Contents

1 Abstract

Deformable image registration is important for cancer treatments that utilize an MR-Linac (a linear accelerator with integrated magnetic resonance imaging). Traditional methods of two- and three-dimensional image registration require equipment-specific, handcrafted approaches and lack the necessary image alignment speed for these applications. We introduce a non-black-box deep learning convolutional variational network for two- and three-dimensional deformable medical image registration, called DIRV-Net, which can learn by utilizing either supervised or unsupervised approaches. The model uses variational networks, fields of experts, and image pyramids for iterative prediction of displacement fields. Our model demonstrates its ability to register two- and three-dimensional images deformed with complex, smooth deformations to near maximum theoretical accuracy. It also shows generalizability and robustness against various image structures and features, including previously unseen ones.

2 Introduction

2.1 Background

About half of all cancer patient receive some form of radiation therapy as part of their treatment [1], what makes it a main component in oncological treatment protocols. It utilizes high-energy ionizing radiation to target and destroy malignant tissues. It often relies on a linear accelerator (Linac) to direct a focused beam of radiation to the tumor. The therapeutic regime typically involves multiple treatment sessions (called fractions) and aims to deliver cumulative doses of radiation to the tumor while minimizing exposure to healthy tissues.

One of the inherent challenges in radiation therapy is the management of spatial uncertainties due to the dynamic nature of human anatomy. Variations in metabolic processes, bowel movements, and breathing, along with changes in patient weight, positioning, and other physiological factors, can lead to significant uncertainties in the target location. Traditionally, to compensate for these uncertainties, oncologists have applied relatively wide safety margins around the tumor. Although this ensures that the prescribed dose fully encompasses the tumor, it does not sufficiently reduce the volume of healthy tissues exposed to harmful radiation, thereby escalating the risk of side effects and long-term complications.

The integration of Linac with online magnetic resonance (MR) imaging has been a pivotal advancement in adaptive radiation therapy (i.e., where treatment plans can be updated daily based on the patient's current anatomy) due to its superior capability for imaging soft tissues. By providing high-quality, online images of the patient's anatomy non-invasively, MR enables the adaptation of treatment plans to the patient's current anatomical status, thereby allowing for more precise targeting of radiation doses. This approach can enable the reduction of extensive safety margins, thereby avoiding unnecessary radiation exposure to healthy tissues.

However, using MR imaging introduces new challenges, particularly when handling big datasets of volumetric images that must be precisely and relatively swiftly aligned across treatment fractions. Image registration involves computing a voxel-wise, non-linear mapping between a series of image volumes to achieve accurate spatial alignment of those images with each other [2] despite anatomical deformations and patient movement.

3D deformable image registration is an active area of research in medical image analysis. It relies heavily on the mathematical foundations of reversing deformations and optimization techniques. It is considered an ill-posed problem due to the infinite number of potential solutions; thus, regularization strategies are crucial to constrain the solution space, ensuring physically meaningful deformations.

2.2 Related Work

Various methods to address the image registration problem have been proposed. One of the early approaches was based on optical flow methods [3]. Other methods included tissue elasticity estimation [4] and parametric total-variation (pTV) [5]. These methods mostly relied on iterative optimization techniques [5] and required hand-crafted complex regularization terms to bound the target energy for optimization, which could become uncontrolled due to the ill-posed nature of the image registration problem. Additionally, these methods usually demanded high computational power, making them unsuitable for semi-real-time ($\sim$ 5Hz) image registration.

More advanced methods, such as the introduction of variational networks by [6], depend on unrolling the iterative scheme of an energy optimization process and bridging variational methods with neural networks. These were important milestones in the field. This approach has been used successfully for image reconstruction by the Goksel group [7]. Later, [8] demonstrated that this approach is extendable to image registration problems by introducing multiple neural network components, each specializing in different tasks. Almost simultaneously, a master's student in this group applied this approach to deep learning-based unrolling of iterative image registration problems in 2D [9].

Modern approaches in this domain leverage various deep learning techniques. These approaches mostly focus on predicting the deformation fields directly from the image data [10]. They vary between unsupervised, weakly supervised, and supervised methods. In general, deep learning has demonstrated the ability to learn complex, non-linear mappings tailored to the specific distribution of medical image data, eliminating the need for hand-crafted parameterization.

2.3 Research Question

In this work, we introduce the design of a non-black-box CNN-based variational network unrolling, which we call DIRV-Net, and which can learn in both supervised and unsupervised regimes. This is a reconsideration, correction, and extension of [9]'s work to 3D images. We examine the network behavior and compare its performance for two- and three-dimensional medical image registration under various conditions after incorporating various improvements.

In short, the following was introduced:

1. Mathematical reformulation and motivation of DIRV-Net.
2. Full code rewrite for extensibility and decoupling, along with bug fixes and optimizations.
3. Performance and quality improvement through the Swapping trick, Random Patching, and Recurrent Refinement.
4. Generalization to 3D images, implementing related new functionalities, and improving the loss function.
5. Introduction of artifact-free non-rigid transformations, configurable affine deformations, and synthetic dataset generation.

2.4 Notations

To avoid notation abuse, we clarify our use of notations briefly here.

Scalars and scalar functions will use normal thickness symbols, while fields, sets, arrays, or array-valued operators or functions will have bold symbols.

Functions evaluation, in general, is symbolized by surrounding the evaluated parameter with rounded brackets. However, the evaluation of array/vector-valued functions and operators at a single point is symbolized by surrounding the evaluated object with square brackets.

Subscripts point to the vector component index. Superscripts within rounded brackets always indicate some running index.

Any estimated function is distinguished from the ground truth object by a hat over the symbol.

The symbol " := " means that the left-hand symbol is defined by the right-hand expression; the opposite situation applies to the mirrored symbol.

We will define three operators/functions throughout the text, the naming of which could be slightly confusing due to their naming similarity, although they are different: up/down sampling, up/down scaling, and sampling.

Some abbreviations that we will use frequently:

- "FF" stands for a flow field, used to describe deformations.
- "DF" stands for displacement field, a coarser version of the FF.
- "VU" stands for a variational unit.
- "PL" stands for a pyramid level (can be a pyramid level of an image, DF or FF).

3 Image Registration Problem

To describe the registration problem rigorously, we need to introduce some definitions and mathematical notations in what follows.

3.1 Basic Definitions

An *image* of $r \in \mathbb{N}/\{0\}$ dimensions (2 or 3 in this work) and $n_c \in \mathbb{N}/\{0\}$ channels can be described by a discrete map:

$$\boldsymbol{I} : \boldsymbol{\Omega} \to \mathbb{R}^{n_c} \tag{1}$$

where $\boldsymbol{\Omega} \coloneqq \Omega_1 \times ... \times \Omega_r \ : \ \Omega_i \coloneqq [0, ... n_i - 1] \subset \mathbb{N}, i = 1 ... r$ is the set of discrete pixel coordinates $\mathbf{p} \coloneqq (p_1, p_2, ..., p_r) \in \boldsymbol{\Omega}$, and n_i is the image size (in pixels) along the corresponding i'th dimension. Hence $\|\Omega_i\| =: n_i \in \mathbb{N}/\{0\}$.

We call the number of image dimensions (excluding channels dimension) r the *core rank*. Each pixel $\mathbf{p}$ is mapped to the corresponding intensity values in each of the n_c channels as $\boldsymbol{I}[\mathbf{p}] \subset \mathbb{R}^{n_c}$. An alternative way to represent an image is as an array/operator $\boldsymbol{I} \coloneqq I[\boldsymbol{\Omega}] \subset \mathbb{R}^{r \cdot n_c}$.

We also define the image core size vector as $\mathbf{n}_I \coloneqq (n_1, ..., n_r) \subset \mathbb{N}^r/\{\mathbf{0}\}$, and hence $(\boldsymbol{n}_I)_i \coloneqq n_i$. We define the multiplication of all vector components with the symbol $|\cdot|$; thus, the total number of pixels in the image is $|\boldsymbol{n}_I| = n_1 \cdot n_2 ... \cdot n_r$.

In this work, for reasons that will be explained later on, we will consider single-channel images, $n_c \coloneqq 1$.

3.2 The Transformation Map

This work is primarily about deformations, so we want a suitable way to define them.

Assume that we have two continuous vector-valued functions $\boldsymbol{f}_f, \boldsymbol{f}_m$, both defined on the same domain $\mathbb{R}^r$, and that $\boldsymbol{f}_m$ is the result of a continuous and smooth deformation of the $\boldsymbol{f}_f$ function.

To describe such a deformation, we can define a *transformation map* as $T : \mathbb{R}^r \to \mathbb{R}^r$, which moves any point $\mathbf{x} \in \mathbb{R}^r$ from the function's domain to another point in the same domain $\bar{\mathbf{x}} \in \mathbb{R}^r$, such that the values of both $\boldsymbol{f}_f, \boldsymbol{f}_m$ functions will be identical at those points. We can describe this mathematically as:

$$\bar{\mathbf{x}} \coloneqq T(\mathbf{x}) \quad : \quad \boldsymbol{f}_m[\mathbf{x}] = \boldsymbol{f}_f[\bar{\mathbf{x}}] \tag{2}$$

Since this map is continuous, we can also consider the inverse map as:

$$\tilde{\mathbf{x}} \coloneqq T^{-1}(\mathbf{x}) \quad : \quad \boldsymbol{f}_m[\tilde{\mathbf{x}}] = \boldsymbol{f}_f[\mathbf{x}] \tag{3}$$

One may naively apply a similar approach to define deformations on images; however, there are multiple reasons why this approach cannot be applied directly. We will discuss them in Sec. 3.5.2, and until then, we will assume that those issues are resolved in what follows, and that the functions $\boldsymbol{f}_f, \boldsymbol{f}_m$ can be considered as images.

3.3 Problem Statement

Let $\boldsymbol{I}_f : \boldsymbol{\Omega} \to \mathbb{R}^{n_c}$ be an image that serves as a reference or template image for the registration procedure, referred to as the *fixed image*. Similarly, let $\boldsymbol{I}_m : \boldsymbol{\Omega} \to \mathbb{R}^{n_c}$ be the resultant image from $\boldsymbol{I}_f$ after it is deformed. This image is usually called the *moving image*. Here we assume that both images are defined on the same domain and codomain, i.e., they have the same core rank and dimension sizes. We also assume that they have the same channels; otherwise, if we have more than just color channels, this would be called *multimodal image registration*.

The registration of the moving image $\boldsymbol{I}_m$ involves reconstructing this deformed image back to its original state, i.e., to the fixed image $\boldsymbol{I}_f$. This can also be phrased in terms of the transformation map we defined above. Namely, the registration problem seeks to find the map T^{-1} given the images $\boldsymbol{I}_f, \boldsymbol{I}_m$. Such problems are usually classified as *inverse problems*.

In the case of supervised learning, as in this work, we assume that the ground truth inverse map T^{-1} is also known, and we try to train our network to estimate the inverse map $\hat{T}^{-1} \simeq T^{-1}$ to be as close as possible to the ground truth T^{-1}.

By applying this estimated inverse transformation map to the moving image, we generate the network estimate of the fixed image, usually called the *registered image* $\boldsymbol{I}_R$, i.e.:

$$\boldsymbol{I}_f \backsimeq \hat{\boldsymbol{I}}_f \coloneqq \boldsymbol{I}_R \tag{4}$$

3.4 The Flow Field

The registration problem we described focuses more on the repositioning of pixels (between the moving and fixed images) than on the overall form of the transformation T.

To reflect this fact, we introduce the discrete *flow field* (FF). This field defines, at each pixel of the moving image, an r-component vector that describes how much this pixel will move in each of the image's r dimensions (relative to its original position) by the transformation T^{-1}.

Mathematically, the FF can be defined as the map $\boldsymbol{\mathcal{F}} : \boldsymbol{\Omega} \to \mathbb{R}^r$ (note that using the codomain $\mathbb{R}^r$ means that a pixel can go out of the original image boundaries due to the deformation; more on this in Sec. 3.5.2) such that:

$$T^{-1}(\mathbf{p}) =: \mathbf{p} + \boldsymbol{\mathcal{F}}[\mathbf{p}] \tag{5}$$

Note that we can also treat this field as an array/operator, or even as an image of core size equal to the core size of $\boldsymbol{I}$, i.e., to $|\boldsymbol{n}_I|$, but of r channels (since at each pixel, the vector of the FF will consist of r components), i.e., $\boldsymbol{\mathcal{F}} \subset \mathbb{R}^{|\boldsymbol{n}_I| \cdot r}$.

By this, we now aim to estimate this FF instead of the inverse transformation map itself.

3.5 Discretization Considerations

Up to this point, we have not considered the discrete domain for digital processing when we defined the FFs, and we did not address how the discrete transformation map can be defined. In what follows, this is addressed.

3.5.1 The Displacement Field

The defined FF that we need to estimate will contain $\|\boldsymbol{n}_I\| \cdot r$ elements, which is a relatively large number of parameters to estimate by itself. However, usually, we assume that the FF will satisfy the following:

1. The FF is a relatively smooth vector field.
2. In relation to the previous item, we assume that nearby pixels will have similar movement directions and magnitudes.
3. The transformation magnitude is larger than the pixel's physical size.
4. The transformation magnitude is bounded (usually the FF is of the order $\|\boldsymbol{\mathcal{F}}\| \sim \sqrt[r]{|\boldsymbol{n}_I|}/10$).

These assumptions provide us with a way to reduce the number of parameters needed to model the FF: we can approximate it by a *displacement field* (DF) discretization[1], which is a field defined on a sparser grid than the grid of image pixels. The points of the sparse grid defined by DFs are called *control points*, and they approximate the FF vectors in their own neighborhood.

Let the separations (in pixel count) between the control points $\mathbf{k}_m$ be defined as $\boldsymbol{s}_k \coloneqq (s_1, \ldots s_r) : s_i \in \mathbb{N}^r/\{\mathbf{0}\}$, where s_i is the separation in the dimension i. Hence, the number of control points in any dimension is smaller or equal to the size of that dimension in pixels: $s_i \leq n_i$. The number of control points in each dimension, or the size of the DF grid, can be defined as $\boldsymbol{n}_k \coloneqq \boldsymbol{n}_I \oslash \boldsymbol{s}_k : \in \mathbb{N}^r/\{\mathbf{0}\}$, where $\oslash$ is an element-wise division (we assume here that these quantities are integer divisible), and then we can also write that $(\boldsymbol{n}_k)_i \leq (\boldsymbol{n}_I)_i$.

By this, we can define the DF as a map: $\boldsymbol{\mathcal{D}} : \bar{\boldsymbol{\Omega}} \rightarrow \mathbb{R}^r$ where $\bar{\boldsymbol{\Omega}} \coloneqq \bar{\boldsymbol{\Omega}}_1 \times \ldots \times \bar{\boldsymbol{\Omega}}_r : \bar{\boldsymbol{\Omega}}_i \coloneqq [1, \ldots, (\boldsymbol{n}_k)_i] \subset \mathbb{N}/\{0\}$, and such that, for an arbitrary control point $\mathbf{k} \in \bar{\boldsymbol{\Omega}}$, we have:

$$\boldsymbol{\mathcal{D}}[\mathbf{k}] \coloneqq \boldsymbol{\mathcal{F}}[\mathbf{k}] \tag{6}$$

Moreover, for any pixel $\mathbf{h_k}$ in the relatively close neighborhood of the control point $\mathbf{k}$, we have $\boldsymbol{\mathcal{F}}[\mathbf{h_k}] \backsimeq \boldsymbol{\mathcal{D}}[\mathbf{k}]$. The array representation of this DFs can then be written as $\boldsymbol{\mathcal{D}} \in \mathbb{R}^{|\boldsymbol{n}_k| \cdot r}$.

For the following, we would like to have a way to go from FF to DF, and vice versa. For the first case, we define a *sampling operator*, $\boldsymbol{\mathcal{S}}_{\boldsymbol{s}_k} : \mathbb{R}^{|\boldsymbol{n}_I| \cdot r} \rightarrow \mathbb{R}^{|\boldsymbol{n}_k| \cdot r}$, which will simply take samples from the FF at each control point, i.e., at each s_i'th pixel, hence we can write:

$$\boldsymbol{\mathcal{S}}_{\boldsymbol{s}_k}(\boldsymbol{\mathcal{F}}) \coloneqq \boldsymbol{\mathcal{D}} \tag{7}$$

To go from the coarser DF to the finer FF (i.e., denser grid size), we need to upsample the DF by using some kind of interpolation. Let $\boldsymbol{\mathcal{Z}}_{\boldsymbol{n}} : \mathbb{R}^{|\boldsymbol{n}_k| \cdot r} \rightarrow \mathbb{R}^{|\boldsymbol{n}_I| \cdot r}$ be a *up/down sampling operator* that interpolates the input to a new vector field of core size $\boldsymbol{n}_I$ (more on how this interpolation will be exactly performed is discussed in Sec. 5.1). Then we write:

$$\boldsymbol{\mathcal{Z}}_{\boldsymbol{n}_I}(\boldsymbol{\mathcal{D}}) \coloneqq \hat{\boldsymbol{\mathcal{F}}} :\backsimeq \boldsymbol{\mathcal{F}} \tag{8}$$

[1] In the literature, "flow" and "displacement" field definitions are used interchangeably. However, here we dedicate the "flow field" name to the dense field defined on every pixel of the image, while the "displacement field" describes the field defined on the coarsest grid, i.e., on the control points.

where the used approximation symbol is due to information loss that occurs because of the interpolation. Hence, the estimated inverse transformations can be recovered by:

$$\begin{aligned} \hat{T}^{-1}(\mathbf{p}) &\backsimeq \mathbf{p} + \hat{\boldsymbol{\mathcal{F}}}[\mathbf{p}] \\ &\backsimeq \mathbf{p} + \boldsymbol{\mathcal{Z}}_{\boldsymbol{n}_I}(\hat{\boldsymbol{\mathcal{D}}})[\mathbf{p}] \end{aligned} \tag{9}$$

However, as will be shown later on, due to the convolutional nature of the proposed model, the choice of how dense the DF used (i.e., how close control points are) is merely bounded by the available computational resources rather than being a model architecture restriction.

3.5.2 Image Warping

Now we discuss how we can use the transformation map defined in eq. (2) on the discrete image domain.

A direct way will be to re-write that transformation map on a pixel $\mathbf{p}$ basis as $\boldsymbol{I}_m[\mathbf{p}] \sim \boldsymbol{I}_f[T(\mathbf{p})]$ (the reason for $\sim$ will become apparent in what follows). Such a procedure is usually called *forward warping*.

However, such a straightforward generalization to the discrete image domain leads to the following issues:

1. The map assumes that the pixels will be mapped to others of exactly the same intensity. In other words, this definition is structure agnostic, which is not suitable for the main goal of real image registration applications, where we aim to map features to features, and not just pixel intensities.
2. We assumed initially that the map T is continuous, but in the case of image deformations, multiple unique pixels from $\boldsymbol{I}_f$ may get merged and mapped to a single pixel in $\boldsymbol{I}_m$.
3. Conversely, this may also create holes (see Fig. 1); i.e., some of the pixels in $\boldsymbol{I}_m$ may not get an assigned intensity value because the corresponding pixel from $\boldsymbol{I}_f$ got merged with some other pixels of $\boldsymbol{I}_f$ and placed somewhere else in $\boldsymbol{I}_m$.
4. We assumed that the transformation map is defined on a codomain of infinitely large dimensions. However, this is not allowed in the case of image deformations, since some pixels may be pushed out of the moving image boundaries in this way.
5. We assumed that the output of the transformation map could be any real number. However, in images, we can deal only with integer-valued pixel coordinates, and such an output may point to an "in-between" pixel boundary location instead.

From this, we see that for general image deformations, T is neither an injective nor a surjective map, and clearly, those issues may lead to information loss and image distortion. Moreover, this means that while we used to treat the continuous map and its inverse as kind of "equivalent" objects, this cannot be done anymore with general maps that deal with image deformations.

The first issue can be resolved by dropping the requirement that this map should maintain the same pixel intensity in images. However, doing this, along with the fact that the mapping is now neither injective nor surjective, will blow up the space of possible solutions for the registration problem and

make it virtually of infinite size. This is the main reason why the image registration problem is an ill-defined one, mathematically speaking.

Regarding the rest of the issues, heuristically speaking, the second one can be simply resolved by simple averaging. The fifth one can also be avoided by performing some kind of interpolation that will calculate the most likely neighboring pixel intensity values. However, issues 3 and 4 have no obvious resolution.

Let us now consider the other way: applying the inverse map. Namely, let $\boldsymbol{I}_f[\mathbf{p}] \sim \boldsymbol{I}_m[T^{-1}(\mathbf{p})]$. This procedure is called *backward warping*.

Now we can easily see that backward warping will not face the above second and third issues. The reason is that the inverse transformation map guarantees that its output will be within fixed image boundaries since that was the domain of the ordinary map T from the start. Moreover, for the same reason, it will guarantee that this map will cover all of the image pixels, i.e., no holes will occur.

Hence, by using backward warping, we will automatically resolve the more problematic second and third issues. This also explains why we defined the transformation map from fixed images to the moving one in eq. (2), because this makes us interested in estimating the well-behaving inverse map, not the direct one.

