\|_2^2] &= \mathbb{E}[\int \|n_1\|_2^2(y)\,\mathrm{Jac}((\Phi_{\theta\varepsilon}^{BA})^{-1} \circ (\Phi_{\theta\varepsilon}^{AB})^{-1})dy] \\ &= \int \mathbb{E}[\|n_1\|_2^2(y)]\,\mathrm{Jac}((\Phi_{\theta\varepsilon}^{BA})^{-1} \circ (\Phi_{\theta\varepsilon}^{AB})^{-1})\,dy = const\,,\end{aligned}$$

where we performed a change of variables and denoted the determinant of the Jacobian matrix as Jac. The last equality follows from the fact that $\mathbb{E}[\|n_1\|_2^2(y)] = 1\,\forall y$, *i.e.* the variance of the noise is constant equal to 1. Last, we also use the change of variables $y = \Phi^{AB}_{\theta\varepsilon} \circ \Phi^{BA}_{\theta\varepsilon}(x)$. By similar computations, the last term in Equation (3) is equal to

$$\mathbb{E}[\|d\Phi^{AB}_{\theta\varepsilon}(n_2) \circ \Phi^{BA}_{\theta}\|_2^2] = \int \mathbb{E}[(n_2^\top d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon}(n_2)) \circ \Phi^{BA}_{\theta}]\, dx\,. \tag{4}$$

In the next formula, we use coordinate notations. For $i, k \in 1, \ldots, N$, we denote by $\partial_i \Psi^k$ the partial derivative w.r.t. the ith coordinate of the kth component of the map $\Psi : \mathbb{R}^N \to \mathbb{R}^N$, the notation $n^i(x)$ stands for the ith component of the noise, $i \in 1, \ldots, N$. Using these notations, we have

$$\begin{aligned}\mathbb{E}[(n_2(x))^\top d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon}(n_2(x))] &= \mathbb{E}[\sum_{k,i,j} n_2^i(x)\partial_i[\Phi^{AB}_{\theta\varepsilon}]^k(x)\partial_j[\Phi^{AB}_{\theta\varepsilon}(x)]^k n_2^j(x)] \\ &= \mathbb{E}[\sum_{k,i} \partial_i[\Phi^{AB}_{\theta\varepsilon}]^k(x)\partial_i[\Phi^{AB}_{\theta\varepsilon}(x)]^k]\,.\end{aligned} \tag{5}$$

In the previous equation, we used the property of the white noise n_2 which satisfies null correlation in space and dimension $\mathbb{E}[n_2^k(x)n_2^{k'}(x')] = 1$ if $(k, x) = (k', x')$ and 0 otherwise. Recognizing the trace in Formula (5), we finally get

$$\mathbb{E}[\|d\Phi^{AB}_{\theta\varepsilon}(n_2) \circ \Phi^{BA}_{\theta}\|_2^2] = \int \mathrm{Tr}([d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon}] \circ \Phi^{BA}_{\theta})dx = \int \mathrm{Tr}(d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon})\, \mathrm{Jac}((\Phi^{BA}_{\theta})^{-1})\, dy\,, \tag{6}$$

where Tr is the trace. The last equality follows from the change of variables with Φ^{BA}_{θ}.

Approximation and resulting H^1 regularization: Under the hypothesis that $\Phi^{AB}_{\theta} \circ \Phi^{BA}_{\theta}$, $\Phi^{BA}_{\theta} \circ \Phi^{AB}_{\theta}$ are close to identity, one has $\mathrm{Jac}((\Phi^{BA}_{\theta})^{-1}) = \mathrm{Jac}(\Phi^{AB}_{\theta}) + o(1)$. Therefore, the last term of (6) is approximated by

$$\int \mathrm{Tr}(d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon})\, \mathrm{Jac}((\Phi^{BA}_{\theta})^{-1})\, dy = \int \mathrm{Tr}(d(\Phi^{AB}_{\theta\varepsilon})^\top d\Phi^{AB}_{\theta\varepsilon})\, \mathrm{Jac}(\Phi^{AB}_{\theta})\, dy + o(1)\,. \tag{7}$$

