Adjusting the Ground Truth Annotations for Connectivity-Based Learning to Delineate

Doruk Oner, Mateusz Koziński, Lenoardo Citraro, and Pascal Fua, *Fellow, IEEE*

***Abstract*— Deep learning-based approaches to delineating 3D structure depend on accurate annotations to train the networks. Yet in practice, people, no matter how conscientious, have trouble precisely delineating in 3D and on a large scale, in part because the data is often hard to interpret visually and in part because the 3D interfaces are awkward to use. In this paper, we introduce a method that explicitly accounts for annotation inaccuracies. To this end, we treat the annotations as active contour models that can deform themselves while preserving their topology. This enables us to jointly train the network and correct potential errors in the original annotations. The result is an approach that boosts performance of deep networks trained with potentially inaccurate annotations.**

***Index Terms*— Active contours, deep learning, delineation, neurons, snakes, vessels.**

I. Introduction

AS IN many areas of computer vision, deep networks now deliver state-of-the-art results for delineation tasks, such as finding axons and dendrites in 3D light microscopy images. However, their performance depends critically on the accuracy of the ground-truth data used to train them. This is especially true when the delineation task is treated as a segmentation one and the network is trained by minimizing the cross-entropy between the centerline predictions and ground-truth annotations, which is one of the most popular paradigms.

In practice, these so-called ground-truth annotations are usually supplied manually by an annotator who may not draw with the utmost accuracy and can therefore easily be a few voxels off the true centerline. This is not a matter of carelessness but a consequence of 3D delineation being truly difficult to do well on a large scale. As a result, inaccurate annotations are more the rule than the exception and this adversely affects how well the networks ultimately perform. One solution would be to have several annotators delineate the same data and combine their delineations. However, this would turn an already tedious, slow, and expensive process into an even slower and more expensive one that almost no one can afford.

In this paper, we introduce a method that explicitly accounts for annotation inaccuracies and delivers the same performance as if they were perfectly accurate. Our main insight is that the annotations are usually imprecise more in terms of the 3D location of the centerlines than of the topology of the graph they define. We can therefore treat them as deformable contours forming a graph that can be refined by moving its

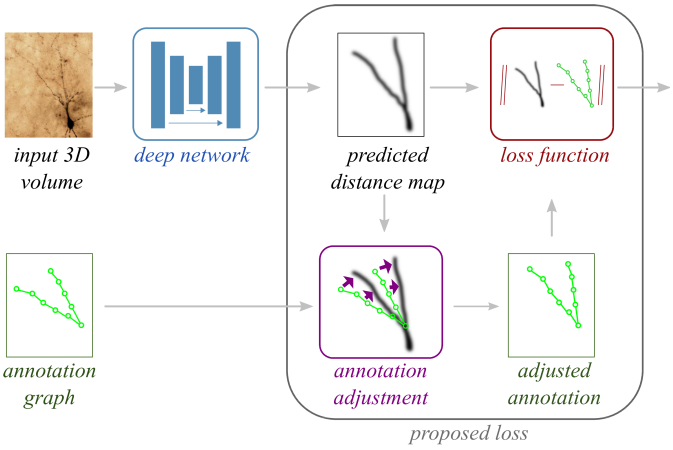


Fig. 1. **Our approach.** To account for annotation inaccuracies during training, we jointly train the network and adjust the annotations while preserving their topology.

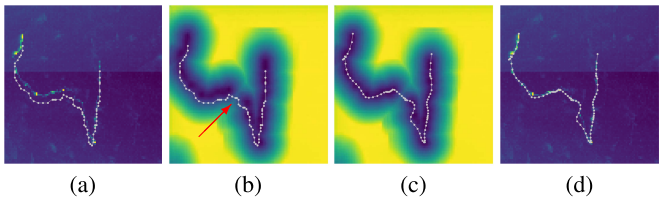


Fig. 2. **Correcting an inaccurate annotation.** (a) A microscopy scan of a neurite with an inaccurate annotation overlaid in white. (b) Distance map predicted by the deep net. Ideally, the pixels crossed by the centerline should have value zero (dark color). In practice, this is not always the case. There are non-zero values in the area indicated by the red arrow, presumably because the neurite is hardly visible there. Nevertheless, the distance map is sufficiently good to adjust the annotation. The adjusted annotation is shown in (c) and (d). This network retrained with adjusted annotations can now generate a better distance map even where the neurite is barely visible.

Manuscript received 19 April 2022; revised 7 July 2022; accepted 13 July 2022. Date of publication 21 July 2022; date of current version 2 December 2022. This work was supported in part by the Swiss National Science Foundation (SNSF) under 'Sinergia' Grant 177237 and in part by the Autrian Science Fund (FWF) 'Lise Meitner' Grant M3374. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to their version of the manuscript. *(Corresponding author: Mateusz Koziński.)*

Doruk Oner, Lenoardo Citraro, and Pascal Fua are with the Computer Vision Laboratory, EPFL, 1015 Lausanne, Switzerland (e-mail: doruk.oner@epfl.ch; leonardo.citraro@epfl.ch; pascal.fua@epfl.ch).

Mateusz Koziński is with the Institute of Computer Vision and Graphics, TU Graz, 8010 Graz, Austria (e-mail: mateusz.kozinski@icg.tugraz.at).

This article has supplementary downloadable material available at https://doi.org/10.1109/TMI.2022.3193072, provided by the authors.

Digital Object Identifier 10.1109/TMI.2022.3193072

nodes while preserving its structure. We cast this approach to training a deep network as a joint optimization over the network parameters and node positions. We then show that we can eliminate the node variables from the optimization problem, which can then be solved by minimizing a loss function. This loss function accounts for the annotation's lack of spatial precision. It can be minimized in the traditional manner and the output of the re-trained network used to refine the annotation.

Fig. 1 depicts our approach and Fig. 2 showcases its behavior. We will demonstrate that it brings substantial improvements when training networks to delineate neurons in two-photon and confocal microscopy image stacks. Hence, our contribution is an automated approach to better leveraging inaccurate training data, which, in our experience, represents the vast majority of data available to practitioners.

II. Related Work

A. Automated Delineation

Automatic delineation of curvilinear structures has been an active research topic for decades. It has evolved from manually designing filters that respond strongly to tubular structures [10], [21], [38] to feeding hand-designed features into boosted trees [3], [44], support vector machines [16], or GradientBoost [36], and finally to fully relying on neural networks [13], [22], [27], [28], [32].

The latter now routinely deliver the best performance when properly trained. However, obtaining accurately annotated data, especially in 3D, is a challenge. In practice it is rarely available in sufficient quantities. And what annotated data there is, is rarely accurate because manually delineating 3D structures is challenging. Introducing a degree of self-supervision is a way to address this difficulty [2], [9] but this does not detract from the fact that the training would work even better if the available annotated data were accurate. This can be partially ascribed to the fact that most current networks are trained by minimizing the standard cross entropy or differentiable intersection-over-union loss [24]. As pixel-wise measures, both are sensitive to even small misplacements of the linear structures' centerlines. In [29], this is partially addressed by introducing a loss component that accounts for global statistics of the network output, but the cross entropy remains a key component of the overall loss. Similarly, the method of [30] relies on introducing a topology-preserving term but still depends on the annotation being accurate.

Accuracy can be improved by having several people annotate and combining their results using robust statistics. This is effective but even more expensive than obtaining one set of annotations and therefore out of reach for most practitioners. The problem can be partially alleviated by annotating only in 2D projections of the 3D data volumes [20], [31], [47], which is easier, but may result in even less precise annotations than those performed in 3D.