Finally, to address the mentioned solutions for the second and fifth issues mathematically, we can define a *warping operator* as $\boldsymbol{\mathcal{W}} : \mathbb{R}^{|\boldsymbol{n}_I|} \times \mathbb{R}^{|\boldsymbol{n}_I| \cdot r} \rightarrow \mathbb{R}^{|\boldsymbol{n}_I|}$, which takes two variables: the whole moving image $\boldsymbol{I}_m$ (since due to the need to average and interpolate pixel intensities, we can't define it at a single pixel level), and the inverse map of the whole pixel coordinates set $\boldsymbol{\Omega}$.

Since we are now focusing on the FF, we can also use eq. (9) to write the warping function as:

$$\boldsymbol{I}_R = \hat{\boldsymbol{I}}_f \coloneqq \boldsymbol{\mathcal{W}} \left[\boldsymbol{I}_m, \hat{\boldsymbol{\mathcal{F}}}(\boldsymbol{\Omega}) \right] \tag{10}$$

Regarding the interpolation, we may use bilinear or trilinear interpolation for 2D/3D images during warping. More information on this will be provided later.

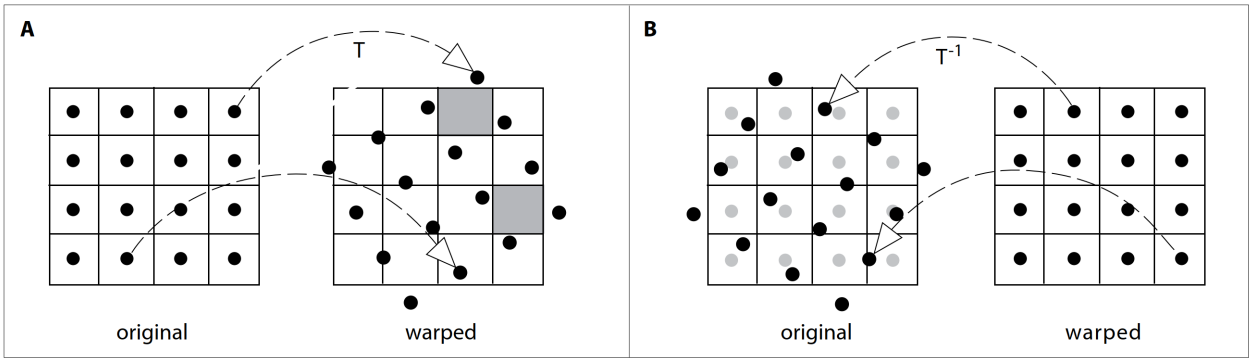

Figure 1: Forward (A) and backward (B) warping illustration. In forward warping, we may get holes (gray cells), while in backward warping, we do not get holes, but we may need to interpolate the pixel values (gray dots) from the mapped pixels (black dots)[11].

4 Network Design

4.1 The Basic Optimization Problem

The registration problem can be defined as an optimization problem that aims to optimize a suitable *loss function* (also called the *energy function*), which is defined as the sum of two terms: the *data term* $\mathcal{L}_{\text{data}}$, which ensures accurate image alignment, and the regularization term $\mathcal{L}_{\text{reg.}}$, which is necessary to mitigate the uncontrollable oscillations of the data term during the optimization process, a consequence of the ill-posed nature of the image registration problem.

There is flexibility in how both terms are defined; therefore, defining them is the main uncertainty in image registration formulation. Typically, they are hand-crafted depending on the problem context and other physical considerations to avoid phenomena such as folding or unrealistic deformations. These difficulties are even more severe if we aim to generalize to other registration contexts.

In general, the data term can be defined as $M(\boldsymbol{I}_f, \hat{\boldsymbol{I}}_f) : \mathbb{R}^{|\boldsymbol{n}_I|} \times \mathbb{R}^{|\boldsymbol{n}_I|} \to \mathbb{R}$, so that it measures the "closeness" between the ground truth fixed image and its estimation. However, selecting a regularization function that effectively penalizes large deviations in the data term is a challenging task and typically involves manual crafting. One of the simplest formulations that has been shown to work well involves some norm of the DF gradients (such as *Total Variation Regularization* [12]). In a more general way, here we will use the following form of the regularization term throughout this work: $R(\boldsymbol{\mathcal{D}}) : \mathbb{R}^{|\boldsymbol{n}_k| \cdot r} \to \mathbb{R}$.

We can now write the target optimization loss function $\mathcal{L} : \mathbb{R}^{|\boldsymbol{n}_I|} \times \mathbb{R}^{|\boldsymbol{n}_I|} \times \mathbb{R}^{|\boldsymbol{n}_k| \cdot r} \to \mathbb{R}$ as:

$$\begin{aligned} \mathcal{L}(\boldsymbol{I}_f, \hat{\boldsymbol{I}}_f, \boldsymbol{\mathcal{D}}) &\coloneqq \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{reg.}} \\ &\coloneqq M(\boldsymbol{I}_f, \hat{\boldsymbol{I}}_f) + \lambda \cdot R(\boldsymbol{\mathcal{D}}) \end{aligned} \tag{11}$$

where λ is a regularization term penalty hyper-parameter. Then the main optimization program becomes:

$$\hat{\boldsymbol{\mathcal{D}}} \coloneqq \underset{\boldsymbol{\mathcal{D}}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{I}_f, \boldsymbol{I}_m, \boldsymbol{\mathcal{D}}) \tag{12}$$

Multiple approaches exist to solve this optimization problem. For example, the *Landweber iterative algorithm* [13] is commonly used for ill-posed linear inverse problems. In this work, we will use a more general but standard method for solving such optimization problems, which is the *gradient descent method*. The DF can then be estimated recursively by the following general expression for some step $t \in \mathbb{N}/\{0\}$:

$$\hat{\boldsymbol{\mathcal{D}}}^{(t+1)} \coloneqq \hat{\boldsymbol{\mathcal{D}}}^{(t)} - \alpha^{(t)} \cdot \nabla_{\boldsymbol{\mathcal{D}}} \mathcal{L}\left(\boldsymbol{I}_f, \boldsymbol{I}_R^{(t)}, \hat{\boldsymbol{\mathcal{D}}}^{(t)}\right) \tag{13}$$

where $\alpha^{(t)} \in \mathbb{R}^+$ is the step size, which can change from one iteration to another.

In this work, we assume a specific form of the data term that we will use. In particular, we assume that it is defined as a function of the fixed and registered images' intensity difference on a per-pixel basis, which we now call *intensity dissimilarity*. That is, we assume that there is a function $\bar{\boldsymbol{M}}$ such that:

$$M\left(\boldsymbol{I}_f, \boldsymbol{I}_R^{(t)}\right) \coloneqq \sum_{\mathbf{p} \in \boldsymbol{\Omega}} \bar{\boldsymbol{M}}\left(\boldsymbol{I}_R^{(t)} - \boldsymbol{I}_f\right)[\mathbf{p}] \tag{14}$$

At this point, to solve this optimization problem, we need to know both $\bar{\boldsymbol{M}}$ and R. However, instead of constructing them by hand, as is done in traditional approaches to image registration, we

aim in this work to formulate them in a way that allows us to "learn" them from the input directly using machine learning approaches. In the following paragraphs, we introduce the necessary tools to achieve that.

4.2 Variational Networks

The iterative formula in eq. (13) that solves the main optimization problem assumes the same data and regularization terms under the gradient at each iteration. *Variational Networks* suggest replacing eq. (13) with $n_u \in \mathbb{N}/\{0\}$ iterative steps:

$$\hat{\boldsymbol{\mathcal{D}}}^{(u+1)} \coloneqq \hat{\boldsymbol{\mathcal{D}}}^{(u)} - \boldsymbol{\mathcal{V}}^{(u)}\left(\boldsymbol{I}_f,\, \boldsymbol{I}_R^{(u)},\, \hat{\boldsymbol{\mathcal{D}}}^{(u)},\, \boldsymbol{\gamma}^{(u)}\right) \tag{15}$$

That is, the gradient term is replaced with a set of parameterized operators $\boldsymbol{\mathcal{V}}^{(u)}$, called *variational units* (VUs for short), where each of these units now has a different set of learnable parameters $\boldsymbol{\gamma}^{(u)}$. This procedure of creating separate VUs for each iteration is called *unfolding* or *unrolling*.

Since VUs now have learnable parameters, it is reasonable to expect that we may need fewer iterations (or in the current context, fewer unfoldings) to achieve relatively good convergence.

In this work, we will still assume that the VUs are a sum of two terms: the data and regularization terms. Moreover, if we assume that the initial condition for the deformation field fed to the first VU is zero everywhere, then by using eqs. (11, 13), we can describe the iterative learning procedure for the parameterized variational network defined in eq. (15) as:

$$\begin{cases} \hat{\boldsymbol{\mathcal{D}}}^{(0)} & \coloneqq \quad \mathbf{0} \\ \hat{\boldsymbol{\mathcal{D}}}^{(u+1)} & \coloneqq \quad \hat{\boldsymbol{\mathcal{D}}}^{(u)} - \boldsymbol{\mathcal{V}}^{(u)} \qquad : \quad \boldsymbol{\mathcal{V}}^{(u)} \coloneqq \alpha^{(u)} \cdot \boldsymbol{\mathcal{V}}_{\text{data}}^{(u)} + \lambda \cdot \boldsymbol{\mathcal{V}}_{\text{reg.}}^{(u)} \end{cases} \tag{16}$$

where, by comparing to eq. (13), and by assuming that $\boldsymbol{\gamma}^{(u)}$, $\bar{\boldsymbol{\gamma}}^{(u)}$ are our new learnable parameters, we can define a correspondence to the data and regularization as new VUs terms:

$$\begin{cases} \boldsymbol{\mathcal{V}}_{\text{data}}^{(u)} & \coloneqq \quad \nabla_{\boldsymbol{\mathcal{D}}}\, M\left(\boldsymbol{I}_f,\, \boldsymbol{I}_R^{(u)},\, \boldsymbol{\gamma}^{(u)}\right) \\ \boldsymbol{\mathcal{V}}_{\text{reg.}}^{(u)} & \coloneqq \quad \nabla_{\boldsymbol{\mathcal{D}}}\, R\left(\hat{\boldsymbol{\mathcal{D}}}^{(u)},\, \bar{\boldsymbol{\gamma}}^{(u)}\right) \end{cases} \tag{17}$$

An illustration of this variational network iterative structure can be found in Fig. 2. One may notice some similarities between the unfolding procedure explained above and *residual networks*. This becomes especially evident by looking at Fig. 2, where we can see that the relations that VUs exhibit, are very similar to how residual networks are constructed. This inherently means that we should expect this network to not suffer from *vanishing gradient* issues, since that was one of the historical purposes of the introduction of residual networks [14].

This procedure allowed us, instead of hand-crafting the data and regularization terms, to parameterize them with a set of learnable parameters $\boldsymbol{\gamma}$. However, we have not yet specified the explicit form of this parameterization. In the context of image registration, it is suitable to use the so-called variational models to parameterize the VUs using the field of experts, which we will briefly review in the following paragraph.

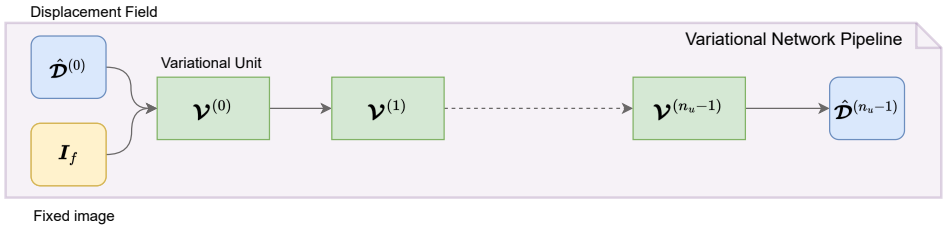

Figure 2: Variational network iterative pipeline.

4.3 Variational Models

Generally speaking, variational models in machine learning have a loose definition. In the current context, a model is considered variational if the corresponding optimization problem aims to minimize a functional of probabilistic distributions that includes, among other terms, a regularization term. The optimization problem defined in eq. (13) can be made variational by introducing the FoE method, which will be summarized as follows.

4.3.1 Fields of Experts (FoE)

The *Fields of Experts (FoE)* method is a probabilistic model, first introduced in [15], that aims to model high-dimensional distributions by a set of low-dimensional distributions, called experts. FoE has been successfully used for various tasks such as image denoising and inpainting [16].

Each expert is designed to model different aspects of features while still capturing the statistical dependencies between those experts.

This simple approach had some shortcomings [15]: it suffered from a large number of parameters, insufficient generalizability, and non-translation invariance. To overcome these issues, generalized models have been proposed that use *Markov random fields*, where only the potentials of *maximal cliques* are modeled with expert products. This made FoE capable of learning highly detailed and complex image priors.

The probabilistic distributions of experts φ_i are usually defined through simple linear filters $\mathbf{J}_i$. These filters are applied to some input $\mathbf{x}$, usually overlapping image patches, and expressed as $\varphi_i(\mathbf{J}_i^\mathsf{T}\mathbf{x};\boldsymbol{\gamma}_i)$, where $\boldsymbol{\gamma}_i$ represents the parameters (we use the same symbol for parameters as in the previous section for a reason that will become apparent in the next section). Originally, the experts were just t-student distributions due to some observations about how experts behave in an image reconstruction context. However, this approach was later generalized, and the target distribution was modeled by a simple (normalized) product of experts.

If we define an optimization problem as a posterior maximization, the logarithm of which is the likelihood function, then the optimization problem becomes a sum of expert logarithms. If gradient descent is used to solve it, the gradient will also allow us to eliminate the logarithms. Note that the logarithms could also be considered part of the experts φ_i from the beginning.

A similar but slightly more general approach will be adopted, as will be demonstrated in the following sections.

4.3.2 The Variational Convolutional Model

In the context of image reconstruction, both [17] and [6] proposed and successfully used data as well as regularization term re-parameterization motivated by the FoE approach. A significant advantage of such FoE re-parameterization is its ability to learn the parameters of the model by considering larger pixel neighborhoods, which enables the learning of more complex dependencies and statistical characteristics within the image, hence leading to better generalization.

The key point is to assume that we can model both data and regularization terms using $n_e, \bar{n}_e \in \mathbb{N}/\{0\}$ experts (for data and for regularization terms correspondingly), and to note that $\varphi, \mathbf{J}$ from the FoE method can be thought of as activation functions and convolutional linear operators respectively (modeled by some kernels of a particular size). Then, we can rewrite the data and regularization terms defined in eqs. (11, 14) using the convolutional operators $\mathbf{K}, \bar{\mathbf{K}}$ and activation functions $\boldsymbol{\varphi}, \bar{\boldsymbol{\varphi}}$ (which now replace both $\boldsymbol{\gamma}$, $\bar{\boldsymbol{\gamma}}$ from eq. (17), and we also drop the u index for simplicity) as follows:

$$\begin{cases} \bar{\boldsymbol{M}}\left(\boldsymbol{I}_R - \boldsymbol{I}_f, \boldsymbol{\gamma}\right) & \rightarrow \quad \displaystyle\sum_{i=1}^{n_e} \boldsymbol{\varphi}_{(i)}\left(\mathbf{K}_{(i)} \cdot \left(\boldsymbol{I}_R - \boldsymbol{I}_f\right)\right) \\ R\left(\hat{\boldsymbol{\mathcal{D}}}, \bar{\boldsymbol{\gamma}}\right) & \rightarrow \quad \displaystyle\sum_{i=1}^{\bar{n}_e} \bar{\boldsymbol{\varphi}}_{(i)}\left(\bar{\mathbf{K}}_{(i)} \cdot \hat{\boldsymbol{\mathcal{D}}}\right) \end{cases} \tag{18}$$

By doing this, we do not need to handcraft an explicit form of the data and regularization terms, since now they can be learned from the training data directly.

To learn the new set of parameters from the FoE re-parameterization, we need to define another optimization problem similar to eq. (12). This new optimization problem will be used to optimize all VU parameters defined in eq. (18) altogether (i.e., $\mathbf{K}, \bar{\mathbf{K}}$ and $\boldsymbol{\varphi}, \bar{\boldsymbol{\varphi}}$), so that the parameters of each VU are optimized to provide the most suitable output for the subsequent VU input, ultimately yielding the best possible prediction of the DF in the final VU.

Furthermore, this also requires introducing another loss function similar to eq. (11). At first glance, it may seem confusing to have two loss functions. However, by optimizing this new loss function, we are automatically optimizing the main loss function introduced in eq. (11). This new loss function is called the *surrogate loss function* [9], and for short, we will refer to it simply as the loss function, or surrogate loss.

Contrary to the constraints that image registration imposes on the construction of the data and regularization terms, we have much more freedom in how we define the surrogate loss. Moreover, there is no need to define or handcraft any regularization terms, since this loss function is related to image registration only implicitly. The only relevant aspect here is that we need to define it in a way that it will be differentiable so that the *autograd* methods available in various machine learning packages can use it for backward propagation. In Sec. 4.7, we will discuss and define a particular form for it.

The last step in the construction of the VUs is to calculate the gradients of the data and regularization terms of eq. (18), which will be the subject of the next section.

4.4 Data Term Gradient

The data term gradient (sometimes also called the *force*) defined in eqs. (14, 17), is taken relative to the DF evaluated at all control points and along all directions. However, to be able to find

the gradient in the context of the re-parameterization defined in eq. (18), we need to re-write the derivative such that it will become relative to the fixed image.

In principle, this can be done easily using the *chain rule*, but it should be done with some care to track the correct dimensions. For simplicity, in what follows, the variational network parameters $\boldsymbol{\gamma}$, $\bar{\boldsymbol{\gamma}}$ are dropped, since they are constant in this context.

First, we will use the sampling operator defined in eq. (7) to write the data term gradient in the i-th direction in terms of the FF (where $i = 1, ..., r$), instead of the DF, next by using also eqs. (6, 14), we can write the following:

$$\begin{aligned}
(\boldsymbol{\mathcal{V}}_{\text{data}}[\mathbf{k}])_i &\coloneqq \frac{\partial M(\boldsymbol{I}_f, \boldsymbol{I}_R)}{\partial \left(\boldsymbol{\mathcal{D}}[\mathbf{k}]\right)_i} = \boldsymbol{\mathcal{S}}_{\boldsymbol{s}_k} \left\{ \frac{\partial M(\boldsymbol{I}_f, \boldsymbol{I}_R)}{\partial \left(\boldsymbol{\mathcal{F}}[\mathbf{k}]\right)_i} \right\} \\
&\coloneqq \boldsymbol{\mathcal{S}}_{\boldsymbol{s}_k} \left\{ \sum_{p \in \boldsymbol{\Omega}} (\boldsymbol{V}^{\mathbf{k}}[\boldsymbol{p}])_i \right\} \quad : \quad (\boldsymbol{V}^{\mathbf{k}})_i \coloneqq \frac{\partial \bar{\boldsymbol{M}}(\boldsymbol{I}_R - \boldsymbol{I}_f)}{\partial \left(\boldsymbol{\mathcal{F}}[\mathbf{k}]\right)_i}
\end{aligned}$$

Now we can apply the chain rule on $\boldsymbol{V}^{\mathbf{k}}$ as follows:

$$(\boldsymbol{V}^{\mathbf{k}})_i = \frac{\partial \bar{\boldsymbol{M}}(\boldsymbol{I}_R(\hat{\boldsymbol{\mathcal{F}}}) - \boldsymbol{I}_f)}{\partial \left(\boldsymbol{\mathcal{F}}[\mathbf{k}]\right)_i} = \frac{\partial \bar{\boldsymbol{M}}(\boldsymbol{I}_R - \boldsymbol{I}_f)}{\partial \left(\boldsymbol{I}_R - \boldsymbol{I}_f\right)} \odot \frac{\partial (\boldsymbol{I}_R(\hat{\boldsymbol{\mathcal{F}}}) - \boldsymbol{I}_f)}{\partial \left(\boldsymbol{\mathcal{F}}[\mathbf{k}]\right)_i}$$

where $\odot$ represents element-wise multiplication.

By applying chain rule again on the last term we can write:

$$\begin{aligned}
(\boldsymbol{V}^{\mathbf{k}})_i &= \frac{\partial \bar{\boldsymbol{M}}}{\partial \boldsymbol{I}_R} \odot \frac{\partial (\boldsymbol{I}_R(\hat{\boldsymbol{\mathcal{F}}}) - \boldsymbol{I}_f)}{\partial (\hat{\boldsymbol{\mathcal{F}}})_i} \odot \frac{\partial (\hat{\boldsymbol{\mathcal{F}}})_i}{\partial \left(\boldsymbol{\mathcal{F}}[\mathbf{k}]\right)_i} \\
&= \frac{\partial \bar{\boldsymbol{M}}}{\partial \boldsymbol{I}_R} \odot \nabla_i \boldsymbol{I}_R \odot (\boldsymbol{G}^{\mathbf{k}})_i \qquad : \quad \boldsymbol{G}^{\mathbf{k}} \coloneqq \frac{\partial \hat{\boldsymbol{\mathcal{F}}}}{\partial \boldsymbol{\mathcal{F}}[\mathbf{k}]}
\end{aligned} \tag{19}$$

The first term represents the rate of change in the dissimilarity measure relative to the estimated fixed image [9]. The second term is simply the image gradient along the i-th direction. These two terms combined are usually referred to as the *Image level force* [11]. Finally, the last term, known as the *Control Points Force*, describes how the interpolated estimation of the FF changes relative to the original FF at the control point $\mathbf{k}$ in the i-th direction. The control points force will be discussed and calculated in Sec. 5.2.