Note that we only need an approximation at zeroth order, since this term appears at second order in the penalty $\mathcal{L}_{\text{inv}}$. Assuming this approximation holds, we use it in Eq. (6), together with the fact that, $\Phi^{AB}_{\theta\varepsilon} = \Phi^{AB}_{\theta} + O(\varepsilon)$ to approximate at order ε^2 the quantity $\mathcal{L}_{\text{inv}}$, *i.e.*,

$$\mathcal{L}_{\text{inv}} = \left\|\Phi^{AB}_{\theta} \circ \Phi^{BA}_{\theta} - \mathrm{Id}\right\|_2^2 + \left\|\Phi^{BA}_{\theta} \circ \Phi^{AB}_{\theta} - \mathrm{Id}\right\|_2^2 + \varepsilon^2 \left\|d\Phi^{AB}_{\theta}\sqrt{\mathrm{Jac}(\Phi^{AB}_{\theta})}\right\|_2^2 + \varepsilon^2 \left\|d\Phi^{BA}_{\theta}\sqrt{\mathrm{Jac}(\Phi^{BA}_{\theta})}\right\|_2^2 + o(\varepsilon^2)\,. \tag{8}$$

Last, the square root that appears in the L^2 norm is simply a rewriting of the term on the r.h.s. of Eq. (7).

We see that approximate inverse consistency leads to an L^2 penalty of the gradient, weighted by the Jacobian of the map. Generally, this is a type of Sobolev (H^1 more precisely) regularization sometimes used in image registration in a different context, see [3] for a particular instance. In particular, the H^1 term is likely to control the compression and expansion magnitude of the maps, at least on average, on the domain. Hence, approximate inverse consistency leads to an implicit H^1 regularization, formulated directly on the map.

In comparison with the literature, the regularization is not formulated on the velocity field defining the displacement by integration as it is standard in diffeomorphic registration of pairwise images since the pioneering work of [2]. In our context, the resulting H^1 penalty concerns the map itself. On the theoretical side, one can ask if such regularization makes the problem well posed from the analytical point of view, *i.e.* existence of minimizers, regularity of solutions. However, few works have explored this type of regularization directly on the map, see for instance the work in [1] in the context of optimal transport. In contrast, H^1 regularization of the velocity field has been explored resulting in a non-degenerate metric on the group of diffeomorphisms as proven in [4].

Further discussion: When the noise level ε is turned to 0, we also observe a regularizing effect when the map is output by a neural network. (Although not when the map is directly optimized.) Since the network does not perfectly satisfy the inverse consistency soft constraint, we conjecture that the resulting error behaves like the white-noise we studied above, thereby explaining the observed regularization.

Another important challenge is to understand the regularization bias which comes from the population effect. In this case, we conjecture that this approach makes learning the regularization metric more adaptive to the given population data.

However, a fully rigorous theoretical understanding of the regularization effect due to the data population and its link with inverse consistency when no noise is used is important, but beyond our scope here.

2. Network Architectures

In this manuscript we refer to four neural network architectures: MLP, Encoder-Decoder, U-Net, and Convolutions. The details of each are provided next.

2.1. MLP

MLP refers to a multilayer perceptron with no special structure. The input, a pair of images, is flattened into a vector. Next, it is passed through hidden layers of size 8000 and 3000, each followed by a ReLU nonlinearity. The output layer is of dimension 2 · Image Width · Image Height, which is reshaped into a grid of displacement vectors from which Φ is calculated.

2.2. Encoder-Decoder

The Encoder-Decoder used in this manuscript is composed of a convolutional encoder and decoder, resembling a U-Net with no skip connections. Each layer consists of a stride 2 convolution or transpose convolution with kernel size 3x3 in the encoder and 4x4 in the decoder, followed by a ReLU nonlinearity. The layers have 16, 32, 64, 256, and 512 features in the encoder, and 256, 128, 64, 32, and 16 features in the decoder. As in all cases, the output is a grid of displacement vectors.

2.3. ConvOnly

This refers to an architecture consisting of six convolutional layers, each with 5x5 kernel, ReLU nonlinearity, and 10 output features. No downsampling or upsampling is performed. Each layer is fed as input the concatenation of the outputs of all previous layers.

2.4. U-Net

This is a U-Net with skip and residual connections. The convolutional layers have the same shapes and output dimensions as the encoder decoder network, but use Leaky ReLU activation placed before convolution instead of after. In addition, batch normalization is inserted before each convolution, and a residual connection is routed around each convolution, using upsampling or downsampling as required to match the image size.