A similar problem to the one we address here also arises in the context of two-dimensional semantic boundary detection. The outlines one finds in annotated training sets are often rather imprecise and training the networks to nevertheless discover contours that overlap with them is an issue. In [45], training is reformulated as simultaneously optimizing the parameters of a deep net and correcting the annotations by solving a mixed binary-continuous optimization problem. However, unlike in our approach, preservation of annotation topology is not warranted and the corrections may break the continuity of annotations. This is a major problem when tracing neurons or blood vessels, because topology changes influence the biological interpretation of the results. The same problem is addressed in [1] by proposing a neural layer and a loss function that can be added on top of an edge detector and make it possible to find more accurate contours than those in the annotations. However, because the regions are represented in an implicit fashion, there is no more guarantee than in [45] that the annotations' connectivity will be preserved. Connectivity being at the heart of our applications, we therefore chose to use explicit deformable models, such as those described below.

B. Handling Noisy Annotations

Even though we know of no other algorithm that adjusts the geometry of centerline annotations during training, explicitly accounting for the fact that the annotations are noisy has received some attention. In [40], annotations produced by non-expert annotators are accommodated by means of a dedicated distillation architecture and a noise-robust Dice loss. In [6], a dedicated network architecture and a semi-supervised training routine encourage equivariance to deformations to handle potential inaccuracies resulting from using a heuristic annotation tool. In [48], annotation noise is handled by a quality assessment module that discounts the loss in regions where the estimated label quality is low. Similarly, in [25], a distillation training setup and architecture based on self-attention are used to suppress the influence of erroneous labels on the trained network. In contrast to all these approaches, ours explicitly distinguishes between inaccuracies in position and topological errors. Because the former occur far more frequently than the latter, our loss function adjusts the centerline locations, while preserving the topology of the annotations.

C. Deformable Contour Models

Deformable contours [12], [18], [37] were initially introduced as a means to semi-automatically delineate simple contours while imposing smoothness constraints on the resulting outlines. They were later generalized to model network structures [5], [11] that can deform while preserving their topology. They are therefore well suited for refining our inaccurate annotations under the assumption they are topologically correct but that their locations are imprecise.

More recent deformable contours rely on minimizing energy functions generated by deep networks [7], [15], [23], [40], which enables end-to-end learning. Unlike in these methods, which rely on evolving the contour for segmenting the image at test time, our use of deformable contours is limited to adjusting the annotations during training.

Active appearance models [8] enable modelling the appearance of imaged objects, in addition to their shape. They can be learnt from coarse annotations, which are adjusted when fitting

the model to the data [33]. The level of detail of the active appearance model can then be increased and, before the more detailed model is fitted to the data, it can be initialized with the parameters of its less detailed version. In this work, we also adjust the annotation during learning, but represent them as network snakes, and train a deep convolutional network, instead of fitting an active appearance model.

III. Method

Given a set of microscopy stacks along with the corresponding and possibly imprecise centerline annotations, we want to train a deep net to produce precise delineation. To this end, when training the deep network, we adjust not only its weights but also the annotations themselves. We first present the vanilla training procedure without annotation adjustment and explain why it is sub-optimal when the annotations lack precision. We then formalize our training procedure with adjustment.

A. Standard Training Procedure

Let us consider a set of N microscopy scans $\{\mathbf{X}_i\}_{1\leq i\leq N}$ and corresponding centerline annotations $\{\hat{\mathbf{y}}_i\}_{1\leq i\leq N}$, in the form of distance maps of the same size as the scans. Voxel p of annotation $\hat{\mathbf{y}}$, denoted $\hat{\mathbf{y}}[p]$, contains the distance from the center of p to the closest centerline. Let $F(\cdot;\Theta)$ be a deep network, with weights Θ. It takes a scan $\mathbf{X}_i$ as input and return a volume $\mathbf{y}_i = F(\mathbf{X}_i;\Theta)$, containing a delineation of centerlines visible in $\mathbf{X}_i$. To keep the notation concise, we omit the dependencies on $\mathbf{y}_i$ on Θ. The traditional approach to learning the network weights is to make $\mathbf{y}_i$ as close as possible to $\hat{\mathbf{y}}_i$ by solving

$$\Theta^* = \arg\min_{\Theta} \sum_{i=1}^{N} \mathcal{L}\big(\hat{\mathbf{y}}_i, \mathbf{y}_i\big), \tag{1}$$

where the loss term $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ measures the voxel-wise difference between the annotation and the prediction. In our experiments, we take $\mathcal{L}$ to be the Mean Square Error. This assumes that the deviations of the annotations from actual centerline trajectories are small and unbiased. In reality, they rarely are. Hence, the network learns to accommodate this uncertainty in the annotations by blurring the predictions. At test time, this leads to breaking the continuity of predictions wherever the image quality is compromised by high level of noise or low contrast between the foreground and the background, as illustrated by Fig. 2.

B. Overview of Our Approach

The formulation of Eq. 1 assumes that the deviations of the annotations from reality are small and unbiased. This work is predicated on the fact that they rarely are and that we must allow for substantial non-Gaussian deviations from the original annotations. Thus, instead of encoding the annotations in terms of volumes $\hat{\mathbf{y}}_i$, we represent the annotated centerline $\mathcal{C}_i$ of each $\mathbf{X}_i$ as a graph, with the set of vertices $\mathcal{V}_i$ and the set of edges $\mathcal{E}_i$. Each vertex $v \in \mathcal{V}_i$ has a 3D coordinate c_v, and each edge $(u, v) \in \mathcal{E}_i$ represents a short line segment. This is shown in Fig. 2 where the circles along the annotations denote the vertices. Let $\mathbf{c}_i$ be the vector formed by concatenating coordinates of all the vertices of $\mathcal{V}_i$. To accommodate the possible lack of precision of the annotations, we let $\mathbf{c}_i$ change its initial value. Doing so changes the shape of $\mathcal{C}_i$ but preserves its topology and can be used to explicitly model the deviation of the annotated centerlines from their true position. In other words, the minimization problem can be reformulated as finding

$$\Theta^*, \mathbf{C}^* = \arg\min_{\Theta,\mathbf{C}} \sum_{i=1}^{N} L(\mathbf{c}_i, \mathbf{y}_i) + R(\mathbf{c}_i),$$
$$\text{where} \qquad L = \mathcal{L}\big(D(\mathbf{c}_i), \mathbf{y}_i\big)\ ; \tag{2}$$

$\mathbf{C}$ is the vector obtained by concatenating all the $\mathbf{c}_i$; R is a regularization term that forces the deformed centerlines to be smooth, and that we define in Sec. III-C; $\mathcal{L}$ is the same MSE as in Eq. 1; and D is a distance transform that creates a volume in which a voxel with coordinates q is assigned its truncated distance to the closest edge of $\mathcal{C}$. Formally, we write

$$D(\mathbf{c})[q] = \min\{\delta(\mathbf{c}, q), d\}, \tag{3}$$
$$\text{where} \quad \delta(\mathbf{c}, q) = \min_{(u,v)\in\mathcal{E}} \min_{0\leq\phi\leq 1} \|\phi c_u + (1-\phi)c_v - q\|_2, \tag{4}$$

d is the threshold used to truncate the distance map, and the minimization over ϕ serves to find the point on edge (u, v), that is closest to q.

Solving the problem of Eq. 2 means training the network to find centerlines that are smooth and with the same topology as the annotations. This is what we want but, unfortunately, this optimization problem involves two kinds of variables, the components of $\mathbf{C}$ and Θ respectively, which are not commensurate in any way. In practice, this makes optimization difficult. We address this problem by eliminating the $\mathbf{C}$ variables by rewriting Eq. 2 as

$$\mathbf{c}_i^*(\mathbf{y}_i) = \arg\min_{\mathbf{c}} L(\mathbf{c}_i, \mathbf{y}_i) + R(\mathbf{c}_i), \tag{5}$$

$$\Theta^* = \arg\min_{\Theta} \sum_{i=1}^{N} L\big(\mathbf{c}_i^*(\mathbf{y}_i), \mathbf{y}_i\big) + R\big(\mathbf{c}_i^*(\mathbf{y}_i)\big), \tag{6}$$

In the following section, we describe our choice of R and the formulation of $\mathbf{c}_i^*(\mathbf{y}_i)$ that results from it. Eq. 6 is a standard continuous optimization problem that we can solve using the usual tools of the trade.