4.5 DIRV-Net

At this point, we have most of the necessary tools to introduce our model, called the *Deformable Image Registration Variational Network* or DIRV-Net for short [9].

By combining eqs. (18, 19), we can express the data and regularization derivatives in eq. (17) as follows:

$$\begin{cases}
\dfrac{\partial \bar{\boldsymbol{M}}(\boldsymbol{I}_R - \boldsymbol{I}_f, \boldsymbol{\gamma})}{\partial \boldsymbol{I}_R} & \rightarrow \quad \displaystyle\sum_{i=1}^{n_e} \mathbf{K}_{(i)}^{\intercal} \cdot \boldsymbol{\psi}_{(i)} \Big(\mathbf{K}_{(i)} \cdot (\boldsymbol{I}_R - \boldsymbol{I}_f) \Big) \\
\nabla_{\boldsymbol{\mathcal{D}}} R(\hat{\boldsymbol{\mathcal{D}}}, \bar{\boldsymbol{\gamma}}) & \rightarrow \quad \displaystyle\sum_{i=1}^{\bar{n}_e} \bar{\mathbf{K}}_{(i)}^{\intercal} \cdot \bar{\boldsymbol{\psi}}_{(i)} \left(\bar{\mathbf{K}}_{(i)} \cdot \hat{\boldsymbol{\mathcal{D}}} \right)
\end{cases} \tag{20}$$

where $\boldsymbol{\psi} := \boldsymbol{\varphi}'$ are called *potential functions*. These functions can be parameterized and learned (instead of $\boldsymbol{\varphi}$) similarly to the linear operators $\mathbf{K}$ (hence, those parameters replaced $\boldsymbol{\gamma}$, $\bar{\boldsymbol{\gamma}}$). We will discuss how to calculate these potential functions in Sec. 5.3.

For simplicity, we rewrite this in terms of what we call *FoE operators* $\boldsymbol{\mathcal{E}}$, with n_k, $\bar{n}_k$ convolutional kernels $\boldsymbol{k}, \bar{\boldsymbol{k}}$, and potential functions $\boldsymbol{\psi}$, $\bar{\boldsymbol{\psi}}$ as follows:

$$\begin{cases} \dfrac{\partial \bar{\boldsymbol{M}}(\boldsymbol{I}_R - \boldsymbol{I}_f, \boldsymbol{\gamma})}{\partial \boldsymbol{I}_R} & \rightarrow \quad \boldsymbol{\mathcal{E}}_{n_k}\Big(\boldsymbol{I}_R - \boldsymbol{I}_f, \boldsymbol{k}, \boldsymbol{\psi}\Big) \\ \nabla_{\boldsymbol{\mathcal{D}}} R(\hat{\boldsymbol{\mathcal{D}}}, \bar{\boldsymbol{\gamma}}) & \rightarrow \quad \boldsymbol{\mathcal{E}}_{\bar{n}_k}\Big(\hat{\boldsymbol{\mathcal{D}}}, \bar{\boldsymbol{k}}, \bar{\boldsymbol{\psi}}\Big) \quad := \quad \boldsymbol{\mathcal{V}}_{\text{reg.}} \end{cases} \tag{21}$$

such that, for some input $\boldsymbol{X}$ that has $i \in \mathbb{N}/\{0\}$ channels or components, the i-th component of $\boldsymbol{\mathcal{E}}_n$ is defined as:

$$\Big(\boldsymbol{\mathcal{E}}_n(\boldsymbol{X}, \boldsymbol{k}, \boldsymbol{\psi})\Big)_i := \sum_{j=1}^{n} \boldsymbol{k}^{(i,j)} \circledast^{\intercal} \boldsymbol{\psi}^{(i,j)} \Big(\boldsymbol{k}^{(i,j)} \circledast (\boldsymbol{X})_i\Big) \tag{22}$$

where $\circledast$ denotes discrete convolution, $\circledast^{\intercal}$ denotes discrete transposed convolution, and the indexes (i, j) represent the j-th kernel/potential function for the i-th channel/component. The structure of the FoE operators can be depicted as in Fig. 3.

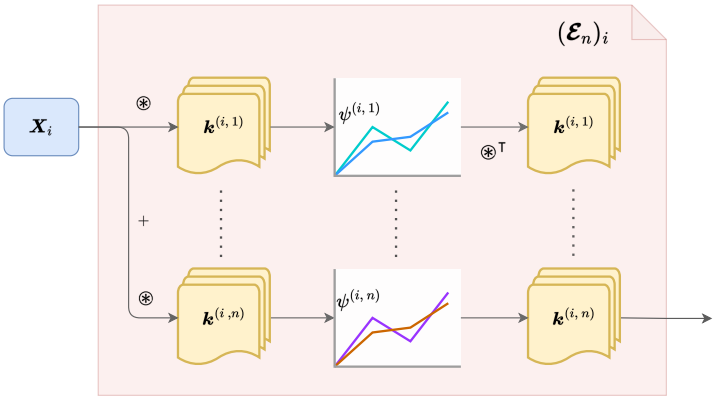

Figure 3: Field of experts diagram.

Up to this point, we have not addressed the fact that the magnitude of the deformations can vary considerably. The next paragraph aims to cover this.

4.6 Resolution Pyramids

The deformations that we aim to register can be local as well as global (affine). To better capture such variations in deformations nature, a standard method proposed by [18] can be used, which is based on creating *image resolution pyramids*. Due to the different resolutions of the images in the pyramid, different scales of features can be captured at different resolution levels. We call each resolution pyramid a *pyramid level* (PL for short), and we assume that they are sorted from the lowest resolution to the highest, up to the original image itself.

We can apply a copy of the model in eq. (16) to every PL image separately. This allows the model to capture relatively large, non-local affine deformations at low-resolution PLs, while other model copies will be better suited for capturing finer and more local features and deformations at higher resolution PLs.

We can junction those PL models similarly to how unfoldings is done. Specifically, we can feed the output of the model from one PL into the input of the model for the next PL. We start with the lowest resolution PL image, where the DF & image size will be the smallest. Then, after a specific number of unfoldings, we feed the result into the next PL model, which follows the same pipeline (see Fig. 4, in contrast to Fig. 2). Finally, this is repeated until we reach the highest resolution PL, i.e., the original moving image size and DF, which will capture the finest deformations in the image. Note that we assume that the separation between the control points is a fixed predefined value in all PLs and VUs.

Of course, this comes at the price of increasing the count of model parameters, so a balance between the number of PLs and VUs should be maintained.

To formalize the above mathematically, let $n_\ell \in \mathbb{N}/\{0\}$ be the number of PLs that we want to have (this number includes the original image size also, so we have $\boldsymbol{I}^{(n_\ell)} \coloneqq \boldsymbol{I}$). In this work, we consider the scaling factor between PLs to be fixed at $s \coloneqq 2$. Then, by using the up/down sampling function we defined in eq. (8), for the pyramid level ℓ we can write:

$$\boldsymbol{I}^{(\ell+1)} \coloneqq \boldsymbol{\mathcal{Z}}_{s\cdot \boldsymbol{n}_{\boldsymbol{I}^{(\ell)}}} \left(\boldsymbol{I}^{(\ell)} \right)$$

By using this, we downsample the original moving and fixed images $\boldsymbol{I}_m, \boldsymbol{I}_f$, and we get the set of PL images $\boldsymbol{I}_m^{(\ell)}$, $\boldsymbol{I}_f^{(\ell)}$.

Feeding the estimated DF from a coarser PL into the next finer PL, which also requires a larger DF (or finer FF), will necessitate upsampling the DF from the previous level. However, just upsampling the field as we did above with PL images is not sufficient, since such upsampling will not rescale the magnitude of the displacement vector field components (this is crucial since DF describes the pixel displacement value on an image that is now twice as big). To address this point, we define an *up/down scaling function* $\boldsymbol{\mathcal{U}}$, which can be defined by using the previously defined up/down sampling function $\boldsymbol{\mathcal{Z}}$, but now we will not only up/down sample but also rescale the DF vector components by a factor of $s = 2$:

$$\begin{aligned} \boldsymbol{\mathcal{D}}^{(\ell+1)} &\coloneqq \boldsymbol{\mathcal{U}}_{s\cdot \boldsymbol{n}_{k^{(\ell)}}} \left(\boldsymbol{\mathcal{D}}^{(\ell)} \right) \\ &\coloneqq s \cdot \boldsymbol{\mathcal{Z}}_{s\cdot \boldsymbol{n}_{k^{(\ell)}}} \left(\boldsymbol{\mathcal{D}}^{(\ell)} \right) \end{aligned} \tag{23}$$

With these tools, we can extend the iterative model in eq. (16) so that it will iterate over n_ℓ pyramid levels (iteration index is ℓ), as well as over n_u variational units (iteration index u), as follows:

$$\begin{cases} \hat{\boldsymbol{\mathcal{D}}}^{(0,0)} \coloneqq \boldsymbol{0} \\ \hat{\boldsymbol{\mathcal{D}}}^{(0,\ell+1)} \coloneqq \boldsymbol{\mathcal{U}}_{\boldsymbol{n}_k^{(\ell+1)}} \left(\hat{\boldsymbol{\mathcal{D}}}^{(n_u-1,\ell)} \right) \quad : \quad \boldsymbol{n}_k^{(\ell+1)} \coloneqq \boldsymbol{n}_{\boldsymbol{I}^{(\ell+1)}} \oslash \boldsymbol{s}_k \\ \hat{\boldsymbol{\mathcal{D}}}^{(u+1,\ell)} \coloneqq \hat{\boldsymbol{\mathcal{D}}}^{(u,\ell)} - \boldsymbol{\mathcal{V}}^{(u,\ell)} \end{cases} \tag{24}$$

where:

$$\boldsymbol{\mathcal{V}}^{(u,\ell)} \coloneqq \boldsymbol{\alpha}^{(u,\ell)} \odot \left(\boldsymbol{\mathcal{V}}_{\text{data}}^{(u,\ell)} + \boldsymbol{\mathcal{V}}_{\text{reg.}}^{(u,\ell)} \right) \tag{25}$$

which is a slightly modified version of eq. (16). Here, $\boldsymbol{\alpha}^{(u,\ell)} \subset \mathbb{R}^+$ are the learning rates, which are now also part of the learnable parameters in our model, and they absorb the regularization penalty λ. This is justified because this penalty, typically imposed to prevent overfitting and enforce

normalization over the regularization term's smoothness and magnitude, is no longer a concern, as our learnable VUs should address these issues automatically, while by keeping the learning rates, we preserve the ability to interpret the data term meaningfully. Finally, the components of VUs $\boldsymbol{\mathcal{V}}^{(u,\ell)}$ are already obtained in eqs. (19, 21).

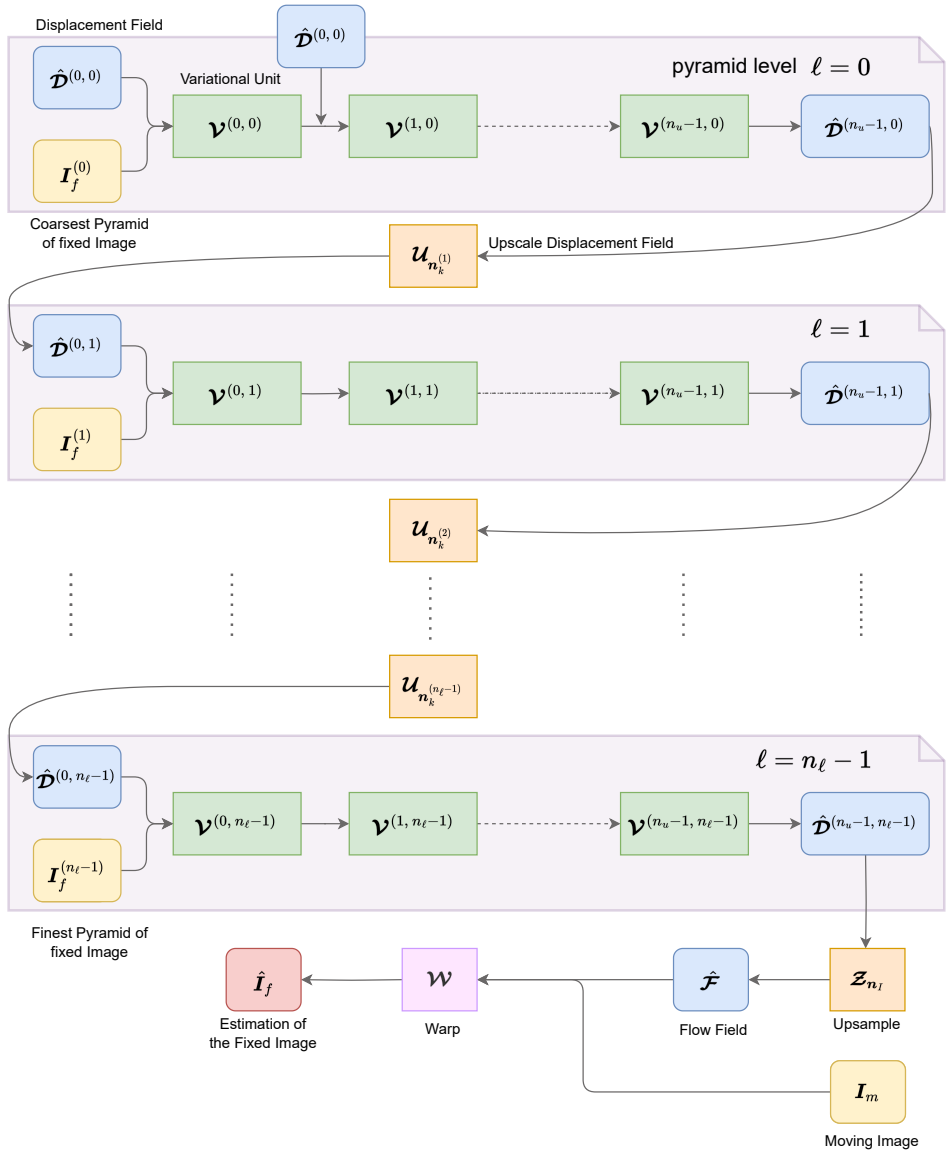

Figure 4: The full pipeline diagram of DIRV-Net.

4.7 Surrogate Loss Function

The last piece to define in this model is the surrogate loss function, which will control how all VUs across all PLs defined in eq. (24) will be optimized during the learning process.

There is some flexibility in defining this function. The most intuitive method is to measure the image dissimilarity between the registered image (i.e., the estimated fixed image) and the ground truth fixed image, or the so-called *sum of squared distances (SSD)* (distances here will be the

dissimilarity). SSD is simply the squared norm of the difference between the fixed and registered image intensities (recall that, in the same way, we also assumed a difference in the main loss function in eq. (14)), and hence it will be easily differentiable for gradient descent.

To encourage all parameters in the PLs to learn effectively, we may measure the dissimilarity at every level. The most straightforward approach is to calculate the loss as the averaged SSD of the image output of the last VU of every PL. Since at each PL we have images of a different resolution, we need to calculate SSD on a per-pixel basis to make the intermediate error of each PL comparable to the next one. Mathematically, we can describe such a surrogate loss function based on intensity dissimilarity as:

$$\begin{cases} \tilde{L}_{\text{int.}}^{(\ell)}[\mathbf{p}] & \coloneqq & \sqrt{\frac{1}{c}\left\|\boldsymbol{I}_f^{(\ell)}[\mathbf{p}] - \boldsymbol{I}_R^{(n_u-1,\,\ell)}[\mathbf{p}]\right\|^2} \\ L_{\text{int.}}^{(\ell)} & \coloneqq & \dfrac{1}{|\boldsymbol{n}_{\boldsymbol{I}}|}\displaystyle\sum_{\mathbf{p}\in\Omega}\tilde{L}_{\text{int.}}^{(\ell)}[\mathbf{p}] \\ L_{\text{int.}} & \coloneqq & \dfrac{1}{\ell}\displaystyle\sum_{\ell}^{n_\ell-1} L_{\text{int.}}^{(\ell)} \end{cases} \tag{26}$$

where $\tilde{L}_{\text{int.}}^{(\ell)}$ is the square root of the channels-averaged squared norm of the difference between the ground truth fixed image at the ℓ'th PL, and the registered image as an output of the n_u'th VU (i.e., the last VU for each PL). Then, to get a pixel average intensity dissimilarity $L_{\text{int.}}^{(\ell)}$, we average $\tilde{L}_{\text{int.}}^{(\ell)}$ over all image pixels. Finally, we calculate the pyramid levels-averaged $L_{\text{int.}}^{(\ell)}$ to arrive at the *intensity dissimilarity-based surrogate loss function* $L_{\text{int.}}$, or "intensity dissimilarity" for short.

The intensity dissimilarity introduced is sufficient to train DIRV-Net in an *unsupervised* manner, i.e., when the ground truth displacement fields are unknown. However, when the deformations between fixed and moving images are known (e.g., produced by simulated synthetic deformations), we can use the same approach to measure the dissimilarity between the ground truth DFs and the estimated one in our model, based on control points instead of pixels, and hence train our model in a *supervised* manner, which will be the focus of the current work (for a reason that will be explained latter).

However, since for DFs, we are using the scaling operator defined in eq. (23) when we go from one PL to the next, we should also similarly rescale the SSD of different PLs to maintain a unified scale and make the control point-averaged level error comparable. For that, we slightly alter eq. (26) by using eq. (24) as follows:

$$\begin{cases} \tilde{L}_{\text{dis.}}^{(\ell)}[\mathbf{k}] & \coloneqq & \sqrt{\frac{1}{r}\left\|\boldsymbol{\mathcal{D}}^{(\ell)}[\mathbf{k}] - \hat{\boldsymbol{\mathcal{D}}}^{(n_u-1,\,\ell)}[\mathbf{k}]\right\|^2} \\ L_{\text{dis.}}^{(\ell)} & \coloneqq & \dfrac{1}{|\boldsymbol{n}_k|}\displaystyle\sum_{\mathbf{k}\in\bar{\Omega}}\tilde{L}_{\boldsymbol{\mathcal{D}}}^{(\ell)}[\mathbf{k}] \\ L_{\text{dis.}} & \coloneqq & \dfrac{1}{\ell}\displaystyle\sum_{\ell}^{n_\ell-1} s^{2(n_\ell-\ell-1)}\cdot L_{\text{dis.}}^{(\ell)} \end{cases} \tag{27}$$

where in $\tilde{L}_{\text{dis.}}^{(\ell)}[\mathbf{k}]$ we now average by vector field components count instead of channels count, and in $L_{\text{dis.}}$, before averaging $L_{\text{dis.}}^{(\ell)}$, we scale it up according to what we did in the scaling operator. By this, we defined a *displacement dissimilarity-based surrogate loss function* $L_{\text{dis.}}$, or "displacement dissimilarity" for short.

5 Implementation Aspects

In this section, we will briefly discuss some of the technical nuances and numerical methods used in this work.

5.1 Deformations and Image Up/Down Sampling

In this work, we will need various interpolation approaches, as will be pointed out below. Moreover, we need these approaches to handle both 2D and 3D images. Due to the lack of many such functions in the standard packages, we had to implement this functionality from scratch.[2]

Upsampling and Downsampling Since there are no built-in functionalities that work with both two and three-dimensional images, we implemented linear and nearest neighbor up/down sampling using convolutional kernels. For downsampling, a strided convolution is used.

This approach is not the most accurate, as it may lead to some information loss and artifacts that can affect image features. Nevertheless, this is compensated for by its speed and suitability for GPU processing.

However, this approach also imposes some limitations on the possible image sizes we can feed into DIRV-Net. This is one of the reasons why we will limit ourselves to images with sizes that are powers of 2. We would like to emphasize that this is a technical and artificial limitation, not one due to the network architecture.

Gaussian Smoothing Direct usage of the nearest neighbor for up/down sampling may result in artifacts that can be mitigated by applying Gaussian smoothing.

For that, we approximate the Gaussian kernel, for speed considerations, with the *binomial kernel* of order 3, which has a size of 5 per dimension. When the images are relatively small, we instead use binomial kernels of order 1, which have a size of 3 per dimension.

To avoid border artifacts caused by smoothing, we also implemented an automatic padding procedure with reflective boundary conditions before performing up/down sampling.

Linear Interpolation To generate image PLs, we need to downsample images. We use linear interpolation with smoothing for this procedure.

For generating the PLs from DFs, which are needed to calculate the surrogate loss function, we used the nearest neighbor with smoothing.

For the up/down scaling operator used to push DFs from one PL to the next, we use the nearest neighbor method to minimize field manipulations during this procedure.

For image warping, we implemented bi-linear interpolation for 2D images, and tri-linear interpolation for 3D images.