For all four of these architectures the weights of the final layer of the neural network are initialized to zero instead of randomly, such that training begins with the network outputting a displacement field of zero. The code specifying these architectures is included in the file `networks.py`

3. Software Architecture

In the codebase that we developed for this paper, registration algorithms are implemented as subclasses of `pytorch.nn.Module`. A registration algorithm's `forward` method takes as input a batch of pairs of images to register I^A and I^B, and returns a python function $\Phi[I^A, I^B] : \mathbb{R}^N \to \mathbb{R}^N$ that maps a batch of vectors from the space of image B to the space of image A. For example, to use a neural network that outputs a displacement field as a registration algorithm, we wrap it in the class FunctionFromVectorField:

```
class FunctionFromVectorField(nn.Module):
    def __init__(self, net):
        super(FunctionFromVectorField, self).__init__()
        self.net = net

    def forward(self, image_A, image_B):
        vectorfield_phi = self.net(image_A, image_B)

        def ret(input_):
            return input_ + compute_warped_image_multiNC(
                vectorfield_phi, input_, self.spacing, 1
            )
        return ret
```

where `compute_warped_image_multiNC` interpolates between the vectors of its first argument at tensor of positions specified by its second argument. This code corresponds to equation (11) in the manuscript. Note especially that in the returned function `ret`, we add the input to the interpolated displacement. We do not attempt to interpolate a grid representation of a map (ie, a voxelized displacement field added to a voxelized identity map), as a displacement field can be extrapolated naturally, but a map cannot.

We find this organizational convention to be highly composable: this approach makes it simple to construct a registration algorithm that expands on the behavior of a component registration algorithm. For example, to operate on a high resolution pair of images with a low resolution registration algorithm, we use a wrapper with the following simple `forward` method:

```
def forward(self, image_A, image_B):
    x = self.avg_pool(x, 2, ceil_mode=True)
    y = self.avg_pool(y, 2, ceil_mode=True)
    return self.wrapped_algorithm(x, y)
```

Since the output is a fully fledged function : $\mathbb{R}^N \to \mathbb{R}^N$ which is resolution agnostic, it does not need to be modified by this method and can simply be passed along.

4. Composition of transforms

We have defined a registration algorithm as a functional Φ which takes as input two functions $\mathbb{R}^N \to \mathbb{R}$, I^A and I^B, and outputs a map $\mathbb{R}^N \to \mathbb{R}^N$ that aligns them, specifically satisfying $I^A \circ \Phi[I^A, I^B] \simeq I^B$. Most registration algorithms have multiple steps, such as an affine step followed by a deformable step, and so it is useful to define how to compose two algorithms (i.e., two procedures for computing a map from a pair of images) Φ and Ψ. The most obvious approach to this problem is to apply Φ to the problem $\{I^A, I^B\}$, yielding a function Φ^{AB} such that $I^A \circ \Phi^{AB} \simeq I^B$. Then, the intermediate image $\tilde{I}^A := I^A \circ \Phi^{AB}$ is computed using the function Φ^{AB} that was found, and the second registration problem is declared to be registering $\{\tilde{I}^A, I^B\}$ by computing a map using the algorithm Ψ. Putting this together, we set out to define an operator TwoStep satisfying the equation

$$I^A \circ \mathit{TwoStep}\{\Phi, \Psi\}[I^A, I^B] = (\tilde{I}^A) \circ \Psi[\tilde{I}^A, I^B] \simeq I^B\,, \tag{9}$$

$$I^A \circ \mathit{TwoStep}\{\Phi, \Psi\}[I^A, I^B] = (I^A \circ \Phi[I^A, I^B]) \circ \Psi[I^A \circ \Phi[I^A, I^B], I^B] \simeq I^B\,. \tag{10}$$

Since composition is associative, we can move the parentheses and isolate TwoStep as

$$\mathit{TwoStep}\{\Phi, \Psi\}[I^A, I^B] = \Phi[I^A, I^B] \circ \Psi[I^A \circ \Phi[I^A, I^B], I^B]\,. \tag{11}$$