C. Annotations as Network Snakes

We propose to represent each $\mathcal{C}_i$ as a network snake, and to take R to be a classical sum of spring and elasticity terms [5], [11]. This regularization term takes the form

$$R(\mathbf{c}) = \alpha \sum_{(u,v)\in\mathcal{E}} \|c_u - c_v\|^2 + \beta \sum_{(u,v,w)\in\mathcal{T}} \|c_u - 2c_v + c_w\|^2, \tag{7}$$

where α and β are hyper-parameters that balance the strength of the two terms, $\mathcal{E}$ is the set of edges of $\mathcal{C}$ and $\mathcal{T}$ is the set of node triples (u, v, w) such that $(u, v) \in \mathcal{E}$, $(v, w) \in \mathcal{E}$, and

v is a node of order two, that is, not a junction of multiple snake branches. As shown in [5], [11], R can be written as

$$R(\mathbf{c}_i) = \frac{1}{2}\mathbf{c}_i^T \mathbf{A}\mathbf{c}_i, \tag{8}$$

where A is a sparse symmetric matrix. Given this quadratic formulation of R, we can use the well-known semi-implicit scheme introduced to deform snakes, also known as active contour models [18], to minimize Eq. 5. It involves initializing each snake $\mathbf{c}_i^0$ to the manually produced annotation and refining it by iteratively solving

$$(\mathbf{A} + \gamma\,\mathbf{I})\mathbf{c}_i^{t+1} = \gamma\,\mathbf{c}_i^t - \frac{\partial L}{\partial \mathbf{c}}(\mathbf{c}_i^t, \mathbf{y}_i) \tag{9}$$

for $\mathbf{c}_i^{t+1}$, where γ is a hyper-parameter known as the *viscosity* and is inversely proportional to the step size in each iteration. We refer the reader to [18] for the complete derivation. Here we only note, that when the iteration stabilizes, we have $\forall i, \mathbf{c}_i^t \approx \mathbf{c}_i^{t+1}$. We can therefore denote the stable vector of node locations by $\mathbf{c}_i^*$, substitute $\mathbf{c}_i^{t+1} \approx \mathbf{c}_i^t \approx \mathbf{c}_i^*$ in Eq. 9, and use the derivative of Eq. 8, to write

$$\forall i, \quad \frac{\partial R}{\partial \mathbf{c}}(\mathbf{c}_i^*) + \frac{\partial L}{\partial \mathbf{c}}(\mathbf{c}_i^*, \mathbf{y}_i) \approx 0, \tag{10}$$

which means that $\mathbf{c}^*$ minimizes $R + L$ and is a solution of Eq. 5.

In practice, we solve Eq. 9 by inverting the matrix $(\mathbf{A} + \gamma\,\mathbf{I})$ at the start of the training procedure and then multiplying the right-hand-side of the equation by the inverse at each iteration. Hence, we write

$$\mathbf{c}_i^{t+1} = (\mathbf{A} + \gamma\,\mathbf{I})^{-1}\big(\gamma\,\mathbf{c}_i^t - \frac{\partial L}{\partial \mathbf{c}}(\mathbf{c}_i^t, \mathbf{y}_i)\big). \tag{11}$$

We perform the update or Eq. (11) for $0 \leq t < T$. We take $T = 10$ in our implementation, which is sufficient for the process to stabilize, and denote the result of the last iteration by $\mathbf{c}_i^*(\mathbf{y}_i) = \mathbf{c}_i^T$.

D. Computing the Gradients of the Loss Function

Performing the minimization in Eq. 6 requires computing at each iteration the gradient of the loss with respect to the network output $\mathbf{y}_i$. To avoid cluttering the notation, we denote $\mathbf{c}^*(\mathbf{y}_i)$ by $\mathbf{c}^*$. The gradient can then be expressed as

$$\begin{aligned}
&\frac{\partial}{\partial \mathbf{y}}\big(L(\mathbf{c}_i^*, \mathbf{y}_i) + R(\mathbf{c}_i^*)\big) \\
&\quad = \frac{\partial L}{\partial \mathbf{y}}(\mathbf{c}_i^*, \mathbf{y}_i) + \big(\frac{\partial L}{\partial \mathbf{c}}(\mathbf{c}_i^*, \mathbf{y}_i) + \frac{\partial R}{\partial \mathbf{c}}(\mathbf{c}_i^*)\big)\frac{\partial \mathbf{c}_i^*}{\partial \mathbf{y}} \\
&\quad \approx \frac{\partial L}{\partial \mathbf{y}}(\mathbf{c}_i^*, \mathbf{y}_i),
\end{aligned} \tag{12}$$

where we used Eq. 10 to eliminate the second term. In other words, even though $\mathbf{c}^*$ is a function of $\mathbf{y}_i$, we do not need to compute its derivatives with respect to $\mathbf{y}_i$ to train the neural network. We only need those of L, and can treat $\mathbf{c}^*$ as a constant when evaluating them. Therefore, the only difference between using our approach and the standard one of Section III-A is that instead of evaluating the loss using the original annotation $\mathbf{c}$, we use its optimized version $\mathbf{c}^*$. We call this approach *SnakeFull* and it is depicted at the top of Fig. 3.

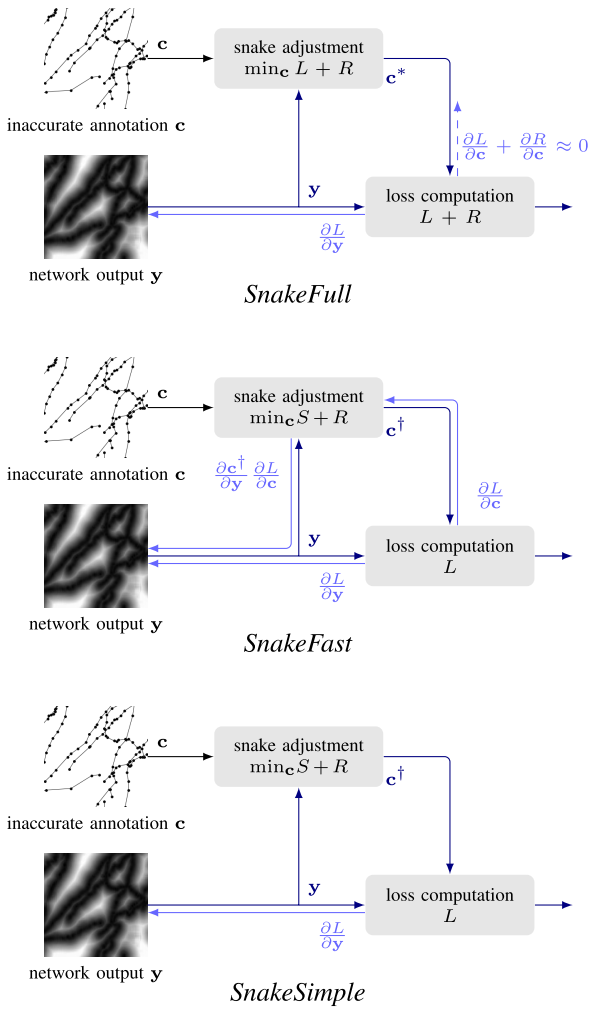


Fig. 3. The three approaches to training described in Sec. III-D and III-E. In *SnakeFull*, the training objective is also used as the objective of the snake. This makes some gradient components vanish, simplifying gradient computation, but results in snake updates that are costly to compute. *SnakeFast* can accommodate an arbitrary snake objective, which makes it faster than *SnakeFull*, even though it requires backpropagation through a sequence of snake updates. In *SnakeSimple*, the backpropagation over the snake updates is simply omitted. This approach is the fastest. We analyze the accuracy vs. speed tradeoff induced by these three methods in section IV.