[2]Most of the ML libraries lack this functionality, especially for 3D images. Others implement them partially, such as supporting either upsampling or downsampling, or accepting only integer scaling factors. Contrary to that, many CPU-based frameworks have those functions.

B-Splines B-Splines, in general, provide smoother interpolation output compared to other methods we used in this work and will be primarily used for DF interpolation to the FFs.

Since our fields may exhibit relatively sharp changes due to their random nature, we will restrict ourselves to only first-degree B-Spline interpolation to avoid the *overshooting* behavior of higher-degree B-Splines that may occur during the approximation of sharp transitions. This will also ensure that the results are bounded by the original displacements [19]. A good introduction to B-Splines can be found in [11].

One of the main reasons why we are using B-Splines for field interpolations is their ease of differentiation, which will allow us to calculate the control points force, as will be discussed in Sec. 5.2.

5.2 Spline Linear Smoothing Kernel

The last term that we need to evaluate in eq. (19) is the control points force $\boldsymbol{G}^{\mathbf{k}}$. As mentioned above, we are using first-degree B-Splines to perform the DF to FF interpolation, as defined in eq. (23). The main benefit of using B-Splines is to simplify the calculation of its derivative, as will be described in what follows.

It has been shown in [11] that this derivative is zero everywhere except in the neighborhood of the control points (i.e., where we assumed that the FF is near-constant). Moreover, it does not depend on the underlying FFs, DFs, or the positioning of the control points within the grid.

It has also been shown that the control points force in a *relative* neighborhood $\mathbf{h} \in \prod_i [-(\mathbf{s}_k)_i, (\mathbf{s}_k)_i] \subset \boldsymbol{\Omega}$ of an *arbitrary* control point $\mathbf{k}$, takes the following constant values:

$$\boldsymbol{g}[\mathbf{h}] = \prod_i^r \left(1 - \left|\frac{(\mathbf{h})_i}{(\mathbf{s}_k)_i}\right|\right)$$

This allows us to treat it as a discrete convolution kernel (see Fig. 5), which is called a *linear smoothing kernel*, and allows us to rewrite eq. (19) in a convolutional form.

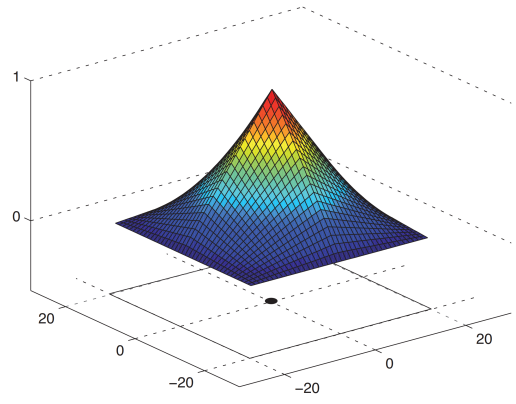

Figure 5: A plot of a linear smoothing kernel around an arbitrary control point [11].

To rewrite eq. (19) in a non-component-based form, we define a new operator, which we call the *tiling operator* $\boldsymbol{\mathcal{T}}_r$. This operator clones the input r times and stacks the repeated clones together, essentially increasing the number of channels. By this, the eqs. (19, 21) can be combined in a

convolutional manner as:

$$\hat{\boldsymbol{\mathcal{V}}}_{\text{data}} \backsimeq \boldsymbol{\mathcal{S}}_{\boldsymbol{s}_k}\left\{\left(\boldsymbol{\mathcal{T}}_r(\boldsymbol{\mathcal{E}}_{n_k}) \odot \nabla \boldsymbol{I}_R\right) \circledast \boldsymbol{g}\right\} \tag{28}$$

Since the smoothing kernel depends on the control points' spacings, and as mentioned before, in this model, we consider those spacings to be fixed numbers in all directions and among all PLs, we can pre-calculate and reuse the kernel $\boldsymbol{g}$ in all iterative steps, which can save some processing resources.

By using this machinery, and especially eqs. (21, 25, 28), we can finally construct the full VU pipeline that we depicted in the following Fig. 6.

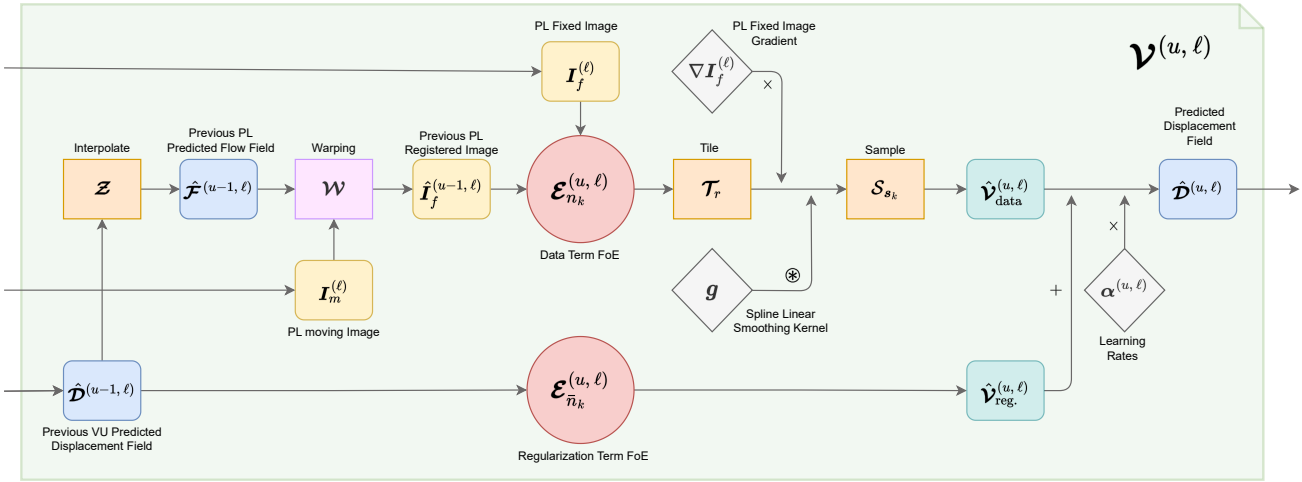

Figure 6: Variational unit pipeline diagram of DIRV-Net.

5.3 Potential Function Parameterization

The potential functions we defined in eq. (22) should also be learned along with the rest of the network parameters.

These single-variable functions can be discretized/parameterized in various ways. In this work, we will parameterize them using piecewise linear interpolation, where these lines connect a predefined number of knots, that can be learned along with the rest of the parameters.

Furthermore, we can integrate the area below these functions to finally obtain the shape of the actual activation functions. See Fig. 7 for visualization of both potential and activation functions.

5.4 Synthetic Transformations

As mentioned before, we will focus on the supervised regime of the proposed model, i.e., we will generate the deformation fields syntactically, and consider the ground truth DFs/FFs as known.

Multiple approaches are proposed to generate deformation fields. The most robust methods are likely model-based transformation generators (see, for example, [20]). However, in this work, the image deformations will be generated synthetically by creating random local non-rigid transformations, as well as global affine transformations.

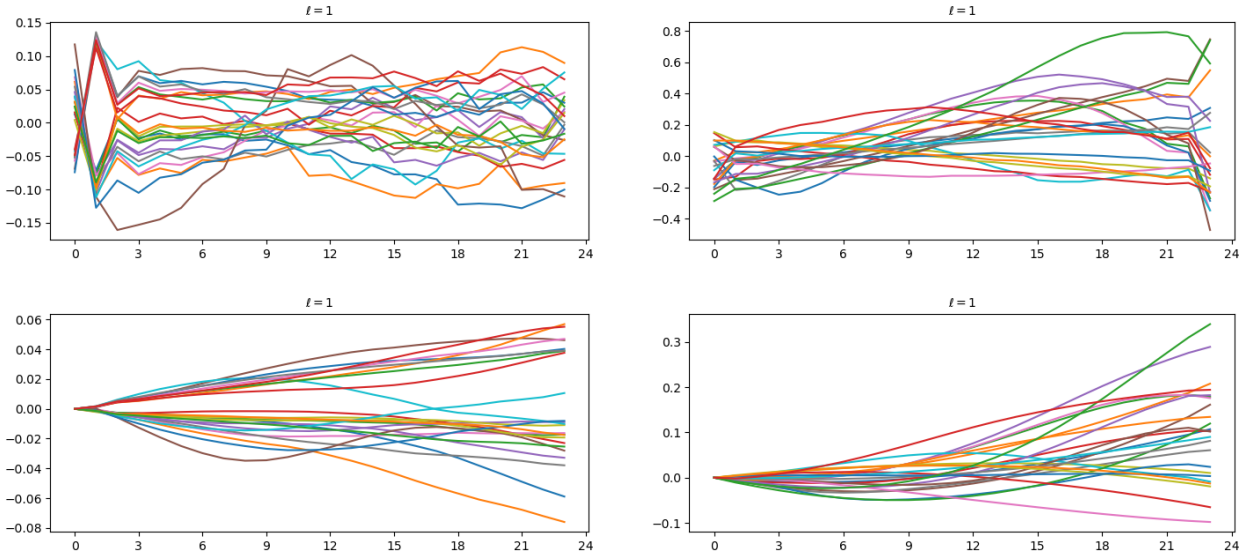

Figure 7: An example of piecewise linearly parameterized (by 24 knots, presented as x-axis) potential functions (first row) for data (left column) and regularization terms (right column), and their integration result, i.e., the activation functions (second row). These functions belong to the last VU in the D-2 run described in Tbl. 4. Note that there is no correspondence between the curve colors across multiple plots.

Affine transformations, which consist of 6/15 parameters for 2D/3D transformations correspondingly, are implemented from scratch. Usually, these transformations are applied to the center of the image. However, to make them even more realistic, we slightly randomize the application center of coordinates around the image center. Hence, our implementation of the affine transformations has $6 + 2 = 8$ and $15 + 3 = 18$ parameters for 2D/3D.

Random non-rigid fields are smoothed with multiple Gaussian kernels and generated with random complexity (in terms of how many times the fields go through zero along one image) and of random magnitude to better resemble realistic scenarios and to avoid possible artifacts that arise from overly simplistic approaches in the generation of such fields.

5.5 The Swapping Trick

Our goal is to predict the FFs, which we defined as those that deform the moving images back to the fixed images, and hence serve as the inverse of transformations (see eq. (5)).

However, initially, we have only the fixed images during the learning process, and we aim to generate the deformed moving images from the fixed ones. To achieve this, we also need to know the inverse of the FFs to warp them with the fixed images, and eventually to obtain the moving images.

Unfortunately, inverting a general vector field is an iterative procedure, and to obtain a high-quality result, a very computationally expensive process is needed, especially for 3D images. Therefore, we would like to avoid field inversion altogether. Although multiplying the fields by -1 offers a very fast and simple approximation for inversion, this approach would result in a considerable loss of information and inaccuracies.

A simple way to mitigate this is by using what we call the *Swapping Trick*, which consists of the following steps (see Fig. 8 for a visual representation):

1. Generate synthetic fields.
2. Consider them to represent the inverse of the FFs for now.
3. Apply them (see eq. (10)) to the fixed images to generate the moving images.
4. Swap the fixed images with the generated moving images.
5. In doing so, the synthetic fields now actually represent the direct FFs.

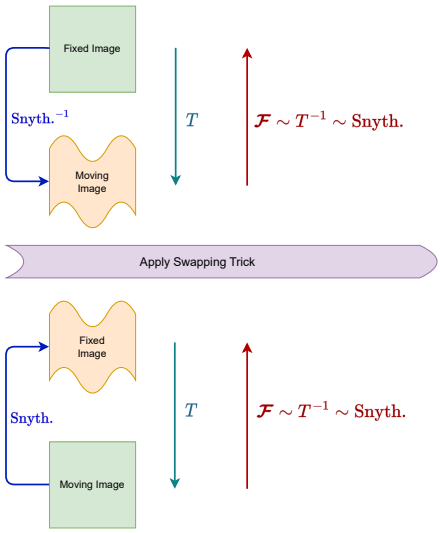

Figure 8: A diagram that illustrates how the swapping trick works and eliminates the need to calculate the inverse of the synthetic fields.

In this way, we have the exact FFs that we need to predict without the need to invert any vector fields, thus saving a lot of CPU cycles.

However, this approach has two small drawbacks:

1. The network will predict how to deform original images, not how to invert deformations, which is slightly counterintuitive.
2. Generally speaking, inverse vector fields are usually mathematically more complex than the "direct" once (e.g., they could be singular). However, since we are dealing with smooth vector fields in this project, this issue should not impact our results.

Still, using the trick is justified when working with relatively large image sizes, as it makes the calculations less resource-intensive ($\approx$ 40 - 50% reduction in the needed GPU memory). Additionally, we avoid the quality loss during the inverse fields calculation.

5.6 Learning on Image Patches

Our network may need significant resources in terms of GPU memory during training, especially when dealing with high-resolution medical 2D/3D images. This primarily happens due to the long pipeline that the model introduces (see Figs. 4, 6).

To resolve this issue with the limited resources available to us, we took advantage of the fact that DIRV-Net is primarily a convolutional network and that our model is DF density agnostic. This implies that the size of image dimensions should not significantly affect network performance as long as the image features are well distinguishable. Hence, we may be able to train the network on smaller patches of the input images instead of feeding the whole high-resolution image into the network pipeline.

To achieve this, we implemented the capability to generate batches by randomly selecting a fixed image, taking a random patch from it, and randomly applying deformation fields to generate the corresponding moving image. This allows us to feed high-resolution images to the DIRV-Net network while maintaining moderate resource usage and network performance.

This is by no means a perfect solution. Some of the drawbacks of this approach are:

- A patch may miss important features that are essential for correct image registration.
- A patch may contain many texture-free regions, which will prevent any sensible image registration.

We overcome the first drawback by using datasets of feature-rich images in our experiments, and we overcome the second one by discarding mostly empty patches. Discarding mostly empty patches was done by calculating the patch intensity standard deviation (STD) and discarding any patch that has an STD smaller than a certain threshold value. Doing this by STD calculation is also not a perfect solution in general, but it worked well enough for the datasets used in this work.

5.7 Sensitivity to Deformation Noise

We found experimentally that any FFs with a component strength of less than 0.005 lead to image warping errors of less than 1 in pixel intensities, when our interpolation methods are used for that.

To account for this, we considered any fields of such magnitude as white noise and ignored them by trimming them to zero.

Additionally, with a predefined probability, we trained the network to treat such small fields as an indication of the absence of deformation. Moreover, with some other probability, we "degraded" our synthetic deformation fields with truncated white noise (truncated by the above-mentioned value) during warping, but we still considered the non-degraded version of those deformations, as the ground truth for surrogate loss function, to teach our network to ignore them.

By doing this, we managed to reduce the network's sensitivity to white noise deformations and improved its performance by around 3% (in terms of the resultant average displacement dissimilarities), as indicated by our preliminary experiments (not presented in this report for sake of brevity).

This insensitivity also proved to be useful for the next section.

5.8 Recurrent Refinement

In most of the experimentally tested cases, the following behavior is observed:

- If the fields have too strong deformation in some regions but not in others, the model tends to underestimate the strong regions of the field.

- Our model hits a plateau in prediction performance at some point, and increasing the count of the model's parameters does not significantly affect this.

Overall, this reduces the so-called *capturing range*, i.e., how much the model can revert the moving image deformations in the default model setup.

Since the moving image will become "closer" to the targeted fixed image after registration, we can use this registered image as a new moving image and feed it into our network again for registration.

Moreover, this can be done recursively multiple times, with the hope of getting a progressively closer registered image in each iteration. We will call this procedure *recurrent refinement*.

However, at some iteration, the resultant registered image may become too close to the moving image, so that the network becomes incapable of registering the image appropriately, which may eventually degrade the final results.

To automate this, and since we lack the ground truth DFs in the subsequent iterations[3], we monitored how the intensity dissimilarity and image difference standard deviation (STD) change from one iteration to the next, and we stop once those values start to increase instead of decrease. Our experiments showed that monitoring the standard deviation provides better results, so we use that as a measure of convergence.

This approach has some drawbacks, such as slower inference due to multiple iterations. Also, various errors may accumulate during such a cascading process.

There are two ways to apply this procedure: on a per-image basis and on a per-batch refinement basis. The per-batch refinement could be less accurate since the iteration will stop based on the collective STD rather than the single image STD. This becomes more relevant if the batch size is too big. On the other hand, the per-batch refinement should be considerably faster than the per-image one[4]; however, this may be mitigated by parallelizing the inference process. Our experiments showed a considerable improvement ($\approx 30\%$) in the averaged results over the initial image registration.

[3] They can actually be approximated by subtracting the predicted DFs in the previous iteration from the original ground truth displacements. However, this will introduce a new source of errors into our implementation that we want to avoid.

[4] This happens due to the need to initialize the DIRV-Net pipeline less frequently, as well as the reduction in how many times data should be shuffled between RAM and VRAM when batches are used.

6 Experiments

6.1 Used Hardware and Software

The network is built using Python v3.9.19, and primarily TensorFlow v2.10.0. Various other standard Python libraries are also used, such as Tensorboard for reporting.

Training and inference are performed on an Nvidia Quadro RTX 6000 GPU with 24GB of graphical memory and an AMD Ryzen Threadripper PRO 3945WX 12-Core CPU, with 48GB of RAM.

The code is written with extensibility in mind, using OOP concepts and different abstractions motivated by our mathematical formulation for easy maintainability.

6.2 Used Datasets

By default, all used images are normalized to the $[0, 1]$ domain of pixel intensities for better learning performance. The following datasets were used for evaluation:

Gradient Chessboard: Visually inspecting deformations on detailed medical images is somewhat challenging, so we generated random synthetic images that resemble a chessboard-like structure with gradient gray shades from corner to corner. The size of the chessboard cells or grid and the gradient shades are generated randomly within a predefined range. Also, we had the flexibility to generate 2D as well as 3D images of any size and any level of detail (i.e., cell size).

This simplified spotting issues and verifying our model's correctness at different processing stages, which was essential given the numerous steps in our processing pipeline. Furthermore, it serves as a synthetic baseline for our network performance. Example from this dataset can be seen later in Fig. 9.

ANHIR Challenge Dataset: This dataset comes from a challenge of nonlinear image registration of 2D very high-resolution (around 50 thousand pixels per dimension) microscopy images of histopathology tissue stained with different dyes [21]. The dataset consists of 160 training fixed and moving images, with the corresponding landmarks provided. Also, it contains 64 testing fixed and moving images with no landmarks provided.

Due to the high resolution of these images, which requires hardware resources that exceed those we had available by multiple orders of magnitude, we needed to crop them to a smaller size. However, due to the sparsely defined landmarks, they became unusable in this work after cropping. Consequently, we are unable to compare our results with those of other methods in this challenge, as they solely rely on these landmarks.

Thus, to utilize this dataset, we generated 1614 training and 389 testing monochrome images by first generating 2000×2000 pixel images by cropping original images from the center, and then taking random patches of 256×256 pixels in size. We also skipped any patches that had a standard deviation in pixel intensities of less than 10 (out of 255) to discard images containing many empty areas with little tissue information.

The images in this dataset are more complex; tissue structure and pixel intensity considerably vary across the dataset (see latter Fig. 13), so we normalized each image individually to the [0, 1] domain of intensities.

6.3 Hyperparameters and Settings

Some of the hyper-parameters used in our setup are shared among all runs, and they were sufficient to achieve satisfactory results in general. We will list them here along with other parameters that may be needed for reproducibility.

- We use the *Adam optimizer* for all runs, with a fixed learning rate of 2^{-4}, and first and second betas equal to 0.9 and 0.999, respectively. This was appropriate for relatively stable convergence in our experiments.
- We used fixed sizes for convolutional kernels: 7 pixels per dimension for data term kernels and 5 pixels per dimension for the regularization term kernels, since we wanted the data term to capture bigger structures and features.
- Potential functions were line-wise parameterized by 24 knots, which we found to be sufficient for most cases.
- Control point spacing for DFs was always 4 pixels. This allowed DFs to not skip relatively small features while avoiding noisy artifacts and still being able to reduce the number of required parameters.
- Image up/down sampling was performed using linear interpolation with Gaussian smoothing. Upward interpolation of DFs to FFs was performed with first-degree B-Splines. Upgrading DFs (between one PL and the next one) and generating the ground truth DFs for PLs were performed using the nearest neighbor method to avoid any smoothing. Interpolation approaches were chosen to ensure the best quality when dealing with images and minimal interference when manipulating flow or displacement fields.
- Learning rate model parameters are always initialized with random numbers in the range [0, 0.1], while potential functions are initialized with a truncated normal distribution with a standard deviation of 0.01 to simulate standard activation functions in ML initially.
- We used a displacement dissimilarity-based surrogate loss function as defined in eq. (27) (averaged by the number of images in the current batch). The reason will be justified later on.
- We used 42 as a seed for the training random generator and 77 for testing.
- The maximum recurrent refinements iteration count had been capped at 15.
- White noise poisoning probability was set to 1%. Pure white noise deformations are injected into the dataset with a 0.1% probability, and the white noise strength threshold is 0.005.