This is implemented in our code base as another registration algorithm with the following forward method

```
def forward(self, image_A, image_B):
    phi = self.netPhi(image_A, image_B)
    phi_vectorfield = phi(self.identityMap)
    self.image_A_comp_phi = compute_warped_image_multiNC(
        image_A, phi_vectorfield, self.spacing, 1
    )
    psi = self.netPsi(self.image_A_comp_phi, image_B)
    return lambda input_: phi(psi(input_))
```

5. Training Procedure for synthetic data and MNIST

5.1. Regularization by approximate inverse consistency

To investigate the regularizing effects of approximate inverse consistency, we register an image of a circle and a triangle by three methods: By directly optimizing the forward and reverse displacement vector fields, by directly optimizing the displacement vector fields with added noise, and by optimizing a U-Net that outputs a displacement vector field over a dataset that includes the image of a circle and triangle. This dataset was generated as follows: a center (cx, cy) for each image is sampled uniformly from $[0.4, 0.7] \times [0.4, 0.7]$, a radius r is sampled from $[0.2, 0.4]$, and an angle θ from $[0, 2\pi]$. Each point in the image is then associated with an intensity by one of the following formulas: Half of the generated images are chosen to be circles and their intensities are set via the expression

$$\tanh\left(-40 \cdot (\sqrt{(x-cx)^2+(y-cy)^2}-r)\right) , \tag{12}$$

while the remainder are chosen to be triangles and their intensities are set to

$$\tanh\left(-40 \cdot (\sqrt{(x-cx)^2+(y-cy)^2}-r \cdot \frac{\cos(\frac{\pi}{3})}{\cos((\arctan 2(x-cx, y-cy)+\theta)\%(\frac{2\pi}{3})-\frac{\pi}{3})}\right) . \tag{13}$$

In each case, the ADAM optimization algorithm is chosen, with a learning rate of 0.0001. While training the U-Net, a batch size of 128 is used. The code to train the U-Net is included as `training_scripts/triangle_example.py`, and the code to directly optimize the deformation fields is included as `notebooks/NoNetwork.ipynb`

5.2. Regularization for different networks

For all architectures and choices of λ, the following training procedure was followed. The network is trained in pytorch using the Adam algorithm, a batch size of 128, and a learning rate of 0.0001, for 18750 steps (400 epochs). The self contained notebook to generate/download the datasets, train each combination and generate Figure 6 is included in the supplementary files as `notebooks/InverseConsistencyGenerateFigure6.ipynb`. This code can be downloaded and run on its own, as it only depends on pytorch and matplotlib.

6. Training Procedure for our OAI knee results

6.1. Automatic increase of λ during training

During our initial experiments with training our registration network on real data, we found that, in the event that an initial value of λ was selected that was too low, leading to an unacceptable degree of folding, we were able to increase λ and continue training the network, suppressing the folds without significantly reducing the achieved DICE score. However, when we repeated the training with λ beginning at this high value, training proceeded much more slowly due to the ill-conditioned nature of solving a constrained optimization problem using a large quadratic penalty. This was never an issue when registering 2-D datasets, because it was feasible to train on them for a sufficient number of iterations for Adam optimization to automatically resolve the ill conditioning using a small step size. However, on the 3-D dataset this issue threatened to make training times impractical. Our initial solution to this problem was to begin training with λ small, and manually increase λ during training whenever the number of folds became large. While this worked well, it introduced a large number of hyperparameters in the form of a complex training schedule, defeating the purpose of our approach. Instead, we decided to select as a hyperparameter the acceptable number of folds, and increase λ at each iteration of training if the number of folds measured that iteration exceeded the decided-upon acceptable number of folds.

6.2. Details

The "acceptable number of folds" hyperparameter was set to 200. 200 was the first value tried for this hyperparameter, however this choice was informed by the outcome of previous experiments where λ was set manually.

First, the 'low resolution network' is composed of two U-Nets that each take as input a pair of knee images at half resolution, and output a displacement map. These are combined using the operator TwoStep as described above. The low resolution network is trained end to end with λ incremented whenever the batch-mean number of folds exceeds 200, as described above. The batch size used is 128, the learning rate is set to 0.00005, and the network is trained for 16,000 steps. This low resolution pretraining serves to greatly reduce the overall time needed to train the neural network, since much larger batches of images can fit into GPU memory. This step is performed by the included script `training_scripts/double_deformable_knee.py`, and the resulting loss curve is reproduced here in Figure 2.