E. Speeding Things Up

We will show in Section IV that *SnakeFull* performs well but is slow to train. The culprit is the term $\frac{\partial L}{\partial \mathbf{c}}$ in the update Eq. 11, which involves a time-consuming computation of the gradient of a distance map. To speed things up, we introduce a faster approach that we call *SnakeFast*. In it, we replace the term L in Eq. 5 by a simpler objective function S directly inspired by the classical external snake energy [18]. We take it to be

$$S(\mathbf{c}, \mathbf{y}) = \sum_{v \in \mathcal{V}} \big(\mathbf{y} * G\big)[c_v], \tag{13}$$

where $*G$ denotes a convolution with a Gaussian kernel and $\mathbf{y}[c_v]$ denotes the network output at vertex v. S is very similar to the energies used in traditional network snake formulations [5], [11]. Importantly, S and its gradients are easy and

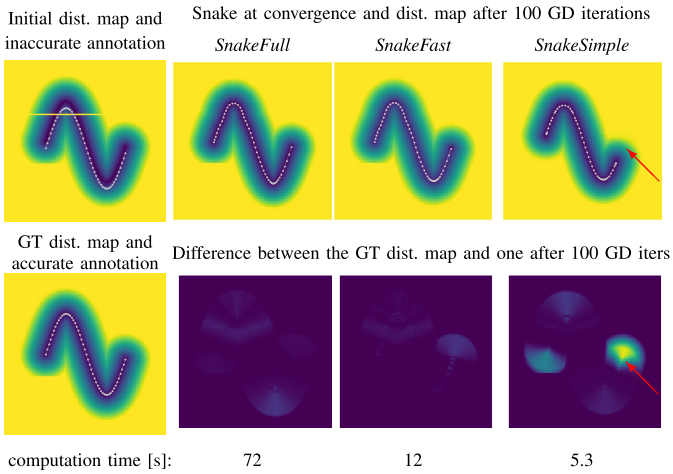


Fig. 4. **Compared behavior of *SnakeSimple*, *SnakeFast*, and *SnakeFull* on a synthetic 2D example.** (*Left column*) At the bottom, distance map and corresponding annotation. At the top, we simulated an unwarranted break in the distance map (horizontal yellow line) and shifted the annotation by several pixels. (*Other Columns*) In three separate runs, we performed 100 Gradient Descent using either *SnakeFull*, *SnakeFast*, or *SnakeSimple*. In the *top row*, we show the corrected annotation and the updated distance maps. The *bottom row* depicts the differences between the updated maps and the ground-truth one. We also indicate the computation times. *SnakeFull* removes the interruption in the distance map but the computation is slow. *SnakeFast* is much faster and fills the gap in the distance map almost as well. *SnakeSimple* is even faster but yields a corrected annotation that is too short, as highlighted by the red arrow.

fast to compute because doing so only requires convolving $\mathbf{y}$ with a Gaussian kernel and sampling the result at the locations of the snake nodes. Deforming the annotations then involves finding

$$\mathbf{c}_i^\dagger(\mathbf{y}_i) = \arg\min_{\mathbf{c}} S(\mathbf{c}, \mathbf{y}_i) + R(\mathbf{c}), \tag{14}$$

which means that the sum of distance values along the snake should be as low as possible while preserving snake smoothness. As in Section III-C, the snake update takes the form

$$\mathbf{c}_i^{t+1} = (\mathbf{A} + \gamma\,\mathbf{I})^{-1}\big(\gamma\,\mathbf{c}_i^t - \frac{\partial S}{\partial \mathbf{c}}(\mathbf{c}_i^t, \mathbf{y}_i)\big). \tag{15}$$

In practice, we take $\mathbf{c}_i^\dagger(\mathbf{y}_i) = \mathbf{c}_i^T$, where $T = 10$, as in Section III-C. Finally, we take the network training objective to be

$$\Theta^* = \arg\min_{\Theta} \sum_i L(\mathbf{c}_i^\dagger(\mathbf{y}_i), \mathbf{y}_i), \tag{16}$$

where we still use the original L of Eq. 2. We do this because S only depends on a small subset of voxels of $\mathbf{y}$. Hence, it only provides a sparse supervisory signal and is not well suited as the training objective for the network that produces a dense distance map. The gradient of the objective of Eq. 16 is

$$\frac{\partial}{\partial \mathbf{y}} L(\mathbf{c}^\dagger, \mathbf{y}) = \frac{\partial L}{\partial \mathbf{y}}(\mathbf{c}^\dagger, \mathbf{y}) + \frac{\partial L}{\partial \mathbf{c}}(\mathbf{c}^\dagger, \mathbf{y})\frac{\partial \mathbf{c}^\dagger}{\partial \mathbf{y}}. \tag{17}$$

Because we minimized S instead of L in Eq. 14, we can no longer assume that the second term is zero as we did in Section III-D. Hence, to compute it during the minimization, we backpropagate through the snake update procedure of Eq. 15, as depicted by the middle row of Fig. 3. In practice, we use the autograd functionality of Pytorch to this end.

The non-zero second term of Eq. 17 helps guide the snake to a position where the data loss L is low and ultimately influences the distance map that our deep network F outputs. It could be argued that ignoring this term so that the networks focuses exclusively on fitting the annotations would be preferable. To test this assertion, we implemented *SnakeSimple*, a third variant or our approach in which we take the second term of Eq. 17 to be zero. *SnakeSimple* is even faster than *SnakeFast*. In essence, it is a simplified version of *SnakeFull* and *SnakeFast* in which we successively optimize the network weights and then the snake position without any direct interaction between these two optimization steps.

Fig. 4 uses a synthetic example to illustrates the differences between our three variants. *SnakeFast* and *SnakeFull* yield similar results with the former being much faster whereas *SnakeSimple* is even faster but prone to generating artifacts. We now turn to our experimental results on real data that confirm this.

IV. Experiments

A. Datasets

We tested our approach on the following data sets.

- The *Brain* data set comprises fourteen two-photon microscopy 3D scans of fragments of a mouse brain, with manually traced neurites. We use four volumes for testing and ten for training, each of size $200 \times 250 \times 250$ voxels and spatial resolution $0.3 \times 0.3 \times 1.0$ μm.
- The *Neurons* data set contains two 3D images of neurons in a mouse brain. They had been outlined manually while viewing the sample under a microscope and the image was captured later. The sample deformed in the meantime, exacerbating misalignment between the annotation and the image. We use one stack of size $151 \times 714 \times 865$ voxels and a resolution of 1 μm for training and one of size $228 \times 764 \times 1360$ for testing.
- The *MRA* is a publicly available set of Magnetic Resonance Angiography brain scans [4]. It consists of 42 annotated stacks, which we cropped to $416 \times 320 \times 128$ voxels by removing their empty margins. Their resolution is $0.5 \times 0.5 \times 0.6$ mm. We randomly partitioned the data into 31 training and 11 test volumes.

None of our data sets can be considered as perfectly annotated. All annotations were performed as accurately as possible, but their precision is affected by the uneven distribution of the dye, image noise, and generic difficulty of annotating 3D volumes. In *Neurons*, the difficulty is compounded by the fact that the annotation were performed live days before image acquisition, and the sample deformed in the meantime.

B. Metrics

We used the following performance metrics.

- *CCQ*. Since standard segmentation metrics such as the F1 score [35] and precision-recall break-even point [26] are very sensitive to misalignment of thin structures, we use

the *correctness-completeness-quality*, which is specifically designed for linear structures [43]. Correctness corresponds to precision, completeness to recall, and quality to the intersection-over-union. However, the notion of a true positive is relaxed from perfect coincidence of the ground truth and the prediction to their co-occurrence within a distance of d pixels. We used $d = 3$. Although it accounts for possible ground truth misalignment, *CCQ* is still a voxel-wise metric, insensitive to topological errors, such as short interruptions of neurites.