There are also numerous additional parameters that define how we generate the synthetic fields, and we will not list them since they are not essential. At least the non-rigid fields' complexity (i.e., the number of times they pass through zero) was kept constant throughout all runs.

Moreover, to make it easier to compare the visual output of our runs, we implemented the ability to reserve and freeze the first batch, and make it last one in all epochs so that it will always contain the same images and the same synthetic deformation fields. Hence, it will be easier to inspect and compare the results between consecutive epochs or even different runs.

6.4 Presentation of Results

Since we aim to explore and understand DRIV-Net behavior rather than tuning the parameters for the best performance, we will report training and testing results only. When the testing results are not essential to make arguments, we will not report them to reduce clutter.

Also, whenever a pixel intensity-related parameter is reported, we assume that it is in the grayscale intensity [0, 255] domain, to make it more intuitive.

Experiments will be performed in multiple steps, where in each step we investigate network behavior in increasingly complex setups, with more challenging datasets or larger image sizes.

All of the reported results are from the original capturing range by default, i.e., without using recurrent refinement, unless explicitly mentioned.

6.4.1 Table Contents Description

We will report all results in tables, which will consist of the following columns:

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\boldsymbol{\mathcal{D}}\|\|$	$\overline{\|\|\boldsymbol{\mathcal{D}}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$

Below is a description and the meaning of each column. When we mention the "last epoch", we mean the last training epoch or the entire testing procedure since testing consists of virtually one epoch:

- Run ID: We will give each experimental run an ID so that we can reference it in the text. To distinguish training results from testing results, we will add the suffix "t" to the testing run ID.
- Img./Pat.: The image size used and, if relevant, the size of the randomly generated image patches for training.
- Feat. size: The range of the feature size of the synthetically generated images; this will be applicable only to the chessboard images. It refers to the range in pixels that the randomly generated board cells can take in size.
- Type: Describes the type of synthetically generated transformation fields. It can be either "Affine", "NonRig." for non-rigid transformations, or "Both" for the sum of both types.
- $\max ||\boldsymbol{\mathcal{D}}||$: Since we can generate fields of various strengths, and the larger the image size is, the stronger DFs we would like to apply. This column will report the maximum norm of the DFs that have been applied during this run across all epochs.
- $\overline{||\boldsymbol{\mathcal{D}}||}$: Similar to the previous one, but describes the mean norm of the displacement vector fields.
- Ep. Cnt.: Describes how many epochs the network was trained on. Not applicable for testing. We report this because not all runs were executed until "full" convergence since that was not our goal, and because this may affect the maximum and average of the randomly generated DFs.

- Bat./Pat.: Describes the batch size that was used during training and how many patches were generated from each batch image, if applicable. Not relevant for testing. This may also affect the maximum and average of the randomly generated DFs.
- n_ℓ: The number of the used PLs in the model, as described in Sec. 4.6.
- n_u: The number of the used VUs per all of the PL, as defined in Sec. 4.2.
- $n_e, \bar{n}_e$: The number of convolutional kernels/filters used to re-parametrize the data and regularization terms consequently. We defined these in Sec. 4.3.2. In this work, we will always use the same number of kernels for both terms.
- $\sum$: The calculated total number of learning parameters in DIRV-Net with the current setup of hyperparameters, and the used core rank of images.
- $\bar{L}_{\text{int.}}$: The averaged intensity dissimilarity (per pixel) of all images in the last epoch. This is calculated based on eq. (26). The closer this value is to its minimum, 0, the better.
- $\bar{L}_{\text{dis.}}$: Similar to the previous column, this is the averaged DF dissimilarity of the applied fields on the analyzed images in the last epoch, and it is based on eq. (27). It is measured in pixels per pixel.
- $\Delta\bar{L}_{\text{int.}}$: The reduction, in percentage, in the average intensity dissimilarity in the last epoch, i.e., it describes how much the model managed to decrease the average intensity dissimilarity $\bar{L}_{\boldsymbol{I}}$ compared to the initial average intensity, and both of which use eq. (26). Note that the initial average dissimilarity is calculated on all fixed and moving images that were used in all epochs, in contrast to the average dissimilarity of only the last epoch. The closer this value is to its maximum of 100% (which means initial deformations are fully resolved), the better.
- $\Delta\bar{L}_{\text{dis.}}$: Similar to the previous column, but for displacement dissimilarity.

Since we will sometimes apply deformation fields of different strengths, especially when the image sizes differ, and because the datasets used vary in their intensity levels, we will mainly use $\Delta\bar{L}_{\text{int.}}$ and $\Delta\bar{L}_{\text{dis.}}$ to compare the model's performance in various configurations. We use the results of the runs on test sets (which have "t" prefixes) for performance comparison.

The hyperparameters n_ℓ, n_u, $n_e = \bar{n}_e$ are the main parameters that we will vary during our experiments since they are the primary settings that determine the total number of model parameters and its structure (however, another factor that affects this is the dataset images core rank).

6.4.2 Patch and Patch-Free Training

We distinguish between two types of experiments for both 2D and 3D experiments. The first type will be patch-free, training the network on the entire images. The second type will utilize image patches for training, as discussed in Sec. 5.6, while still feeding to the network the entire images during inference process, thanks to the fully convolutional nature of our model. This will give us a better view of how the model generalizes with patches when resources are limited, e.g., with larger image sizes.

6.5 2D Experiments

In the initial steps of the experiments, we aim to understand the behavior of the network with various model hyperparameters, such as the effect of dataset properties on learning quality (since we can adjust them in the synthetic chessboard dataset). We hope to find an optimal configuration that achieves favorable output without requiring extensive resources through this approach.

6.5.1 Patch-Free Experiments

Small Chessboard Images: We start by examining the network performance with small 64×64 pixel images, without using any patching. In the following Tbl. 1, we report some metrics of the performed runs:

Table 1: Results of training the network on $10,000$ random 64×64 chessboard images, and similarly for testing, without using patches. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
A-1	64	[6, 10]	Both	10.18	1.97	25	100	3	5	16	41,055	4.65	0.69	60.7%	51.8%
A-1t	64	[6, 10]	Both	8.63	1.98	-	-	3	5	16	41,055	4.74	0.70	60.5%	51.6%
A-2	64	[6, 10]	Both	10.19	1.97	25	100	3	5	32	82,095	4.51	0.67	61.8%	53.7%
A-3	64	[6, 10]	Both	8.29	1.96	25	100	4	5	16	54,740	4.13	0.88	56.6%	38.9%
A-4	64	[6, 10]	Both	10.16	1.98	25	50	3	7	24	86,205	4.55	0.69	61.9%	52.5%
A-4t	64	[6, 10]	Both	8.29	1.96	-	-	3	7	24	86,205	4.53	0.68	62.0%	52.9%
A-5	64	[6, 10]	Both	10.18	1.97	25	100	2	7	24	57,470	4.12	0.53	70.6%	63.0%
A-5t	64	[6, 10]	Both	8.64	1.96	-	-	2	7	24	57,470	4.31	0.55	69.7%	61.9%
A-6	64	[6, 10]	Both	10.15	1.97	40	100	1	7	24	28,735	3.53	0.43	76.9%	69.6%
A-6t	64	[6, 10]	Both	8.64	1.95	-	-	1	7	24	28,735	3.74	0.45	75.9%	68.6%
A-7	64	[6, 10]	Both	10.20	1.95	60	100	1	7	32	40,999	3.39	0.40	77.8%	71.7%
A-7c	64	[6, 10]	Both	8.73	1.95	-	-	1	7	32	40,999	3.40	0.40	77.8%	71.7%
A-8	64	[6, 10]	Both	10.18	2.01	60	100	1	9	24	36,945	3.06	1.39	79.3%	73.2%
A-8t	64	[6, 10]	Both	8.57	1.98	-	-	1	9	24	36,945	3.00	0.36	**79.7**%	**73.6**%

The A-1 run is an initial parameter setup, and the testing results of A-1t showed no signs of overfitting. In the next A-2 run, we doubled the number of kernels, which led to a doubling of the total number of network parameters. However, the improvement in both dissimilarity measures was minor. In the subsequent run, A-3, we increased the number of PLs instead, but this degraded the dissimilarity measures. This occurred because the smallest PL was only 4×4 pixels, which provided very little meaningful information for image registration.

Therefore, we reduced the PL count and increased the number of unfoldings to 7 in A-4, with a slightly higher number of kernels. This resulted in a similar number of parameters and performance as in A-2. In A-5 and A-6, we began reducing the number of PLs. Although this led to fewer parameters, it led to better testing performance output while still showing no signs of overfitting. We present some A-5t results in Figs. 9, 10.

At this point, we noticed that having fewer PLs requires more epochs to converge (convergence here means that the last few epochs did not result in any improvements in network performance) (see Fig. 11). Hence, we increased the epoch count. In A-7, we again tried to see the effect of increasing

the kernel count but with fewer PLs, which led to a significant increase in the parameter count and a slight increment in performance.

In the last A-8 run, we stuck with one PL and increased the VUs count, and a similar pattern to the previous case was noticed, i.e., more unfoldings are more efficient than increasing PLs, and this resulted in the best performance yet (although we increased DFs strength).

We observed that although A-8 gives the best results, it requires around 9 parameters per image pixel, while A-6 needs just 7 parameters per pixel and loses $4 - 5\%$ in dissimilarity measures. Therefore, we will consider $n_\ell = 1$, $n_u = 7$ as an *optimal configuration* for the next runs.

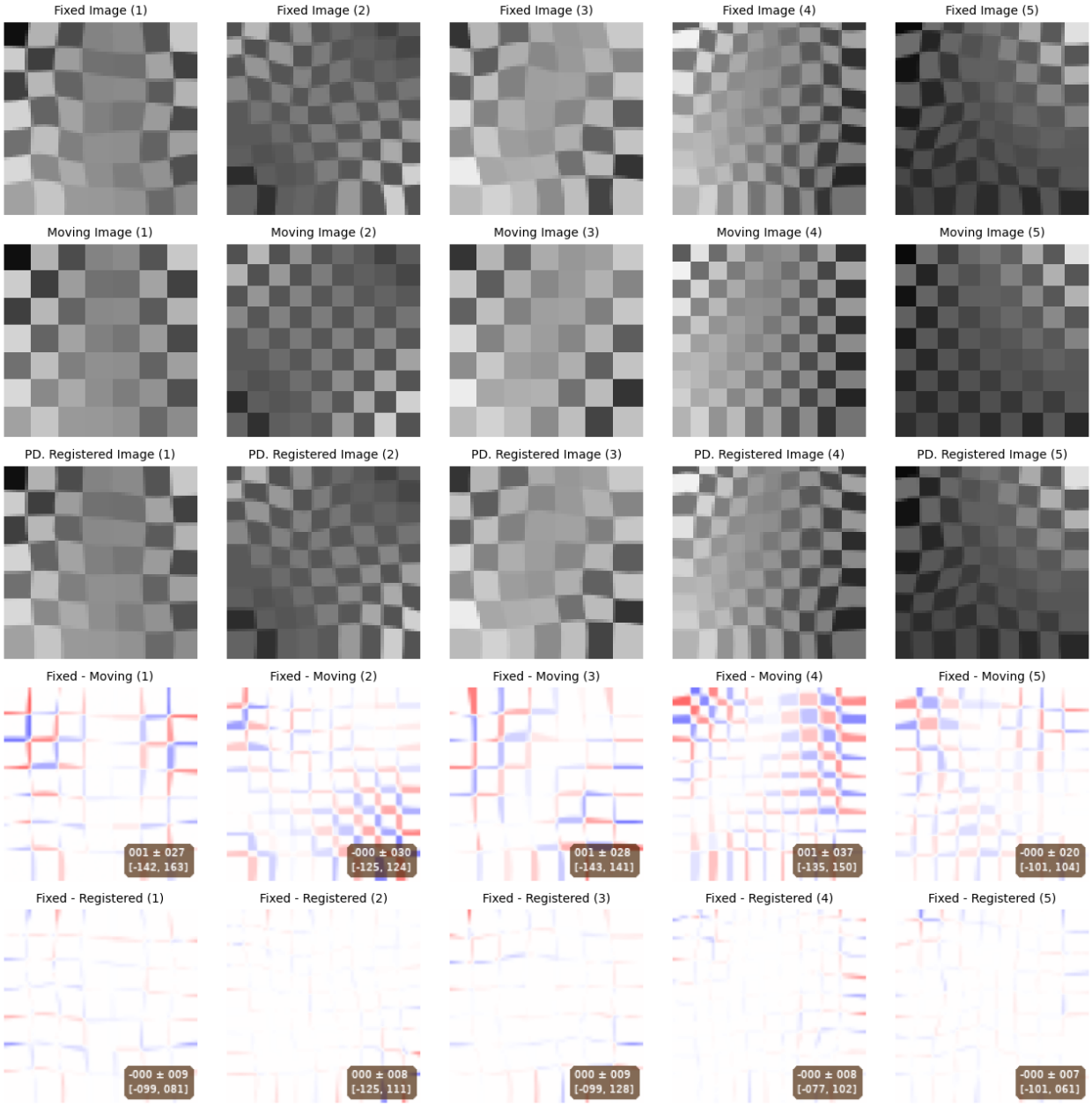

Figure 9: Results of the A-5t run (see Tbl. 1 for details) for 2D gradient chessboard synthetically generated images of 64×64 pixels in size[5]. The first row presents the fixed images after applying synthetic FFs and the swapping trick. The second row shows the moving images, which are the fixed images before the swapping trick. The third row presents the registered images. The fourth and fifth rows show the heat map of image intensity differences, where the blue color represents -255 and red represents 255. The numeric boxes show statistics about the image differences, making it easier to evaluate registration improvements. From top to bottom, and from left to right, they describe: mean value, standard deviation, minimum, and maximum. The visual representation of the applied FFs on these images can be found in Fig. 10.

[5]Note that since those images are originally of small size, some blurring effects and insufficiently sharp features may appear in the report.

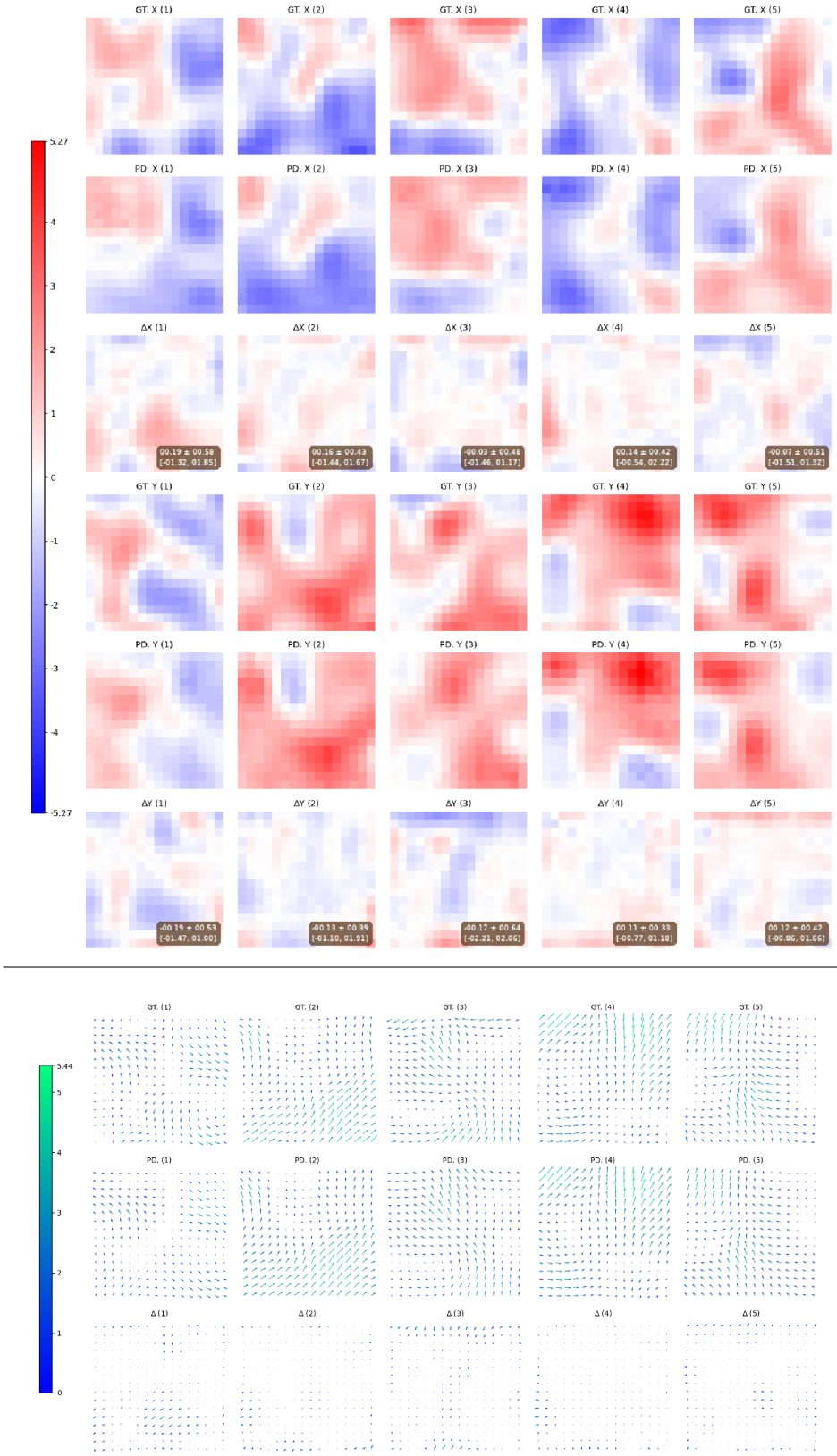

Figure 10: Applied and predicted DFs in the A-5t run (see Tbl. 1 for details) for the 5 images (columns) that presented in Fig. 9. It shows the ground truth (GT) and predicted (PD) displacement for the x (rows 1-3) and y (rows 4-6) components of the fields, as well as the difference (Δ) between the GT and PD (rows 7-9). The last three rows show the vector representation of these fields, as well as their differences (DF error after registration).

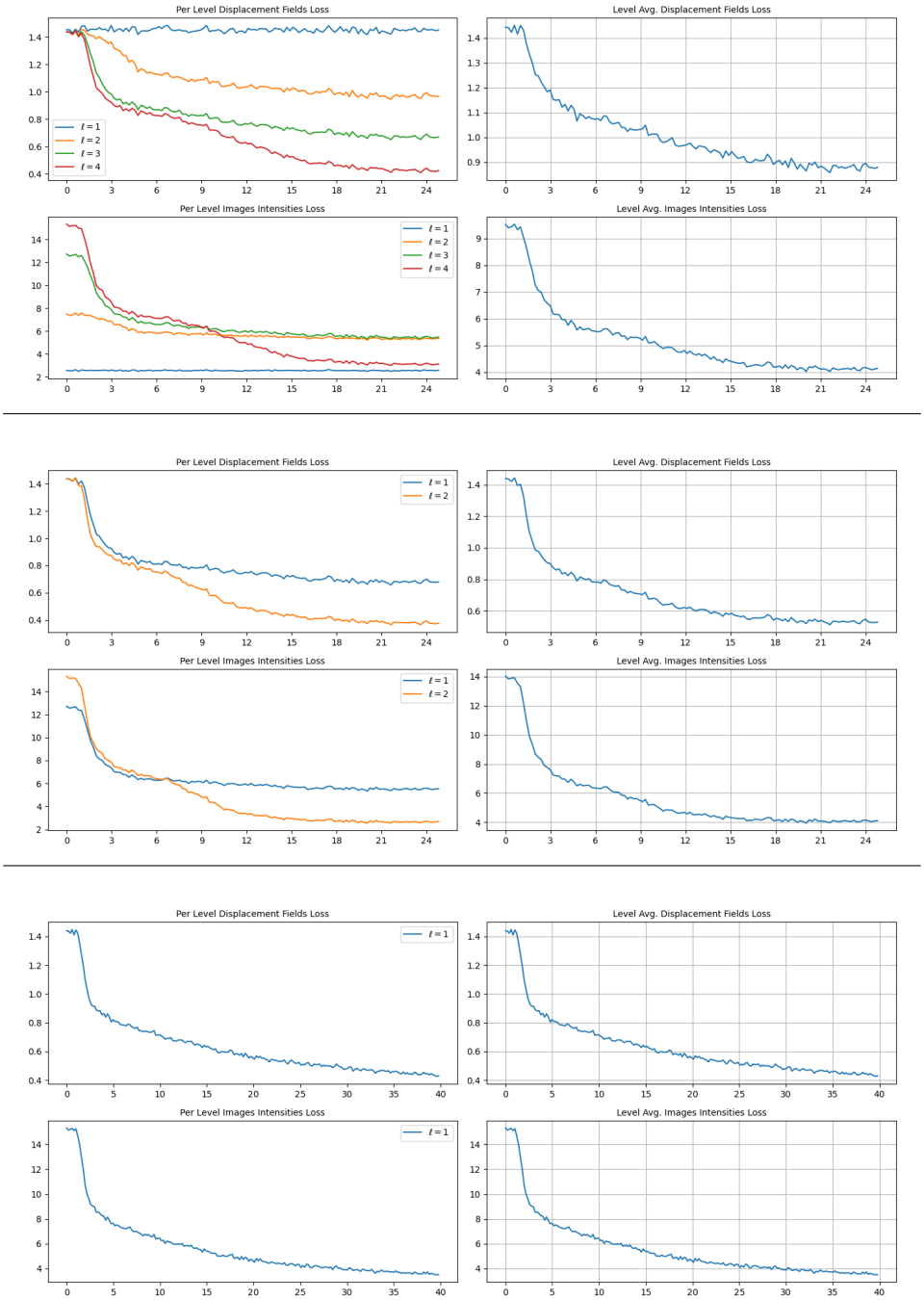

Figure 11: Plots of the intensity and displacement dissimilarity loss progression in various runs are reported for the last VU for each PL (left column) and the average of losses for all PLs (right column). Results are sampled every 10 batches, and the x-axis represents the epoch index. From top to bottom, these plots correspond to runs A-3, A-5, and A-6 (see Tbl. 1 for run details).