Second, the 'low resolution network' is wrapped with a class that downsamples input images, and then combined with a U-Net that takes as input full resolution images, again using the operator TwoStep. The weights of the low resolution network are frozen, and the full resolution network is trained for 75,000 steps, with a learning rate of 0.00005 and a batch size of 16. This step is performed by the included script `training_scripts/hires_finetune_frozen_lowres.py`. The loss curve associated with this step is presented in Figure 2. Finally, evaluation of the low resolution and full networks is performed using the included notebooks. `DoubleDeformableDICE.ipynb` and `DoubleDeformable-HiresDICE.ipynb` respectively. Training was done on a machine with 4 RTX 3090 GPUs, taking 2 days for the low resolution component and 4 days for the high resolution component.

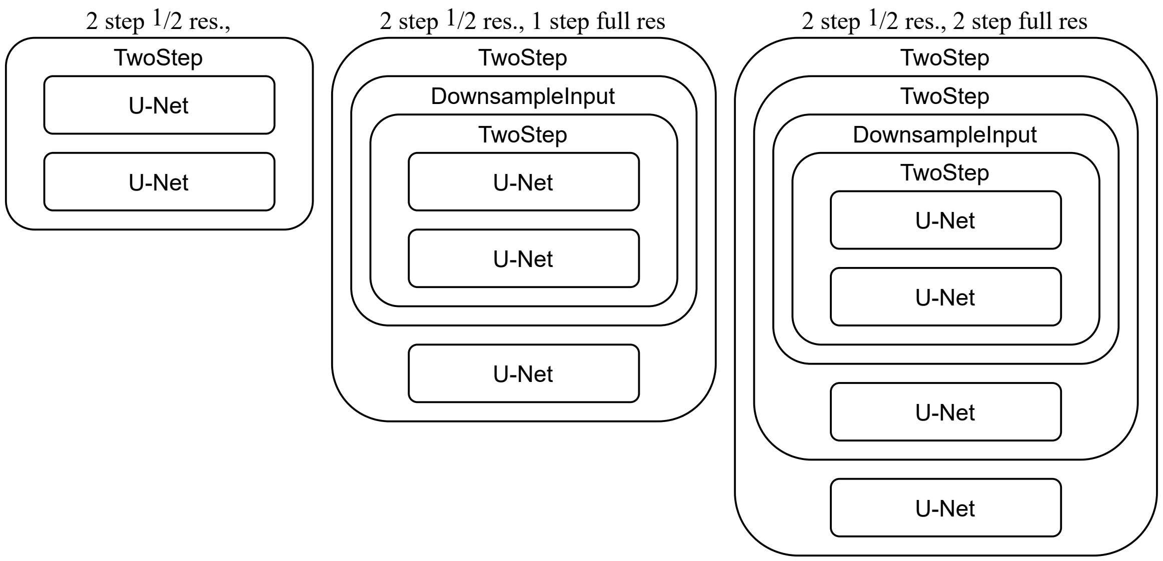

Figure 1. Architectures used for our OAI results. The low resolution component inside DownsampleInput only requires an 8th the memory and computing power of the whole network, and pretraining it alone makes our overall approach computationally feasible on a single 4 GPU machine. Once it is pretrained, it is plugged into the larger model as shown.

References

[1] Luigi De Pascale, Jean Louet, and Filippo Santambrogio. The monge problem with vanishing gradient penalization: Vortices and asymptotic profile. *Journal de Mathématiques Pures et Appliquées*, 106(2):237–279, 2016. ii

[2] P. Dupuis, U. Grenander, and M. I. Miller. Variational problems on flows of diffeomorphisms for image matching. *Quart. Appl. Math.*, 56:587–600, 1998. ii

[3] Anil N. Hirani, Jerrold E. Marsden, and James Arvo. Averaged template matching equations. In Mário Figueiredo, Josiane Zerubia, and Anil K. Jain, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 528–543, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ii

[4] Peter W. Michor and David Mumford. Vanishing geodesic distance on spaces of submanifolds and diffeomorphisms. *Doc. Math.*, 10:217–245, 2005. ii