- *APLS*. The *Average Path Length Similarity* is defined as the aggregation of relative length differences of shortest paths between pairs of corresponding end points, randomly sampled in the reconstructed and predicted graphs. It was introduced to evaluate road map reconstructions from aerial images [39] and aims to evaluate the connectivity of the reconstructions, as opposed to their pixel-wise accuracy, which makes it a perfect performance measure for our task.
- *TLTS*. The *Too-Long-Too-Short* is another performance criterion based on statistics of relative lengths of shortest paths between corresponding pairs of end points in the prediction and the ground truth [42]. We report the fraction of *correct* paths, that is, predicted paths whose relative length difference to the corresponding ground truth paths is lower than 15%.

C. Architectures and Training Details

Our contribution lies in the updating of the annotations and the loss function we use to achieve it, which should improve performance independently of any specific network architecture. To demonstrate this, we used two different architectures.

- *UNet*. A 3D *UNet* [34] with three max-pooling layers and two convolutional blocks. The first layer has 64 filters. Each convolution layer is followed by a batch-normalization and dropout with a probability of 0.15. During training, we randomly crop sub-volumes of size $96 \times 96 \times 96$ and flip them along each dimension with probability 0.5. We combine them into batches of 8.
- *DRU*. A recurrent architecture iteratively refining segmentation output 3 times [41]. The first layer has 64 filters. Each convolution layer is followed by a group-normalization and dropout with a probability of 0.15. During training, we randomly crop sub-volumes of size $96 \times 96 \times 96$ and flip them along each dimension with probability 0.5. We combine them into batches of 4. To compute the loss function, we average the outputs of all 3 refinement steps. During testing, the output of the final step is used to evaluate performance.

We trained both architectures in four different ways: by minimizing the Mean Squared Error to the original annotations, which we will refer to as *OrigAnnot*, and by using the *SnakeSimple*, *SnakeFull*, and *SnakeFast* variants of our approach, described in Sections III-D and depicted by Fig. 3. In all cases, we used Adam [19] with the learning rate set to $1e-4$, and a weight decay of $1e-4$. At test time, the predicted distance map were thresholded at 2 and skeletonized to obtain centerlines. To compute the *TLTS* and *APLS* scores, we converted them into graphs.

TABLE I
PERFORMANCE OF DEEP NETS TRAINED WITH DIFFERENT LOSS FUNCTIONS ON OUR THREE DATA SETS AND THE TIME NEEDED FOR SINGLE TRAINING ITERATION

	Method	Pixel-wise			Topology-aware		iter. t.
		Corr.	Compl.	Qual.	APLS	TLTS	s
Brain	*UNet–OrigAnnot*	98.9	91.3	90.4	80.3	80.9	2.8
	UNet–SnakeSimple	98.4	92.5	91.2	84.2	83.4	3.8
	UNet–SnakeFull	99.0	94.4	93.5	89.3	85.9	18.9
	UNet–SnakeFast	98.7	95.0	93.8	91.1	85.9	5.2
	DRU–OrigAnnot	97.2	94.0	91.5	84.3	83.9	2.7
	DRU–SnakeSimple	97.4	95.2	92.9	90.8	85.9	3.8
	DRU–SnakeFull	96.9	96.9	94.1	**91.8**	**89.3**	19.1
	DRU–SnakeFast	97.0	97.1	94.2	91.7	88.1	5.3
	NR-Dice	97.7	97.0	**94.8**	81.0	83.6	2.8
	QAM	94.5	98.8	93.5	87.3	84.5	4.2
	DS6	97.5	97.0	94.7	83.8	84.1	5.8
Neurons	*UNet–OrigAnnot*	81.8	83.5	70.4	65.8	63.6	2.8
	UNet–SnakeSimple	83.0	83.9	71.6	70.4	68.8	3.4
	UNet–SnakeFull	83.5	85.4	73.1	74.2	69.9	17.8
	UNet–SnakeFast	83.1	85.5	72.9	73.9	70.2	4.9
	DRU–OrigAnnot	82.1	86.5	72.8	68.9	69.5	2.7
	DRU–SnakeSimple	83.2	87.7	74.5	73.8	74.6	3.5
	DRU–SnakeFull	84.4	88.5	76.1	74.8	**78.1**	18.3
	DRU–SnakeFast	84.2	88.9	**76.2**	**75.1**	77.7	5.1
	NR-Dice	85.2	83.4	72.8	67.6	65.2	2.8
	QAM	89.8	79.3	72.8	71.2	68.6	4.2
	DS6	83.2	81.6	70.0	71.0	68.8	5.8
MRA	*UNet–OrigAnnot*	90.1	72.2	66.9	49.8	50.4	2.8
	UNet–SnakeSimple	89.9	73.1	67.5	53.5	53.1	3.7
	UNet–SnakeFull	90.2	73.5	68.0	**55.6**	55.0	18.5
	UNet–SnakeFast	90.3	73.5	68.1	55.4	55.2	5.1
	DRU–OrigAnnot	80.2	79.3	66.3	48.7	49.9	2.7
	DRU–SnakeSimple	80.7	79.9	67.1	53.3	53.0	3.7
	DRU–SnakeFull	80.9	80.5	67.6	**55.6**	55.2	18.8
	DRU–SnakeFast	81.0	80.5	67.7	55.3	**55.4**	5.2
	NR-Dice	85.5	77.3	**68.3**	50.2	53.8	2.8
	QAM	80.2	80.1	66.8	54.3	54.2	4.2
	DS6	82.0	80.3	68.1	55.0	54.9	5.8

D. Label Correction Baselines

As noted in section II, we do not know of other methods that deform the annotation graph during training, while maintaining its topology. However, there are methods designed to train deep nets using noisy annotations, where the noise is understood as flipping some pixel labels. In the following section, we compare our algorithm to three such methods:

- *NR-Dice*. A *UNet* trained with the Noise Robust Dice Loss proposed in [40].
- *QAM*. An architecture with an auxiliary deep network to recognize annotations that might be wrong and downplay their importance during training [48].
- *DS6*. A Siamese architecture and a training routine dedicated to enforcing equivariance of the network to deformations [6].

E. Comparative Evaluation

We present example reconstructions in Fig. 5 and Fig. 6. As shown in Tab. I, *SnakeFull* and *SnakeFast* outperform