In the next Fig. 12, for predicted displacement fields, we show another pattern we noticed: VUs of the very first PL usually do not learn well, or in other words, adding a new PL enhances the learning output for the first couple of VUs of the very next PL.

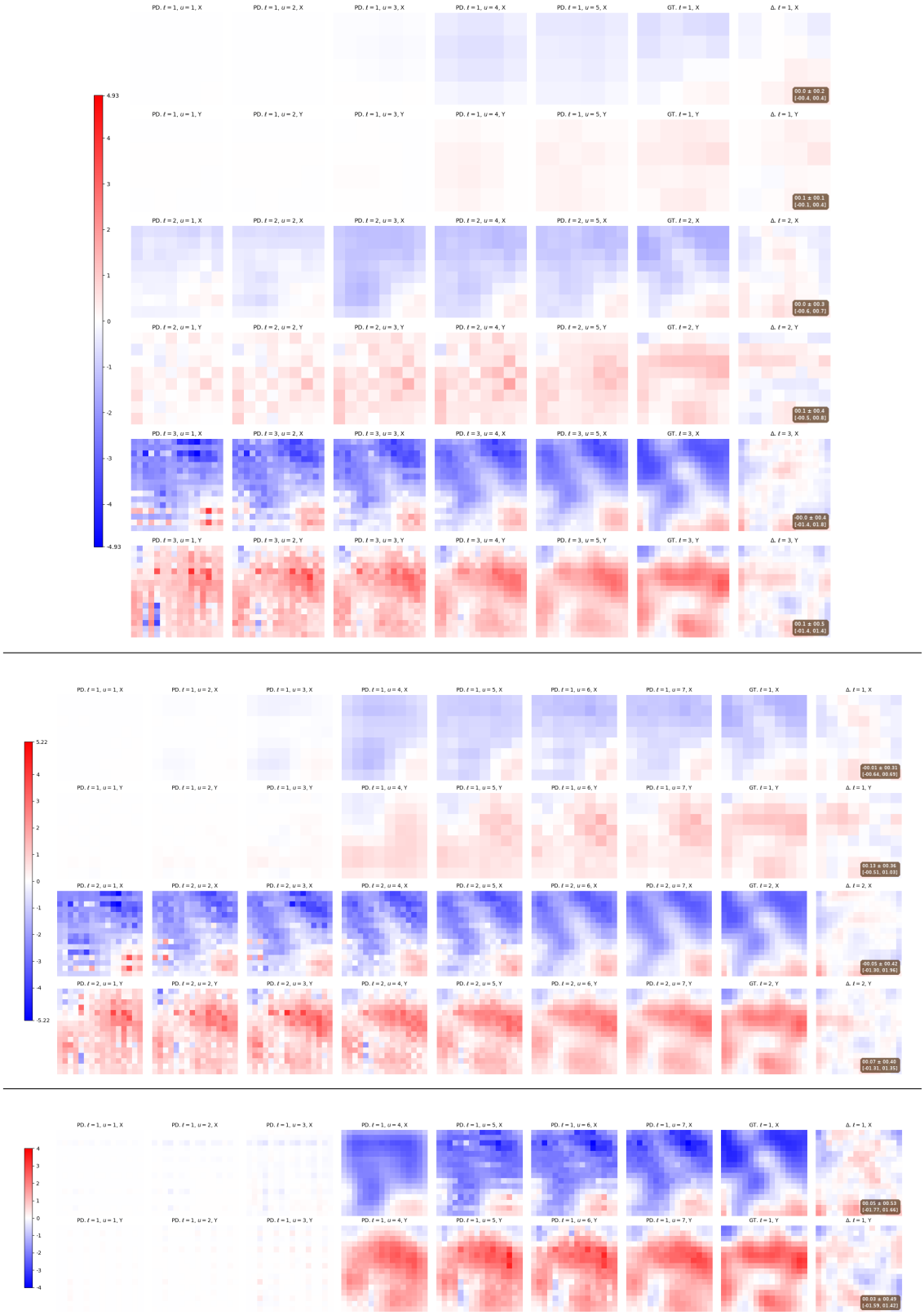

Figure 12: Predicted (PD) DFs for each VU (columns) of each PL (rows), for a sample chessboard pattern, for the (see Tbl. 1) runs A-1, A-5, and A-7 respectively, for top, middle, and bottom panels. The last two columns represent the VUs. Every two rows represent the x and y components per PL. The column before the last one represents the ground truth (GT) of the displacement fields, and the last column is the difference (error) between GT and PD.

Effect of Affine, Non-Rigid Deformations, and Feature Size: Having the above defined optimal configuration, we wanted to check the network's behavior in some "partial" cases, so we performed the runs in Tbl. 2 on chessboard images.

Table 2: Results of training the network on $10,000$ random 64×64 chessboard images, and similarly for testing, without using patches. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
B-1	64	[6, 10]	Both	10.12	1.92	30	100	1	7	24	28,735	1.92	0.23	85.4%	80.3%
B-1t	64	[6, 10]	Both	8.51	1.95	-	-	1	7	24	28,735	1.93	0.23	**85.4**%	**80.4**%
B-2	64	[6, 10]	NonRig	8.59	1.94	25	100	1	7	24	28,735	3.46	0.37	62.1%	54.4%
B-3	64	[3, 7]	Both	10.14	1.97	30	100	1	7	24	28,735	6.95	0.55	68.6%	59.8%
B-3t	64	[3, 7]	Both	8.57	1.95	-	-	1	7	24	28,735	6.91	0.54	68.8%	60.2%

In run B-1, we trained the network with affine transformations only, and in run B-2, we trained the network on non-rigid deformations only. Predictably, the model's performance on solely affine fields was better than on the others. Although we have over-parameterization in the affine run, the testing results show no signs of overfitting. This is mainly due to the highly structured VUs in our model.

The non-rigid run B-2 shows relatively weaker performance compared to run A-6 because it was trained for fewer epochs. Additionally, affine transformations may interfere with non-rigid transformations by simplifying the overall transformation, thereby reducing its complexity.

Finally, run B-3 aims to investigate our network's performance on features that may have a smaller or comparable size to the control point spacing (which is 4 pixels). This slightly degraded the network's performance, but it still managed to make relatively adequate predictions.

Larger Chessboard Image Sizes: Having the above results, we can now investigate the network's behavior on larger size chessboard images, as well as slightly stronger deformation fields, to see how the previous results will generalize on a convolutional network. This is reported in Tbl. 3, which contains runs on chessboard images of 128×128 size.

Table 3: Results of training and testing the network on $10,000$ random 128×128 chessboard images for training, and similarly for testing, without using patches. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
C-1	128	[6, 10]	Both	10.74	2.47	35	30	2	5	24	41,050	3.93	0.52	76.5%	69.9%
C-1t	128	[6, 10]	Both	10.83	2.45	-	-	2	5	24	41,050	3.90	0.50	76.7%	71.2%
C-2	128	[6, 10]	Both	10.50	2.44	30	55	1	7	24	28,735	2.79	0.38	84.7%	77.9%
C-2t	128	[6, 10]	Both	10.51	2.44	-	-	1	7	24	28,735	2.80	0.37	**84.8**%	**78.1**%

We started with a moderate setup of two PLs. Since we now have higher-resolution images, multiple PLs might produce better outcomes. Indeed, C-1t gave us relatively good results compared to those from the A and B series runs, although it utilizes approximately 2.5 parameters per image. However, when we tested the optimal configuration in C-2 with just one PL, it gave us even better results,

outperforming all previous runs. This is probably due to the higher resolution while maintaining the same feature size, which provides more information for the network to align images.

We did not perform more runs with those image sizes, since these two runs already indicate that the model is able to generalize to larger image sizes.

6.5.2 Patched Experiments

Having the previously obtained results as a baseline, we can now evaluate the effect of patched training on model performance:

Large Chessboard Images: To investigate if using patches of images for learning, while performing inference or testing on the whole image (as described in Sec. 5.6), we performed runs on 128×128 chessboard images again, but using patches this time, as reported in Tbl. 4:

Table 4: Results of training on $2,000$ chessboard images and testing on $10,000$ chessboard images of size 128×128, using patches for training. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
D-1	128/32	[6, 10]	Both	9.13	2.32	30	100/20	1	7	24	28,735	7.56	0.81	51.3%	48.1%
D-1t	128	[6, 10]	Both	9.88	2.32	30	-	1	7	24	28,735	6.21	0.73	68.3%	61.0%
D-2	128/64	[6, 10]	Both	8.87	2.04	25	100/20	1	7	24	28,735	2.91	0.34	79.6%	75.8%
D-2t	128	[6, 10]	Both	9.19	2.04	25	-	1	7	24	28,735	3.18	0.41	**82.7%**	**75.8%**

Here we performed two runs with a optimal configuration setup, where in the D-1 run we used smaller 32×32 patches, and in the D-2 run, patches of size 64×64 were used. At least in the D-2 run, the results are very similar to those of C-2t that we obtained with full-sized images in Tbl. 3, which indicates that using patching during learning to reduce resource usage leads to almost no loss in terms of network performance if the used patches are not much smaller than half of the original image size.

ANHIR Dataset: Having obtained previous results, we can now investigate network behavior on the more complex and realistic ANHIR dataset. We will use patches for training, since the images have a pretty high resolution for our hardware to handle. To verify our model on real tissue images, we performed the following runs on ANHIR 256×256 images with patches in Tbl. 5:

We started with the optimal configuration and a 64×64 patch size to see if we could achieve similar performance to the previous run of quarter-sized patches, D-1. However, the results were not satisfactory. Therefore, we reverted to half-sized patches.

Since more PLs introduce more parameters and we are dealing with high-resolution images, the E-2 run utilized additional PLs. This led to a significant improvement in performance, although it is still not at the level of the previous results.

In E-3, we used the optimal configuration with half-sized patches again and observed similar behavior to previous runs with quarter-sized patches. Although runs with chessboard images showed no benefit from introducing more PLs, the rich features in this dataset suggested that adding more

Table 5: Results of training and testing the network on all ANHIR images of size 256×256, with using patches for training. See Sec. 6.4.1 for more details on table columns. The " ? " symbol stands for the undefined feature sizes in this dataset.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
E-1	256/64	?	Both	8.03	2.04	30	100/20	1	7	24	28,735	9.48	0.68	51.0%	52.1%
E-1t	256	?	Both	13.69	2.05	-	-	1	7	24	28,735	10.7	1.75	42.5%	32.4%
E-2	256/128	?	Both	15.48	4.01	30	60/20	2	5	24	41,050	8.85	1.32	52.1%	53.5%
E-2t	256	?	Both	18.28	4.02	-	-	-	-	-	41,050	8.10	1.88	52.6%	48.8%
E-3	256/128	?	Both	13.69	2.45	30	60/15	1	7	24	28,735	9.73	1.44	53.9%	57.6%
E-3t	256	?	Both	13.69	2.05	-	-	1	7	24	28,735	10.7	1.75	48.1%	44.2%
E-4	256/128	?	Both	10.57	2.45	30	40/15	2	7	24	57,470	4.42	0.63	71.1%	63.6%
E-4t	256	?	Both	13.69	2.46	-	-	2	7	24	57,470	5.32	1.18	64.5%	54.3%
E-5	256/128	?	Both	12.02	2.37	10	35/15	3	7	32	114,933	3.04	0.62	74.1%	63.0%
E-5t	256	?	Both	13.63	2.38	-	-	3	7	32	114,933	3.35	0.99	**71.9%**	**60.4%**

PLs might be advantageous. In the E-4 run, we added one more PL over the optimal configuration, which indeed led to better performance during testing. This encouraged us to further increase the PL count in the E-5 run. We also decided to increase the filter count to handle the complexity of the images, resulting in the best outcomes we have achieved on the ANHIR dataset to date. The Figs. 13, 14 provide visual representations of the results from the last run, where we can see that DIRV-Net performs relatively well on this dataset as well.

Comparing the improvements in the dissimilarity measure across various runs, we observe that learning on the ANHIR dataset requires more parameters than on our synthetic chessboard images to demonstrate similar performance. Additionally, PLs play an important role in improving network performance for this dataset.

Due to the high resolution, we were unable to increase the PL or VU count further. However, we consider the obtained results sufficient to demonstrate the network's ability to learn and handle more complex and higher-resolution medical images.

6.5.3 Miscellaneous Experiments

We conducted several additional miscellaneous experiments to better understand network behavior:

We were able to fit our full-sized images from ANHIR into DIRV-Net with the available hardware without using patches for learning. However this was achievable by using the optimal configuration at it most, i.e., with just one PL and 7 VUs, although, as shown in Tbl. 5, having more PLs is essential for good performance with this dataset. This run can be compared with the E-3t run in that table to understand the performance difference between the patched and patchless approaches. We report only the testing dissimilarity reduction: $\Delta\bar{L}_{\text{int.}} \approx \Delta\bar{L}_{\text{dis.}} \approx 44\%$, which means the patched run is highly comparable to the patchless run in this dataset also. However, both are still far from the best results in the E-5t run.

Although the structure and overall size of features in ANHIR and chessboard images vastly differ, it will be interesting to see how a model that trained on ANHIR images performs on chessboard images. Therefore, we fed 128×128 chessboard images to the model trained in the E-3 run (since we know that having more than one PL is not important for the chessboard dataset). The testing results, as

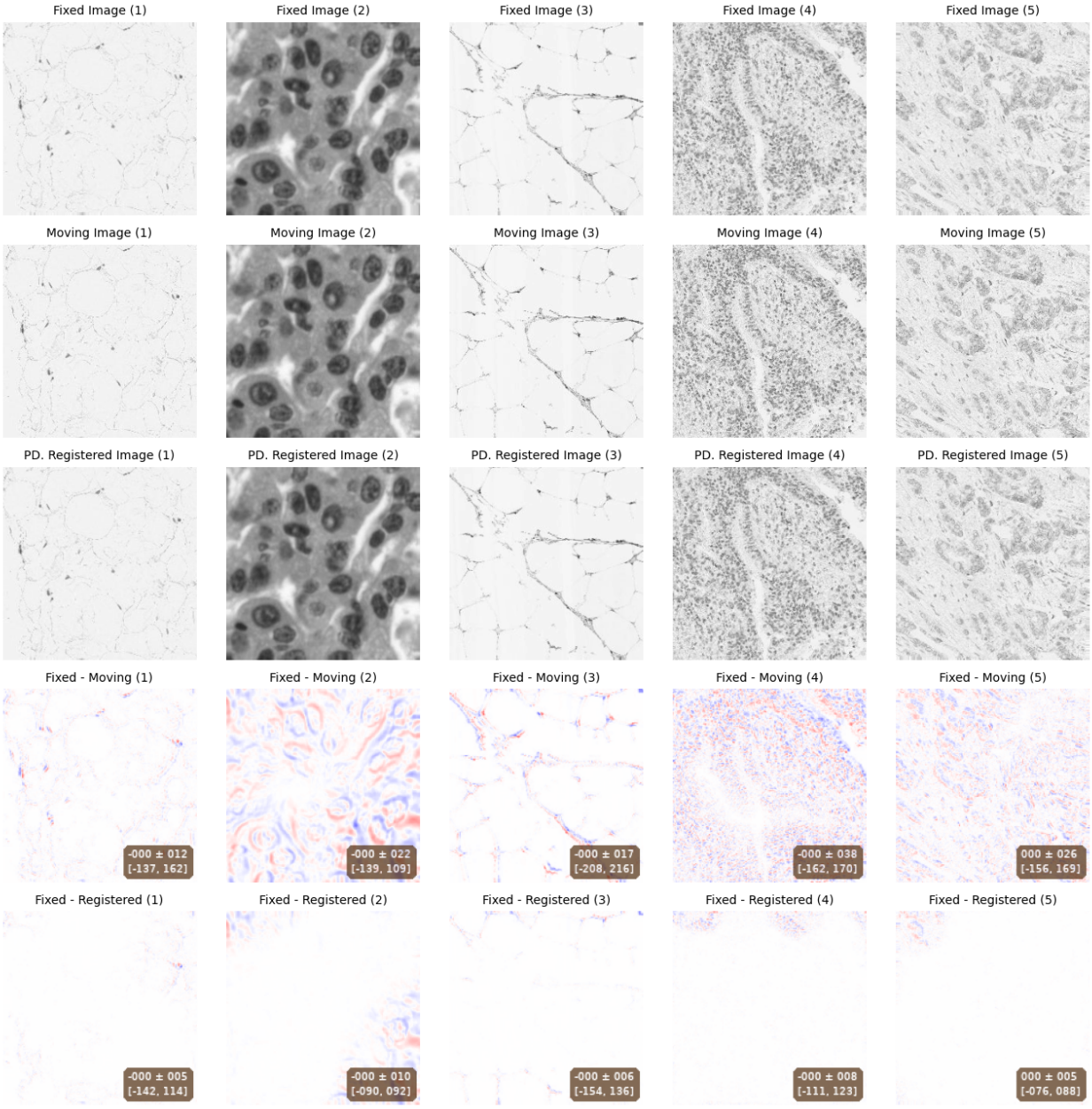

Figure 13: Results of the E-5t run from Tbl. 5, showing the fixed images, moving images and the difference heat maps of ANHIR images. The fields that been used to generate those deformations are depicted in Fig. 14. See Fig. 9 for a more detailed description on images content.

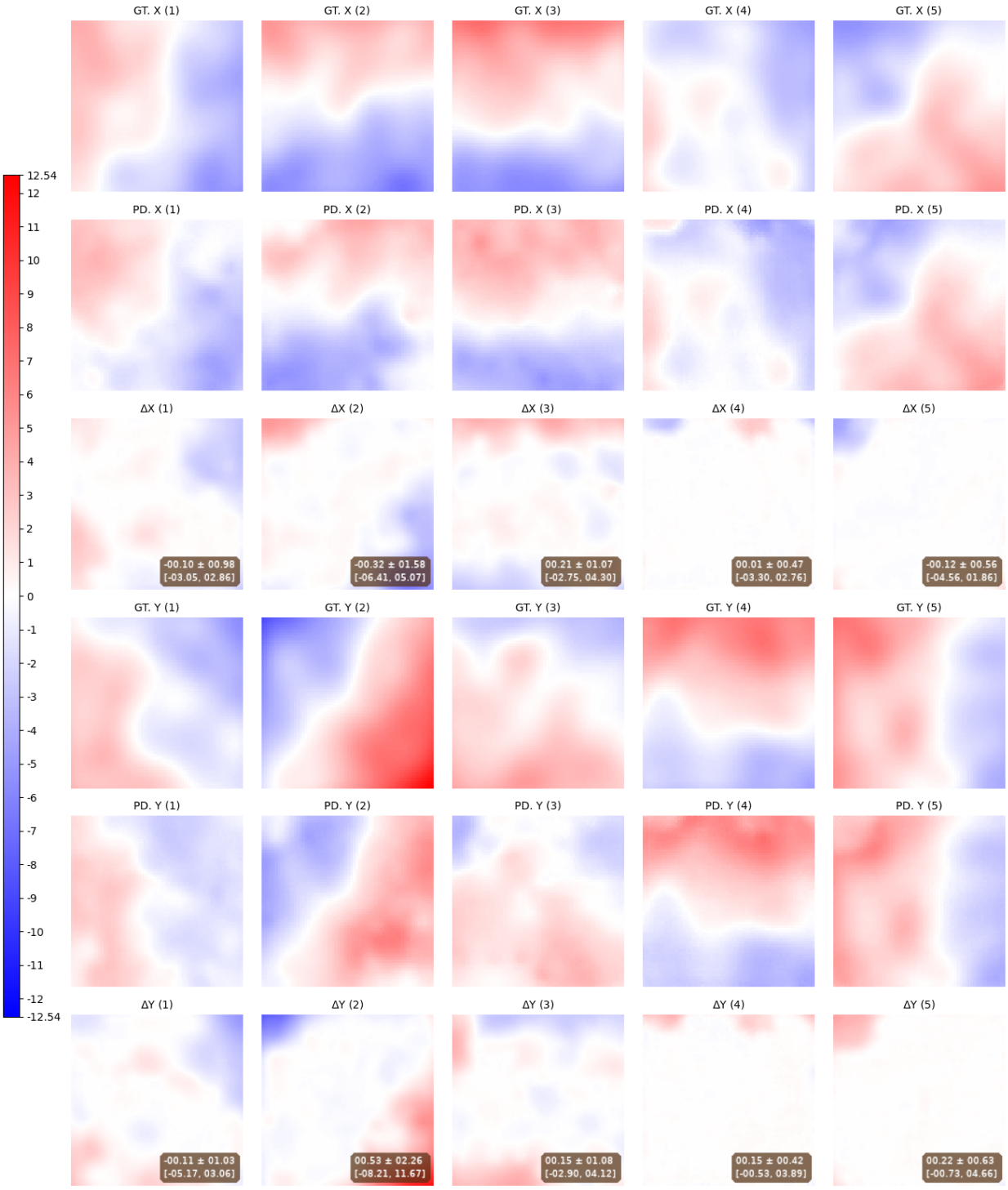

Figure 14: Results of the E-5t run from Tbl. 5, showing the ground truth and predicted DFs, as well as there difference. The images that those fields been applied to are in Fig. 13. See Fig. 10 for a more detailed description on images content.

an reduction in dissimilarities, are: $\Delta\bar{L}_{\text{int.}} = 55.6\%$ and $\Delta\bar{L}_{\text{dis.}} = 53.3\%$, which is even better than the model's results on the original dataset it was trained on (48.1%, 44.2% correspondingly). This shows the model's ability to generalize well to images that differ in nature from the dataset it was trained on.