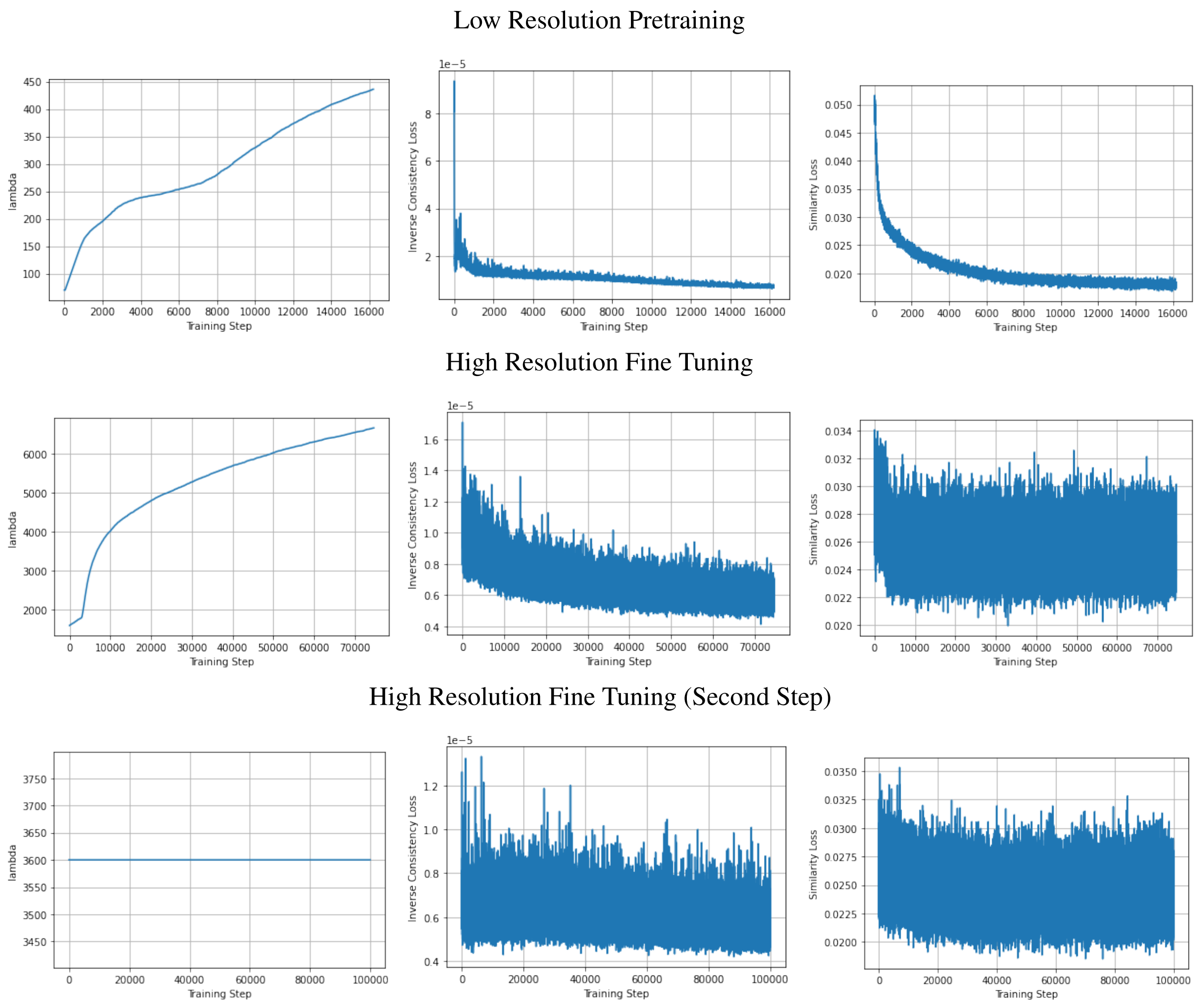

Figure 2. Training curves for our result on OAI dataset. It is interesting that the required value of λ to suppress folding increases over the course of training, and in particular increases rapidly once we begin training in high resolution. Nonetheless, our approach of incrementing λ by a fixed amount whenever the number of folds in a batch exceeds a threshold successfully generates smooth transforms.