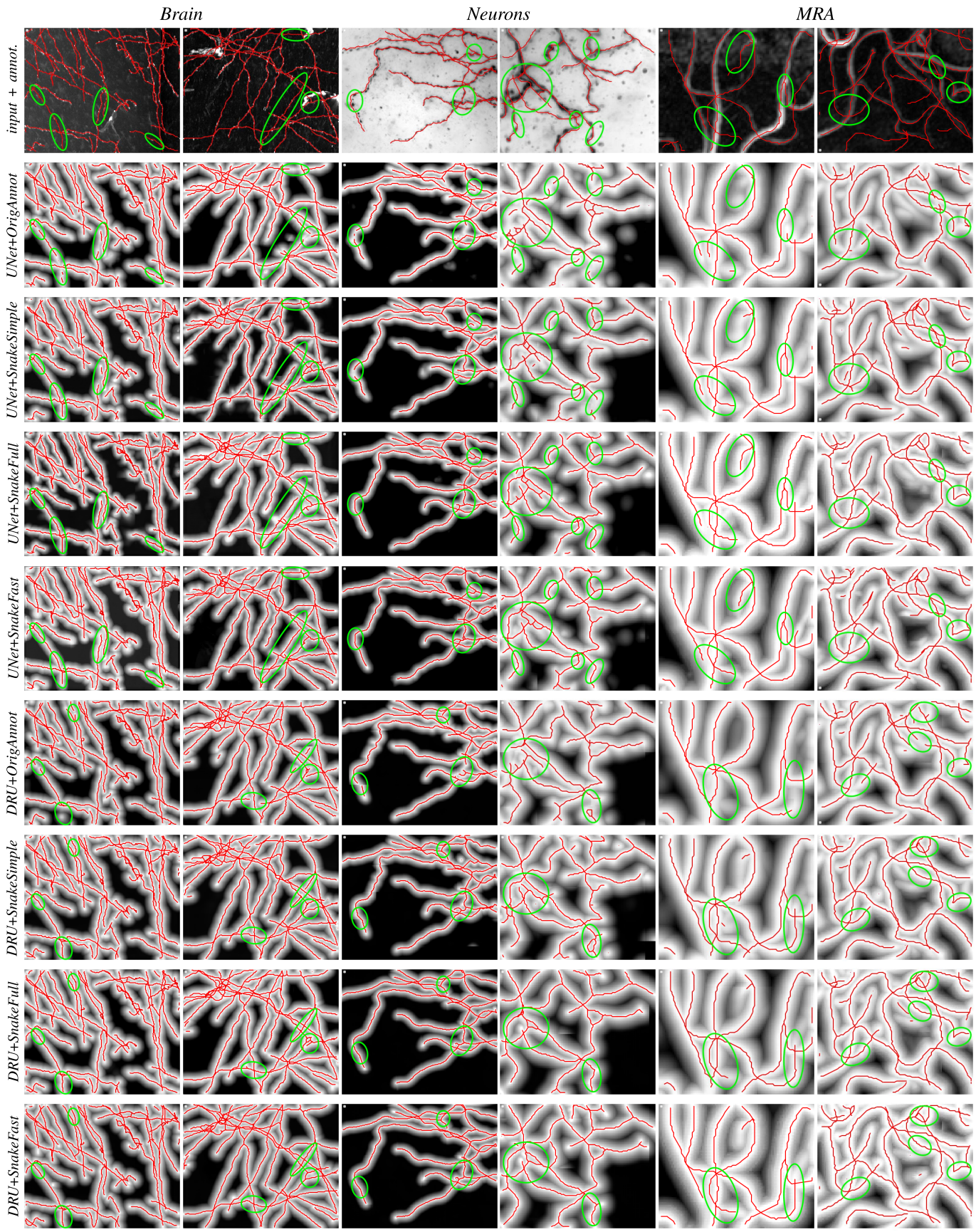


Fig. 5. Test predictions of different methods on three data sets. The green ellipses denote areas where training with the original annotations results in unwarranted breaks in the delineations whereas our approach does not.

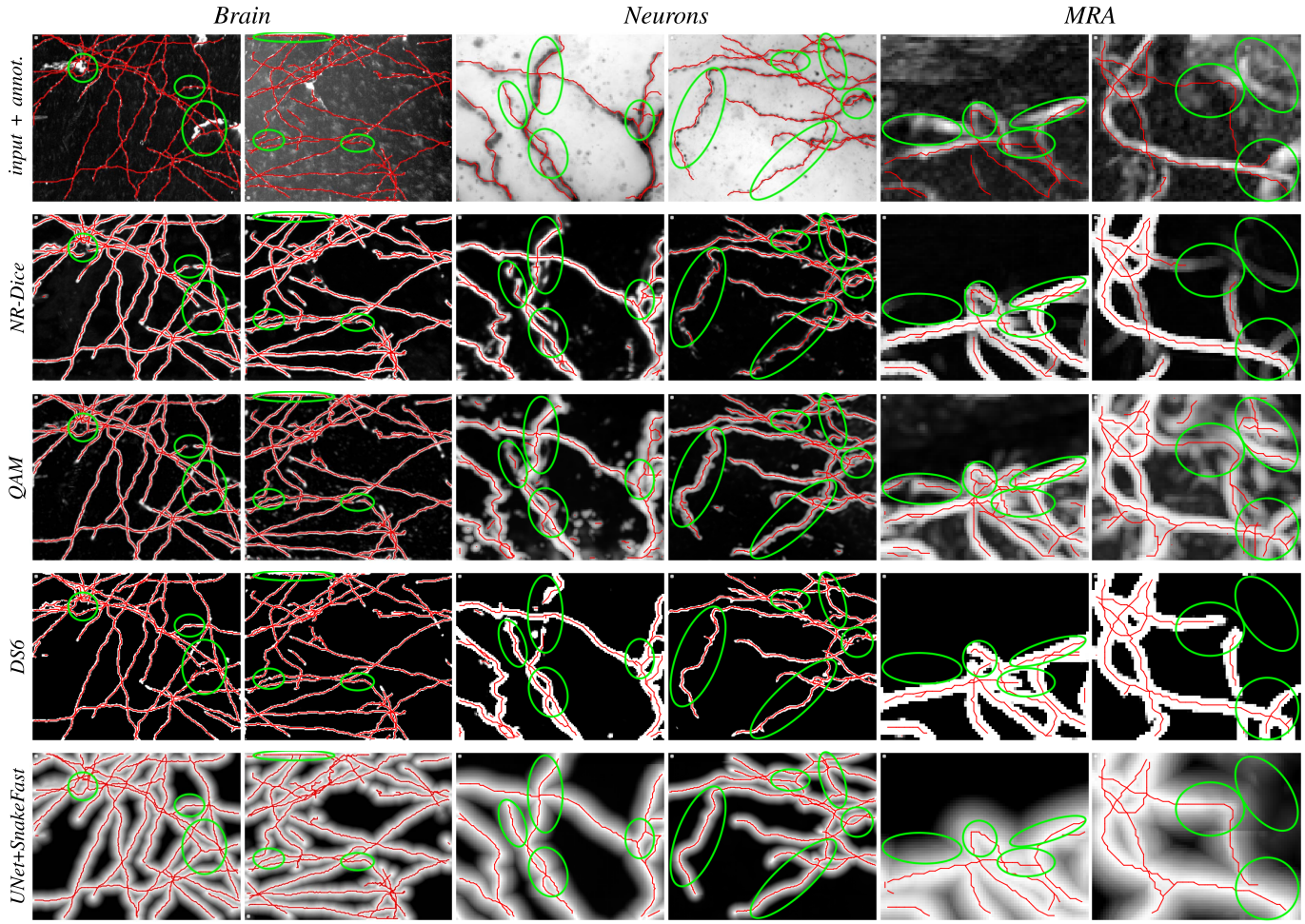


Fig. 6. Qualitative comparison of the results of *SnakeFast* to existing methods of training with noisy labels. The green ellipses denote areas where baselines result in unwarranted breaks in the delineations at test time whereas our approach does not.

OrigAnnot in *CCQ* terms by a small margin, and in *APLS* and *TLTS* terms by a significantly larger one, which confirms that the main benefit of our loss is the improved connectivity of the predictions. As can be seen in Fig 5, our approach to training yields delineations with fewer unwarranted breaks and longer uninterrupted curvilinear segments.

On average *UNet* and *DRU* perform best when trained with *SnakeFull* and *SnakeFast*. However, *SnakeFast* requires three times less time per training iteration. *SnakeSimple* delivers a further 20-30% speedup but incurs a clear performance drop. Crucially, these conclusions apply to both the *UNet* and *DRU* architectures. In fact, the performance gain resulting from switching from *OrigAnnot* to *SnakeFast* is larger than the one resulting from changing from the simpler *UNet* to the more sophisticated *DRU* while retaining the standard *OrigAnnot* approach to training.

In short, *SnakeFast* represents an excellent compromise between training time and performance. This being said, at test time, the run-time is the same no matter how the network was trained, because there is no alignment of annotations anymore. Hence, given sufficient computational resources, *SnakeFull* is also a valid option.