Finally, we tested the performance of the recurrent refinement (discussed in Sec. 5.8) on E-5t output from Tbl. 5, an example of the results are represented in Fig. 15. We found that a per image recurrent refinement, decreases the average intensity dissimilarity by 33.3% (from $3.35 \rightarrow 2.24$ or for $\Delta\bar{L}_{\text{int.}}$: $71.9 \rightarrow 81.2\%$) while increasing the inference time about 20 times. The per-batch refinement (with batch size of 20 images), the average intensity dissimilarity reduced by 26.3% (from $3.35 \rightarrow 2.47$, or for $\Delta\bar{L}_{\text{int.}}$: $71.9 \rightarrow 79.3\%$), while inference time increased only 4 times. In both cases, refinement took in average $2.5 - 3.8$ iterations. This shows that recurrent refinement can extract even better results from the model, with some extra time spent on inference.

We would like to mention that in all of the runs we performed, our model reduces the standard deviation of the average intensity displacement $\bar{L}_{\text{int.}}$ by approximately 2-3 times compared to the initial standard deviation of $\bar{L}_{\text{int.}}$. This means that DIRV-Net reduces the initial diversity of $\bar{L}_{\text{int.}}$ across the dataset, bringing them to a more equal footing, which further illustrates the consistency of the results (we did not report the standard deviations to reduce clutter).

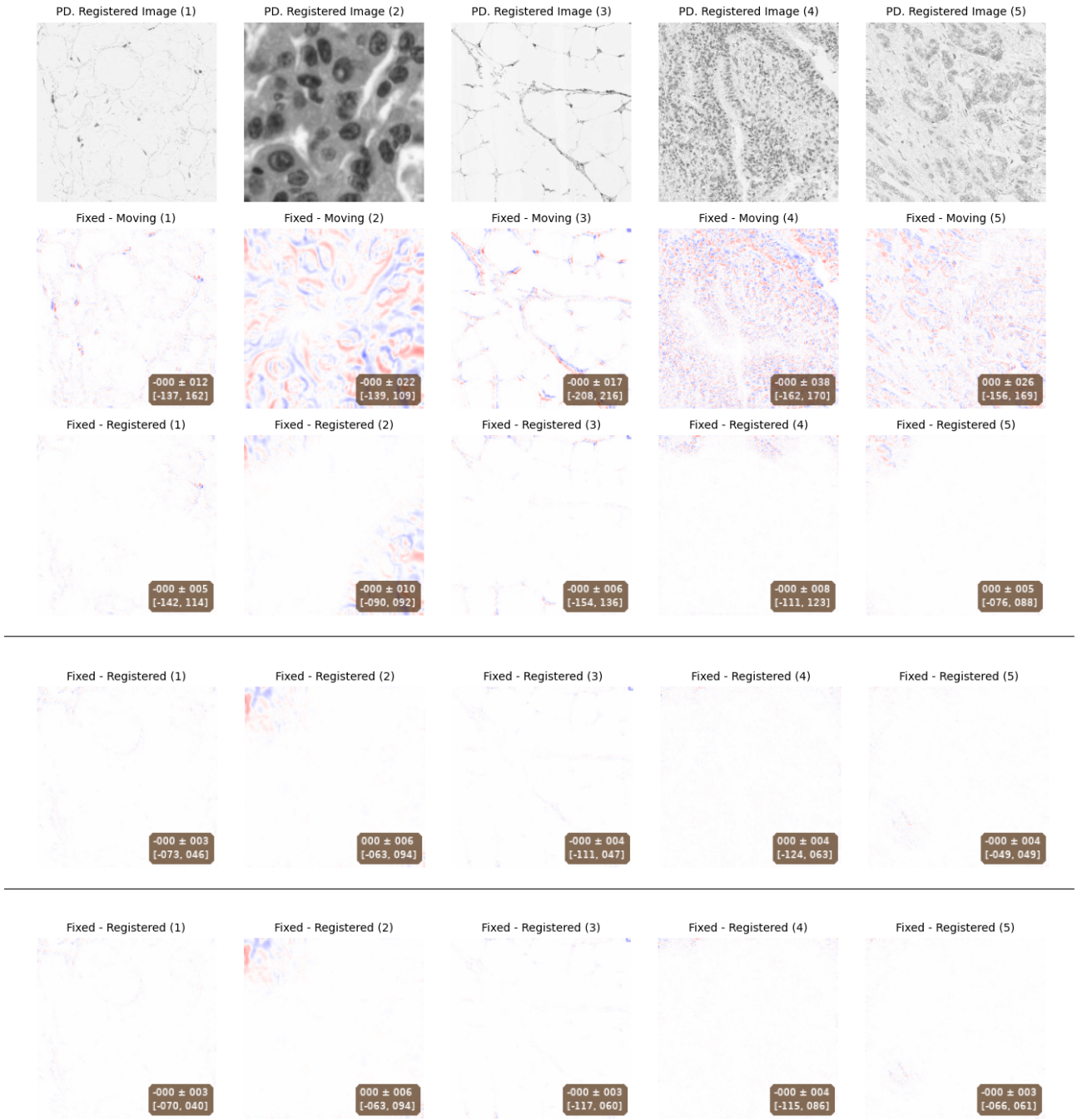

Figure 15: Comparison of testing results of the ANHIR tissues dataset from the E-5t run from Tbl. 5. The first 3 rows represent the normal run output. The 4th row is the output of per-batch (of size 20) recurrent inference. Finally, the last row is the result of a per-image recurrent refinement. These images can be seen in conjunction with Fig. 13 and Fig. 14

6.6 3D Experiments

Performing runs on 3D images requires much more resources than in the case of 2D images, and this is not only due to the larger number of pixels that should be processed but also due to the higher

number of parameters that the model needs to maintain for 3D kernels and other parameters.

Hence, this affected the combination of hyperparameters, batch size, patch count, and even dataset sizes that we were able to test. Moreover, these experiments will be run solely on chessboard images.

6.6.1 Patch-Free Experiments

We start by evaluating the network on relatively small chessboard images. These runs consumed a lot of resources, so we limited ourselves to 15 epochs here, which was enough to achieve a satisfactory level of convergence. Results are reported in Tbl. 6.

Table 6: Results of training on $2,000$ chessboard images and testing on $4,000$ chessboard images of size $64 \times 64 \times 64$, without using patches. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
F-1	64	[6, 10]	Both	9.10	2.69	15	4	1	5	24	97,685	5.41	0.52	68.8%	66.6%
F-1t	64	[6, 10]	Both	9.10	2.69	-	-	1	5	24	97,685	6.12	0.52	68.8%	66.8%
F-2	64	[6, 10]	Both	11.47	2.64	15	3	2	5	24	195,370	5.08	0.63	64.6%	58.1%
F-2t	64	[6, 10]	Both	10.47	2.64	-	-	2	5	24	195,370	5.19	0.64	64.1%	57.6%
F-3	64	[6, 10]	Both	11.47	2.64	15	3	1	7	24	136,759	4.31	0.45	74.5%	70.5%
F-3t	64	[6, 10]	Both	10.47	2.64	-	-	1	7	24	136,759	4.30	0.44	**74.6%**	**70.7%**

Similar to the runs in Tbl. 3, adding more PLs does not improve performance for chessboard images, as can be seen in run F-2 compared to F-1. Additionally, even for 3D images, the optimal configurations still provide relatively good results compared to 2D runs performed in Tbl. 1. Due to hardware limitations, we were not able to fit larger configurations into GPU memory.

6.6.2 Patched Experiments

As a final set of experiments, we tested how well the patching approach generalizes in the 3D case, and the results are reported in Tbl. 7:

Table 7: Results of training on 500 chessboard images and testing on $1,000$ chessboard images in the G-1 run, and 300 for training and 500 for testing in the G-2 run. Both cases use images of size $128 \times 128 \times 128$, with patches for training. See Sec. 6.4.1 for more details on table columns.

Run ID	Dataset		Transformations			Run Parm.		DIRV-Net Parameters				Errors			
	Img./ Pat.	Feat. size	Type	max $\|\|\mathcal{D}\|\|$	$\overline{\|\|\mathcal{D}\|\|}$	Ep. cnt.	Bat./ Pat.	n_ℓ	n_u	n_e, $\bar{n}_e$	$\sum$	$\bar{L}_{\text{int.}}$	$\bar{L}_{\text{dis.}}$	$\Delta\bar{L}_{\text{int.}}$	$\Delta\bar{L}_{\text{dis.}}$
G-1	128/64	[6, 10]	Both	9.92	2.70	15	4/10	1	5	24	97,685	4.26	0.47	69.0%	69.7%
G-1t	128	[6, 10]	Both	11.98	3.44	-	-	1	5	24	97,685	6.12	0.78	69.2%	60.9%
G-2	128/64	[6, 10]	Both	11.65	2.65	12	3/8	1	7	24	136,759	3.62	0.45	**72.9%**	**70.4%**
G-2t	128	[6, 10]	Both	12.27	3.42	-	-	1	7	24	136,759	6.10	0.82	69.2%	60.1%

At first glance, it appears that adding more VUs in the second run, G-2, did not improve performance. However, to execute this run, we had to significantly reduce the training dataset size and patch size. This resulted in similar performance for both runs with 5 and 7 VUs. Consequently, the performance is just apparently weaker than what we obtained previously in 2D runs with similar model configurations. Thus, the presented 3D results can be viewed as a proof of concept and

demonstrate the model's ability to generalize to higher-dimensional images even with patches. Some of the depicted results of the G-2 run are presented in Figs. 16, 17.

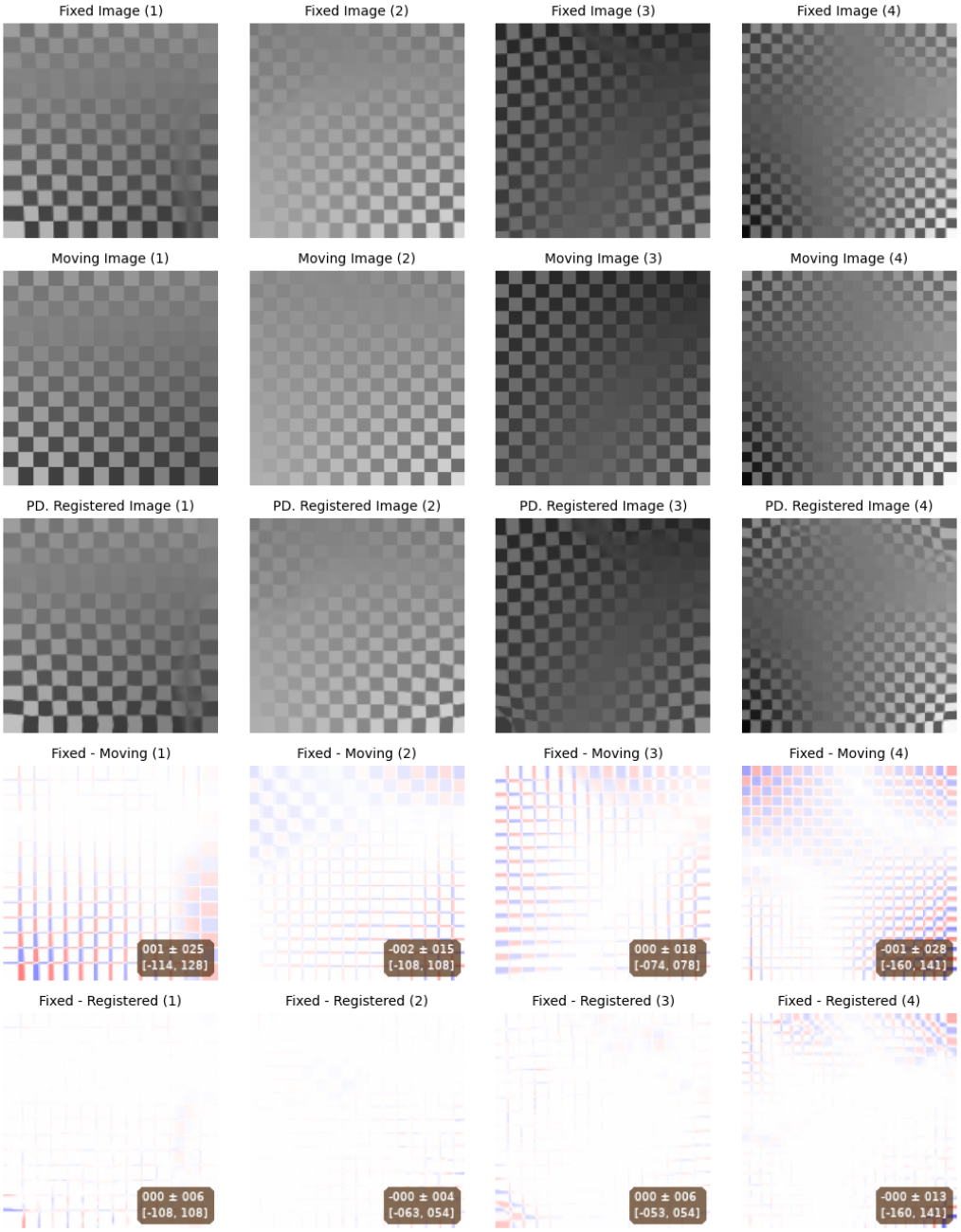

Figure 16: Fixed, moved, registered images and their difference heat maps, for four 3D checkerboards examples (columns). These are results of the G-2t run (see Tbl. 7) with patching during training on $128 \times 128 \times 128$ chessboard images, where these 2D images are slices of the 3D images in the middle of the Z-axis. The fields used to generate these deformations are depicted in Fig. 17. See Fig. 9 for a more detailed description of the images' content.

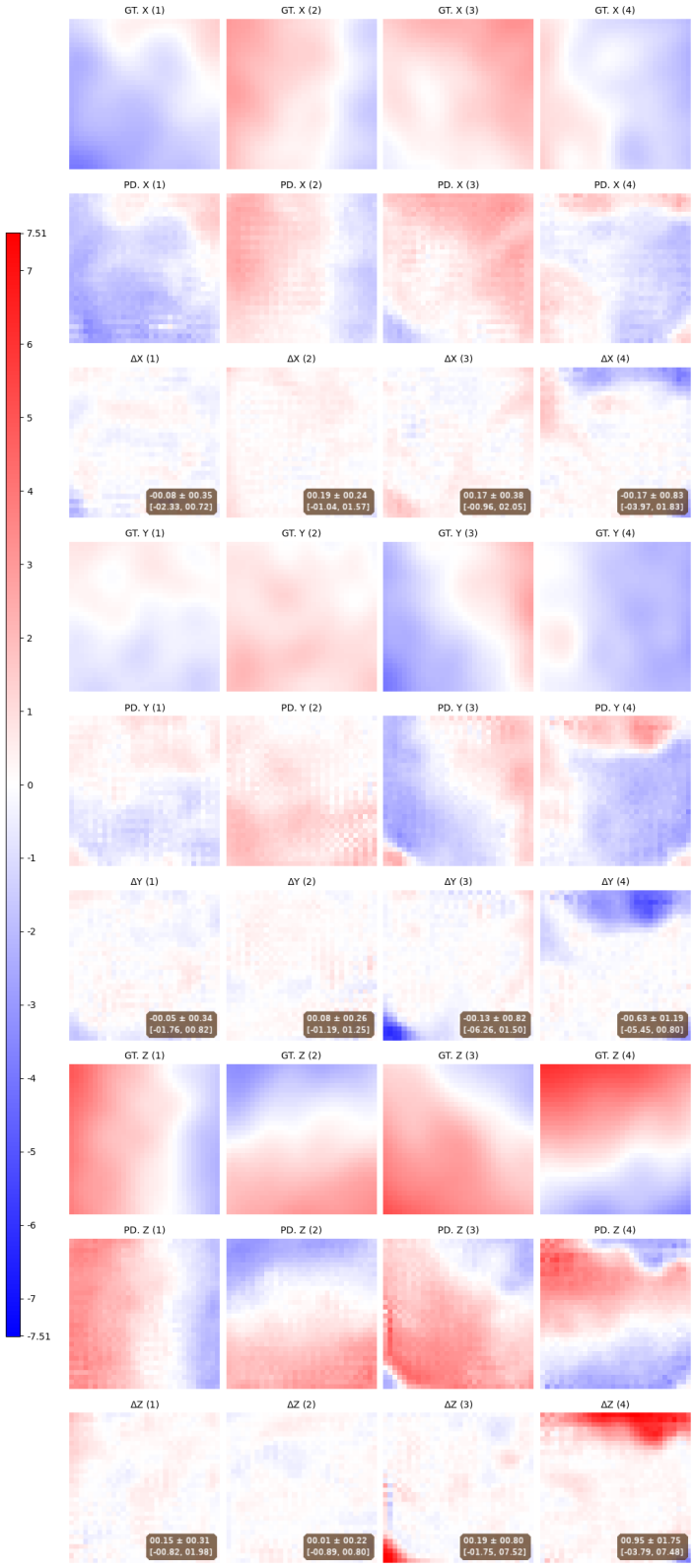

Figure 17: The ground truth and predicted DFs, as well as their difference heat maps, for every vector component; for the four examples in Fig. 16. These are results of the G-2t run (see Tbl. 7) with patching during training on $128 \times 128 \times 128$ chessboard images. See Fig. 10 for a more detailed description of the images' content.

7 Discussion

First, we would like to point out the predictable, by design, limitations of our implementation.

7.1 Inherited Errors

The setup and implementation methods we presented introduce multiple sources of errors that we should account for when results are evaluated.

Besides the issues that may accompany inverse field calculations, which we avoided by the swapping trick (see Sec. 5.5), we can distinguish the following error types:

- **Occlusion Errors**: These errors occur because deformation fields can cause some pixels to go out of image borders, leading to information loss that cannot be accurately reconstructed during image registration. Interpolation can fill those pixels with artificial values, but this does not accurately restore the lost information.
- **Warping Errors**: Bi- and tri-linear interpolation, which are usually used for image warping, assume smooth fields everywhere, and that was the case for our synthetic deformation fields. However, such an assumption leads to a significant loss of detail at feature borders when images are warped (e.g., see Fig. 9, where the borders between chessboard cells appear smoothed out in the fixed images). In real medical images, some tissue features may move as a single entity during deformations without losing their border sharpness. This makes comparing the fixed images with the warping results challenging, even when realistic transformation fields are used, due to the inherent nature of interpolation in the warping process.
- **Displacement Field Errors**: The reduction of the needed calculations by introducing and predicting the DFs instead of FFs (see Sec. 3.5.1) introduces errors due to the necessary interpolation process. Our experimental verification of the dissimilarity for 2D DFs (see Fig. 18) showed that this dissimilarity can reach up to 55% of the field's maximum component magnitude (on average, it is about 12%), which can translate to up to 95% in pixel intensity dissimilarity after image warping (with an average of about 15%).
- **Interpolation Artifacts**: Our use of first-degree B-Splines for DF interpolation introduces some small artifacts. This can also be seen in Fig. 18, where the difference between the ground truth and the interpolated DF produces noticeable block-like patterns. However, since we are predicting the DFs and not FFs, such errors only affect the warping process and will be mostly smoothed out.
- **Up/Down Sampling Intensity Degradation**: Since we are using smoothing with linear interpolation for image up/down sampling (for example, when we are creating resolution pyramids), this leads to a slight intensity "washout" effect. This is one of the reasons why the inner pyramid levels usually slightly underestimate the displacement field strength (see Fig. 12).

By a rough estimation of those errors, we found that they contribute to $\approx 15-20\%$ in the average intensity dissimilarity. This means that the reduction in the average intensity dissimilarity we achieved in our runs of $\approx 70-80\%$ is the theoretically maximum possible level that can be achieved for image registration with the current implementation approach.