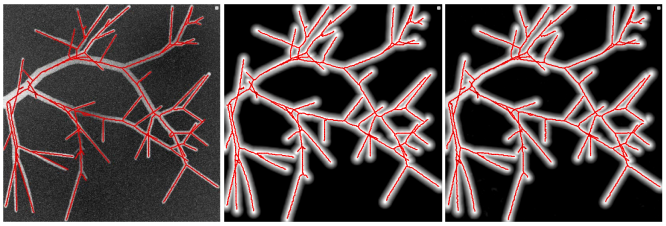
The bottom third of each part of Tab. I measures the performance of the methods designed to accommodate label noise, as described in Section IV-D. Because they don't explicitly preserve annotation topology and we do, *UNet* trained with *SnakeFast* outperform these methods in terms of the topology-aware scores but not necessarily in terms of the pixel-aware ones, which are note our main concern.

TABLE II
PERFORMANCE OF A *UNet* TRAINED WITH THE *OrigAnnot* AND WITH *SnakeFast* ON THE *Synthetic* DATA SET WITH PRECISE ANNOTATIONS

		Pixel-wise			Topology-aware		iter. t.
Arch.	Method	Corr.	Compl.	Qual.	APLS	TLTS	s
UNet	*OrigAnnot*	86.9	86.5	**77.2**	92.8	89.0	2.8
	SnakeFast	86.6	86.7	77.1	**93.2**	**89.3**	4.8

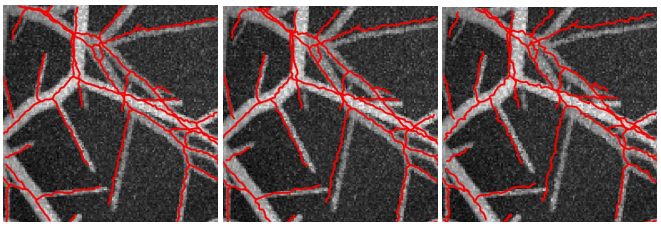
F. Perfectly Accurate Annotations

Having demonstrated that our loss function improves delineation results when the annotations lack spatial precision in Sec. IV-E, we now investigate its behavior when the annotation is precise. Since it is virtually impossible to precisely annotate 3D microscopy scans, we resort to synthetic data set *Synthetic*, which we generated using the VascuSynth algorithm [14], [17] and its implementation [46]. The images are generated from vascular graphs, which we use as perfectly accurate annotations. We used twenty stacks for training and ten for



input and annotation result of *OrigAnnot* result of *SnakeFast*

Fig. 7. Results of training with the precise annotations of the *Synthetic* data set. When the annotations are precise, *SnakeFast* performs as well as training with the *OrigAnnot*.



Level 1 *Level 2* *Level 3*

Fig. 8. **Annotation Deformation Levels.** The deformation magnitude increases from left to right.

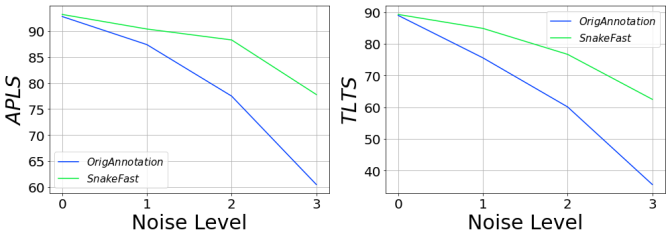


Fig. 9. **Increasing the amount of deformation.** *APLS* and *TLTS* scores as a function of the deformation level. The *OrigAnnot* scores decrease fast whereas those of *SnakeFast* decrease much more slowly.

testing, each of size $400 \times 400 \times 400$. Fig. 7 shows the maximum-intensity projection of a test stack. The results, presented in Tab. II, confirm that, for perfectly accurate annotations, our method reduces to standard training with the MSE without incurring any performance drop.

G. Increasing Annotation Inaccuracy

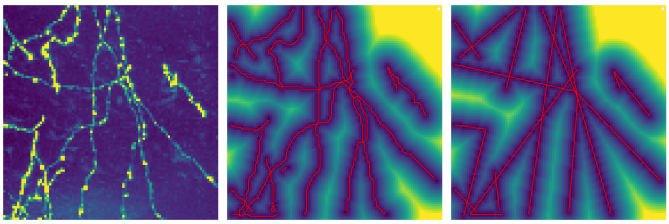
To investigate how increasing the level of inaccuracy of the annotations affects the performance of a *UNet* trained with *SnakeFast*, we perturbed the annotations of the *Synthetic* data set. We applied a random deformation field that varies slowly across space to each annotation graph. We modulated its amplitude to change the level of inaccuracy. This produced three sets of annotations, as depicted by Fig. 8. We trained the network on each of them and present the results in Fig. 9. When the network is trained with *SnakeFast*, its connectivity-related scores degrade much slower than when trained using *OrigAnnot*.

H. Reducing Annotation Effort

The robustness of *SnakeFast* to deviations in the annotation inspired us to ask another question: Can this loss function

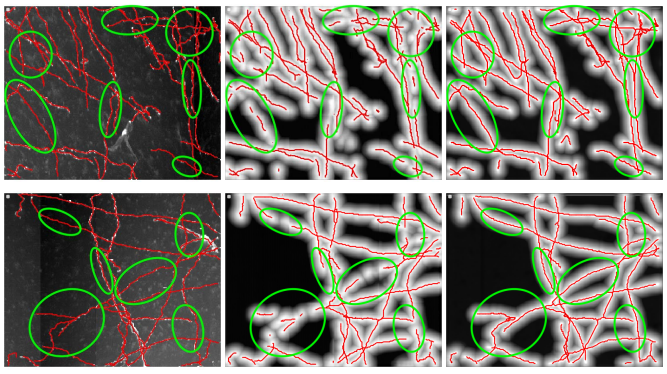
TABLE III
PERFORMANCE OF *UNet* TRAINED USING *SnakeFast* AND *OrigAnnot* ON THE *Brain* DATA SET WITH VERY COARSE ANNOTATIONS. PERFORMANCE OF *UNet* TRAINED USING THE PRECISE ANNOTATIONS SHOWN FOR REFERENCE

		Pixel-wise			Topology-aware	
Annot.	Method	Corr.	Compl.	Qual.	APLS	TLTS
coarse	*OrigAnnot*	85.2	67.6	60.4	46.5	50.9
	SnakeFast	97.6	87.0	**85.3**	**66.8**	**73.5**
precise	*OrigAnnot*	98.9	91.3	90.4	80.3	80.9
	SnakeFast	98.7	95.0	**93.8**	**91.1**	**85.9**



(a) (b) (c)

Fig. 10. **Coarse Annotations** (a) Training image of a neurite (b) Distance map obtained from original annotation overlaid in red (c) Distance map obtained from coarse annotation overlaid in red. Coarse annotations are obtained by connecting neurite end points and bifurcations with straight lines, and are easier to perform than full annotations.



input *OrigAnnot* *SnakeFast*

Fig. 11. Results of training a *UNet* with *OrigAnnot* and *SnakeFast* on the *Brain* data set with easy annotations.

be used to train deep networks with annotations that are simplified to the point where they become much easier, faster, and therefore cheaper to obtain? To answer this, we trained the *UNet* with *SnakeFast* and *OrigAnnot* on the *Brain* data set with very coarse annotations. We obtained them by connecting neurite branching- and end-points with straight lines, as shown in Fig. 10. The results are presented in Tab. III. As expected, training on the coarse annotations without adjusting them results in a significant performance drop as compared to training on precise annotations. Switching from precise to coarse annotations still incurs a performance drop when using *SnakeFast*, but a much smaller one than when using the baseline. Visual inspection of the resulting segmentations, shown in Fg. 11, leads us to conclude that, for tasks where a compromise

TABLE IV
PERFORMANCE OF *UNet* TRAINED USING *SnakeFast* ON THE *Brain* DATA SET WHEN VARYING THE ELASTICITY AND SPRING TERM COEFFICIENTS