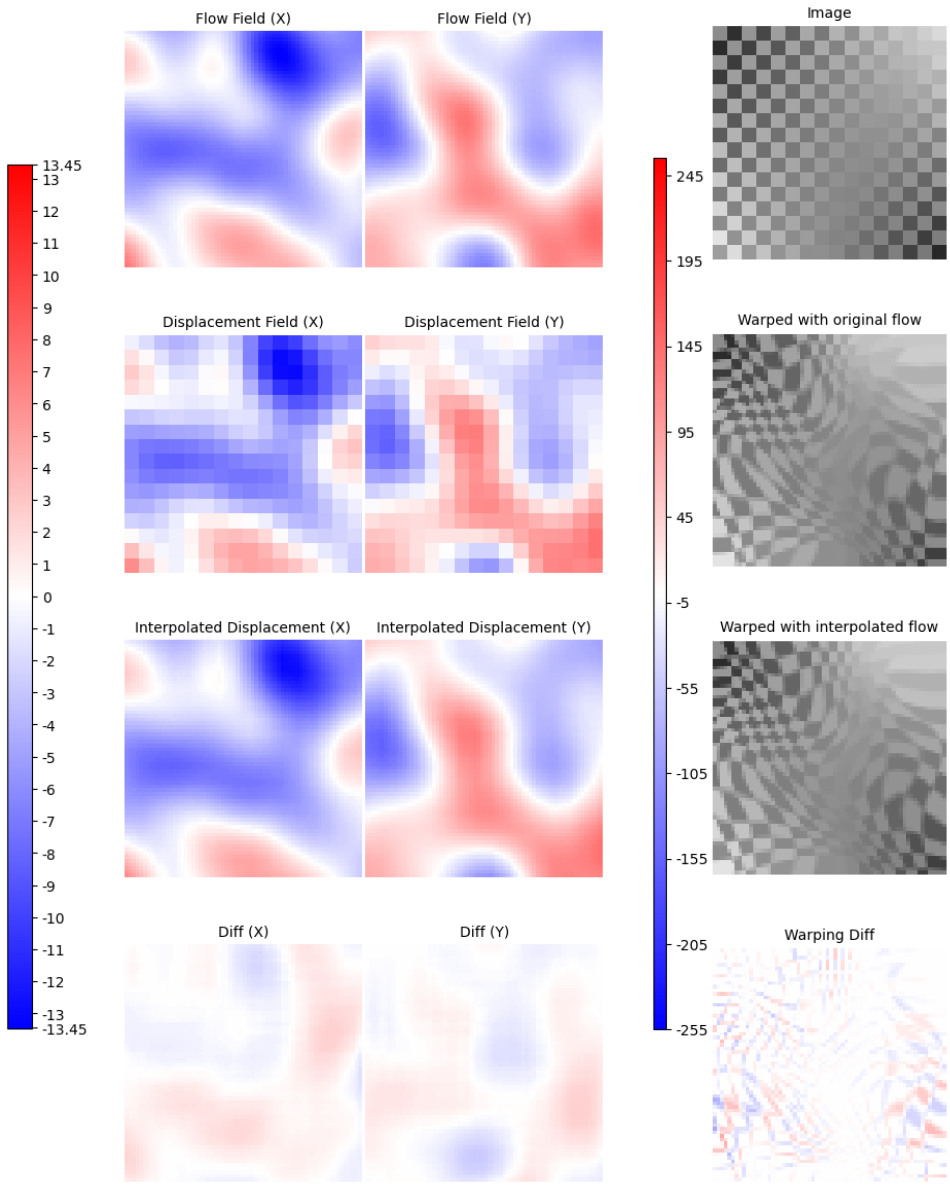

Figure 18: An example of how interpolating a synthetic 2D random DF to an FF with first-degree B-Splines will lead to information loss and errors in the resultant field, and consequently to errors in the image warping.

7.2 Discussion of 2D Experiments

The introduced network architecture is notably stable and converges relatively quickly in almost all the cases we tested here (see Fig. 11). As found earlier, having more PLs leads to faster convergence, although they may not be beneficial for all datasets. On the other hand, increasing the number of VUs generally leads to a notable improvement in model performance, although the first few VUs have difficulties learning unless an additional PL is introduced before them (see Fig. 12). Moreover, signs of overfitting were minimal in most of the configurations we tried. We also observed that, in general, introducing many kernels significantly increases the total model parameters, while the performance gain does not usually justify this increase.

We also observed a high correlation between the intensity and displacement losses (see Fig. 11), which is why we used only the displacement-based dissimilarity surrogate function in our experiments. However, our further experiments showed that this occurs mainly due to the small number and size of semi-empty regions (i.e., featureless areas) in our dataset images. Some results on low-resolution non-medical images with large empty regions can be found in [9].

Having relatively small feature sizes (compared to the spacing of control points in displacement fields) does not significantly affect model performance, and the network is still able to produce well-predicted DFs. Also, the network showed relatively good results in predicting strong FFs, but it still under-predicts the strongest ones (which can be partially resolved by recurrent refinement). From the various figures we provided, it is also easy to notice that in general, the network predicts DFs better in the regions close to the image center.

We also noticed strange behavior in some configurations of our network, which we call *leakage anomalies*, where the network may insist on predicting the fields in some small regions incorrectly, regardless of how long we train it. An example of such behavior is presented in Fig. 19. In the predicted fields (PD) in the second column (bottom right corner) and the fourth column (top border), some small regions (surrounded by green ellipses for clarity) contain highly anomalous and incorrect predictions of the vector fields for both the x and y vector components when compared to neighboring regions. Upon closer inspection, one can notice that the anomalies in the predicted x components are very similar to those in the y components but in the opposite direction (i.e., they appear to be an inversion of each other). This phenomenon seems to appear as if some information "leaks" from one component of the vector field to the other component. We have not found a satisfactory explanation for this behavior; however, it occurs in rare cases.

We also showed that recurrent refinement (see Sec. 5.8) can provide a considerable boost in overall performance and reduce the average errors to levels comparable to the inherent errors discussed in the previous section, which eventually makes the registered images almost indistinguishable from the ground truth fixed images.

In summary, the network demonstrated its ability to learn and accurately predict various deformations in diverse datasets with different feature structures, complexities, image resolutions, and sizes. However, this may become more challenging when there are insufficient hardware resources for the learning process. By using patches, we can partially reduce the required resources, by introducing only a relatively small loss in model performance, which can be compensated for by the introduced recurrent refinement (some additional suggestions about how it is possible to overcome resource limitations will be introduced in Sec. 9).

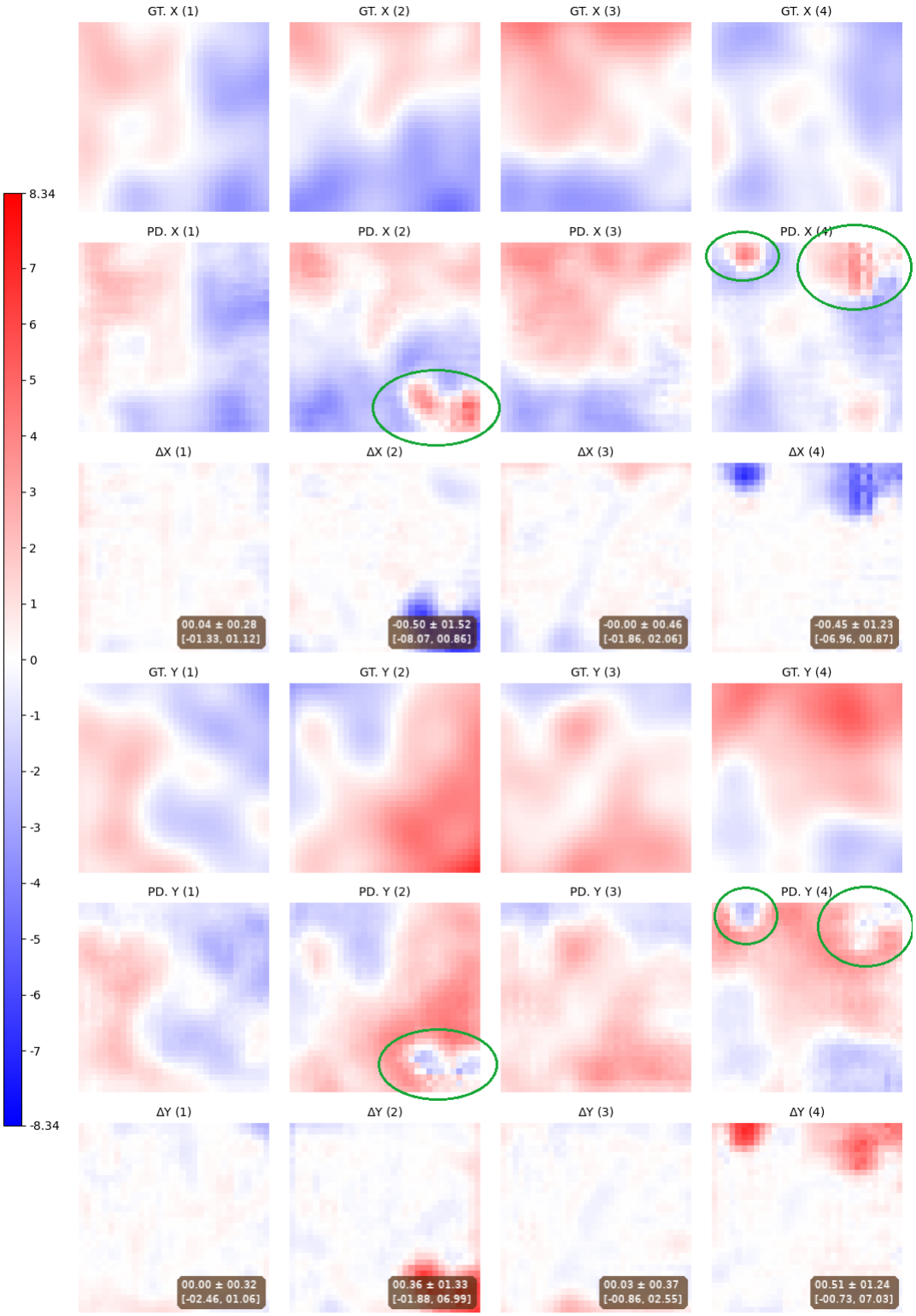

Figure 19: An example of the leakage anomalies (surrounded by green ellipses) in the predicted displacement fields (from the D-2t run in Tbl. 4). See Fig. 10 for a more detailed description of the images' content.

7.3 Discussion of 3D Experiments

Unfortunately, due to the small number of experiments we were able to run on 3D images because of limited hardware resources, it is difficult to draw reliable conclusions for such images.

However, despite the need to significantly reduce the batch size and the number of epochs (to bring down the duration of training), it seems that the model exhibits very similar behavior when dealing with 3D images compared to our observations in the performed 2D runs. Specifically, the effects of the number of unfoldings, the number of PLs, and how patches can decrease the required GPU memory resources by roughly half while still maintaining overall similar performance as when no patching is used.

Hence, more extensive experiments on more powerful hardware are needed to provide a more realistic evaluation of model performance in the case of 3D image registration. (Also, our suggestions in Sec. 9 could probably be useful to overcome this.)

8 Conclusions

In this work, we built the theoretical machinery and provided the motivation for using convolutional variational networks in image registration. As a result, we presented the DIRV-Net model, which can be trained in either a supervised or unsupervised manner. By analyzing the ground truth and deformed images, the network learns how to predict inverse deformation fields, which, when warped with the deformed image, can generate the original ground truth image before deformation.

Multiple variational units and image pyramids can learn to detect image features of various scales and predict the deformation fields iteratively. In this process, the output of each step becomes the input to the next step, which eventually allows the network to refine its predictions and resolve the ill-posed image registration problem.

By using gradient descent and the Adam optimizer, along with deformation field dissimilarity as a loss function (i.e., the deformation fields were given), we trained and tested the model on two- and three-dimensional images. Two different datasets were used: one synthetically generated to facilitate the visual inspection of the results, and another consisting of realistic, feature-rich medical tissue images. By focusing on the supervised learning approach, we synthetically generated the deformation fields, which consisted of both affine and non-rigid random fields.

We also introduced multiple techniques to overcome some hardware resource limitations and improve overall results, such as using image patches for training and recurrent refinement for better predictions.

The model showed resilience and generalizability, with the ability to learn and predict smooth and relatively complicated deformation fields from complex and highly detailed medical images, as well as from simpler synthetic images. Our model also showed comparable performance for both two- and three-dimensional images and good inference speed with moderate hardware resource needs.

9 Future Work

There are multiple areas for possible improvement in the introduced model. Here, we will limit our discussion to possible implementation improvements without suggesting fundamental changes to DIRV-Net:

- Using different numbers of kernels for regularization and data terms may reduce the needed resources while maintaining the same model performance. This may be beneficial because data and regularization terms are generally of different complexity.
- Using improved and more complex surrogate loss functions, for example, we can define the loss function as the weighted sum of eqs. (26, 27). Or, as discussed in Sec. 5.8, it seems that using the standard deviation of the dissimilarity, along with the average value of this dissimilarity, is a good indicator of well-learned deformation fields.

  It is also possible to train the model in different stages with different loss functions to utilize *weakly supervised* learning. Moreover, instead of using loss functions that involve only the last VU loss in each PL (as we did in eqs. (26, 27)), a more refined approach might be to include the loss at every VU output, e.g., by exponentially weighting the losses throughout all VUs, as suggested in [7]. This may also help the under-learned first few VUs from the initial PL to learn more effectively.
- Since the fields we want to predict aim to transform discrete image pixels, the fields that are weaker than being able to move a pixel for a distance longer than one pixel during deformation become irrelevant. This means that the high calculation accuracy of the model parameters is not a necessity, and hence one can use coarser model parameter quantization (by using, say, 16-bit floats instead of the default 32-bit floats), which should lead to a halving of the needed GPU memory resources for DIRV-Net. Due to unknown technical reasons[6], our efforts to quantize DIRV-Net with the used ML framework (TensorFlow) were not successful, but it can be done in theory.
- Implementing higher quality two- and three-dimensional up/down-sampling approaches to mitigate some of the inherited errors discussed in Sec. 7.1 may also enable a direct comparison of this model with the results of other state-of-the-art image registration methods.
- When we used patching, we were applying the deformation fields to the image patch itself. However, it could be interesting to see network performance if we apply the deformation fields to the whole original image first, and only after that, take a patch from the deformed image as well as from the deformation fields at the same coordinates.
- Since introducing more PLs is quite expensive in terms of the additional parameters (about a twofold increase from a single level) and the high resources needed for training such models, and because the ground truth DFs for each PL are known in supervised learning regime, creating an "ensemble" of DIRV-Nets, where each consists of a single dedicated PL, will require much fewer resources than training a model consisting of all PLs together and will add significant value. One may also train the combined model of all single PL models for some more epochs afterward; this may increase the model's overall performance even further.

[6] Our efforts for reducing numerical quantization caused strange GPU memory corruption errors. For example, TensorFlow was reporting completely wrong tensor rank/dimensionality, although the same code, but with float32 tensors, worked flawlessly and reported the correct rank/dimensionality. We were unable to debug this thoroughly in the given project timeline.

- It could be that coarser PLs do not need as much unfolding as the finer PLs since the coarser ones will target larger and rougher features in images. This should reduce the resources needed for this network, and can be implemented easily in conjunction with the previous suggestion of training each PL individually. Moreover, as demonstrated in Fig. 11, coarser PLs converge faster due to smaller image/DF resolution, which can further reduce the training time if each PL is trained separately.
- We mentioned that by visual inspection, it can be observed that the prediction quality near image borders is generally less than in regions that are close to the image center. Given that such behavior is common in convolutional neural networks, various suggestions like dedicating specialized filters for image boundaries [22] may be helpful to reduce this phenomenon.

References

1. Sören Mattsson Anders Brahme, e. a. *Swedish Cancer Society Radiation Therapy Research Investigation* in *Acta Oncologica* **41.** PMID: 28758858 (Taylor & Francis, 2002), 596–603. https://www.tandfonline.com/doi/abs/10.1080/028418602321028193.
2. Hering Alessa, v. G. *mlVIRNET: Multilevel Variational Image Registration Network* in *Medical Image Computing and Computer Assisted Intervention – MICCAI* (Springer International Publishing, Cham, 2019), 257–265. ISBN: 978-3-030-32226-7. https://link.springer.com/chapter/10.1007/978-3-030-32226-7_29.
3. Yang, D., Li, H., Low, D. A., Deasy, J. O. & El Naqa, I. *A fast inverse consistent deformable image registration method based on symmetric optical flow computation* in *Physics in Medicine and Biology* **53** (IOP Publishing, 2008), 6143. https://iopscience.iop.org/article/10.1088/0031-9155/53/21/017.
4. Ou, J. J., Ong, R. E. & Miga, M. I. *An evaluation of 3D modality independent elastography robustness to boundary condition noise* in *Medical Imaging 2007: Physiology, Function, and Structure from Medical Images* (eds Manduca, A. & Hu, X. P.) **6511** (SPIE, 2007), 65111X. https://doi.org/10.1117/12.709944.
5. Vishnevskiy, V., Gass, T., Szekely, G., Tanner, C. & Goksel, O. *Isotropic Total Variation Regularization of Displacements in Parametric Image Registration* in *IEEE Transactions on Medical Imaging* **36** (2017), 385–395. https://ieeexplore.ieee.org/document/7570266.
6. Kobler, E., Klatzer, T., Hammernik, K. & Pock, T. *Variational networks: connecting variational methods and deep learning* in *Pattern Recognition: 39th German Conference, GCPR 2017, Basel, Switzerland, September 12–15, 2017, Proceedings 39* (2017), 281–293. https://link.springer.com/chapter/10.1007/978-3-319-66709-6_23.
7. Vishnevskiy, V., Rau, R. & Goksel, O. *Deep Variational Networks with Exponential Weighting for Learning Computed Tomography* in *Medical Image Computing and Computer Assisted Intervention* (Springer International Publishing, 2019), 310–318. https://link.springer.com/chapter/10.1007/978-3-030-32226-7_35.
8. Jia, X. *et al. Learning a Model-Driven Variational Network for Deformable Image Registration* in *IEEE Transactions on Medical Imaging* **41** (2022), 199–212. https://ieeexplore.ieee.org/document/9525092.
9. Fanconi, C., Goksel, O. & Zhang, L. *Variational Networks for Learning Deformable Image Registration* (Unpublished). Semester Thesis (ETH Zurich, 2021).
10. Fu, Y. *et al. Deep learning in medical image registration: a review* in *Physics in Medicine and Biology* **65** (IOP Publishing, 2020). https://iopscience.iop.org/article/10.1088/1361-6560/ab843e.
11. Schwarz, L., Khamene, A., Zikic, D. & Navab, N. *Non-rigid registration using free-form deformations* Diploma Thesis (Technische Universität München, 2007). https://campar.in.tum.de/twiki/pub/Main/LorenSchwarz/thesis-070510.pdf.
12. Frohn-Schauf C., H. S. W. K. *Multigrid based total variation image registration* in *Computing and Visualization in Science* **11** (2008), 101–113. https://link.springer.com/article/10.1007/s00791-007-0060-2.
13. Engl, H., Hanke, M. & Neubauer, A. *Regularization of Inverse Problems* ISBN: 9780792361404. https://link.springer.com/book/9780792341574 (Springer, 2000).

14. He, K., Zhang, X., Ren, S. & Sun, J. *Deep Residual Learning for Image Recognition* in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778. https://ieeexplore.ieee.org/document/7780459.

15. Roth, S. & Black, M. *Fields of Experts: a framework for learning image priors* in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* **2** (2005), 860–867 vol. 2. https://ieeexplore.ieee.org/document/1467533.

16. Roth, S. & Black, M. J. *Fields of Experts* in *International Journal of Computer Vision* **82** (Apr. 2009), 205–229. https://link.springer.com/article/10.1007/s11263-008-0197-6.

17. Chen, Y., Yu, W. & Pock, T. *On learning optimized reaction diffusion processes for effective image restoration* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 5261–5269. https://ieeexplore.ieee.org/document/7299163.

18. Bajcsy, R. & Kovačič, S. *Multiresolution elastic matching* in *Computer Vision, Graphics, and Image Processing* **46** (1989), 1–21. https://www.sciencedirect.com/science/article/pii/S0734189X89800143.

19. Vishnevskiy, V., Gass, T., Szekely, G., Tanner, C. & Goksel, O. *Isotropic total variation regularization of displacements in parametric image registration* in *IEEE transactions on medical imaging* **36** (IEEE, 2016), 385–395. https://ieeexplore.ieee.org/document/7570266/.

20. Rueckert, D. *et al. Nonrigid registration using free-form deformations: application to breast MR images* in *IEEE Transactions on Medical Imaging* **18** (1999), 712–721. https://ieeexplore.ieee.org/document/796284.

21. *Anhir Grand Challenge DataSet* in (IEEE International Symposium on Biomedical Imaging (ISBI), 2018). https://anhir.grand-challenge.org/Data/.

22. Innamorati, C., Ritschel, T., Weyrich, T. & Mitra, N. J. *Learning on the Edge: Investigating Boundary Filters in CNNs* in *International Journal of Computer Vision* **128** (Apr. 2020), 773–782. https://link.springer.com/article/10.1007/s11263-019-01223-y.