		Pixel-wise			Topology-aware		iter. t.
		Corr.	Compl.	Qual.	APLS	TLTS	s
α =1e-2	β =1e-4	99.0	94.5	93.5	88.1	84.8	5.2
	β =1e-3	98.7	95.0	**93.8**	**91.1**	**85.9**	5.2
	β =1e-2	98.4	94.0	92.7	85.1	84.3	5.2
	β =1e-1	98.9	93.8	92.8	83.8	84.1	5.2
α =1e-4	β =1e-3	98.4	92.9	91.5	86.6	83.0	5.2
α =1e-3		99.0	94.2	93.4	85.3	84.4	5.2
α =1e-2		98.7	95.0	**93.8**	**91.1**	**85.9**	5.2
α =1e-1		98.7	94.3	93.1	79.8	82.5	5.2

TABLE V
PERFORMANCE OF *UNet* TRAINED USING *SnakeFast* ON THE *Brain* DATA SET WHEN VARYING THE INVERSE STEPSIZE, TOGETHER WITH THE NUMBER OF SNAKE UPDATES USED IN EVERY TRAINING ITERATION AND THE RESULTING ITERATION TIME

	Pixel-wise			Topology-aware		no steps	iter. t.
	Corr.	Compl.	Qual.	APLS	TLTS		s
$\gamma = 100$	98.8	94.5	93.4	90.9	85.8	80	6.3
$\gamma = 10$	98.7	95.0	**93.8**	**91.1**	**85.9**	10	5.2
$\gamma = 1$	— the snake diverged —					10	5.2

between accuracy and annotation cost is acceptable, using the easy annotations together with *SnakeFast* is a viable alternative to the classical approach.

I. Ablation Studies

To investigate the impact of hyper-parameters of our method on performance, we run the following ablation studies.

1) Regularization Terms: The regularization term R of Eq. 7 is the sum of a spring term, weighted by a coefficient α, and an elasticity term, weighted by a coefficient β. To investigate their influence on performance, we varied α and β and trained our *UNet* on the *Brain* data set. The results are presented in Tab. IV. The best results are attained with relatively low values of both terms. Higher values of the spring term, originally proposed for closed contours, effectively reguralize loopy topologies, but when used on tree-shaped structures, representing blood vessels and neuronal processes, tend to shorten the reconstructed neurites and vessels. Higher values of the elasticity term make it more difficult to fit irregular trajectories of neurites, like the ones shown in Fig. 5.

2) Step Size for Snake Update: As explained in section III-C, the snake update iteration has a parameter γ, called viscosity, that acts as an inverse step size. We report the results of changing γ in Tab. V. Low viscosity results in large step size and can make the snake update procedure diverge, which we observed for $\gamma = 1$. On the other hand, high viscosity corresponds to small step size and increases the risk that the snake does not converge within the preset number of iterations. With $\gamma = 100$, we needed to increase the number of snake updates from 10 to 80 to ensure convergence. This also increased the iteration time by one second. $\gamma = 10$ made the snake converge within 10 updates, while also resulting in marginally higher performance than $\gamma = 100$.

TABLE VI
PERFORMANCE OF DEEP NETS TRAINED WITH *L1* AND *L2* COSTS ON THE *Brain* DATA SET AND THE TIME NEEDED FOR SINGLE TRAINING ITERATION

		Pixel-wise			Topology-aware		iter. t.
Cost	Method	Corr.	Compl.	Qual.	APLS	TLTS	s
L1	*OrigAnnot*	98.6	91.2	90.1	81.4	80.5	2.8
	SnakeFast	98.8	94.6	**93.4**	**89.9**	**85.8**	5.2
L2	*OrigAnnot*	98.9	91.3	90.4	80.3	80.9	2.8
	SnakeFast	98.7	95.0	**93.8**	**91.1**	**85.9**	5.2

3) L1 vs L2 Distance: We also verified the performance of a *UNet* trained with *SnakeFast* when changing the loss data term from Mean Squared Error to Mean Absolute Error. The results, shown in Tab. VI show very slight advantage of MSE, possibly due to a gradient profile that prioritizes penalizing higher errors.

V. CONCLUSION AND FUTURE WORK

We have proposed a method that accounts for the inevitable inaccuracies in manual annotations of curvilinear 3D structures, such as neurites and blood vessels, in 3D image stacks. It leverages on the network snake formalism to define a loss function that simultaneously trains the deep network to produce the delineation and adjusts the initially imprecise annotations.

Our approach does not depend on the specific network architecture we use. Hence, its effectiveness suggests that handling such imprecisions may be even more important than refining the network architecture, which is something that has been largely neglected in the literature.

In future work, we will investigate the extension our approach to segmenting surfaces, like cell membranes in electron microscopy scans.

REFERENCES

[1] D. Acuna, A. Kar, and S. Fidler, "Devil is in the edges: Learning semantic boundaries from noisy annotations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11075–11083.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[3] D. Breitenreicher, M. Sofka, S. Britzen, and S. K. Zhou, "Hierarchical discriminative framework for detecting tubular structures in 3D images," in *Proc. Int. Conf. Med. Image Comput.-Assist. Intervent.*, 2013, pp. 328–340.

[4] E. Bullitt *et al.*, "Vessel tortuosity and brain tumor malignancy: A blinded study1," *Academic Radiol.*, vol. 12, no. 10, pp. 1232–1240, Oct. 2005.

[5] M. Butenuth and C. Heipke, "Network snakes: Graph-based object delineation with active contour models," *Mach. Vis. Appl.*, vol. 23, no. 1, pp. 91–109, Jan. 2012.

[6] S. Chatterjee *et al.*, "DS6, deformation-aware semi-supervised learning: Application to small vessel segmentation with noisy training data," 2020, *arXiv:2006.10802*.

[7] D. Cheng, R. Liao, S. Fidler, and R. Urtasun, "DARNet: Deep active ray network for building segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7431–7439.

[8] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 6, pp. 681–685, Jun. 2001.

[9] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2051–2060.

[10] A. F. Frangi, J. W. Niessen, and L. K. Vincken, "Multiscale vessel enhancement filtering," *Med. Image Comput.-Assist. Intervent.*, vol. 1496, pp. 130–137, Apr. 1998.

[11] P. Fua, "Model-based optimization: Accurate and consistent site modeling," in *Proc. 18th ISPRS Congr. Tech. Commission III, Theory Algorithms*, Vienna, Austria, vol. 31, K. Kraus and P. Waldhäusl, Eds., Jul. 1996. [Online]. Available: https://www.isprs.org/proceedings/XXXI/congress/part3/

[12] P. Fua and Y. G. Leclerc, "Model driven edge detection," *Mach. Vis. Appl.*, vol. 3, no. 1, pp. 45–56, Dec. 1990.

[13] Y. Ganin and V. Lempitsky, "N^4-Fields: Neural network nearest neighbor fields," in *Proc. Asian Conf. Comput. Vis.*, 2014, pp. 536–551.

[14] G. Hamarneh and P. Jassi, "VascuSynth: Simulating vascular trees for generating volumetric image data with ground-truth segmentation and tree analysis," *Comput. Med. Imag. Graph.*, vol. 34, no. 8, pp. 605–616, Dec. 2010.

[15] A. Hatamizadeh, D. Sengupta, and D. Terzopoulos, "End-to-end trainable deep active contour models for automated image segmentation: Delineating buildings in aerial imagery," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 730–746.

[16] X. Huang and L. Zhang, "Road centreline extraction from high-resolution imagery based on multiscale structural features and support vector machines," *Int. J. Remote Sens.*, vol. 30, no. 8, pp. 1977–1987, 2009.

[17] P. Jassi and G. Hamarneh, "VascuSynth: Vascular tree synthesis software," *Insight J.*, Apr. 2011, doi: 10.54294/j0ws9u.

[18] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vis.*, vol. 1, no. 1, pp. 321–331, Jan. 1988.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimisation," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[20] M. Koziński, A. Mosinska, M. Salzmann, and P. Fua, "Tracing in 2D to reduce the annotation effort for 3D deep delineation of linear structures," *Med. Image Anal.*, vol. 60, Feb. 2020, Art. no. 101590.

[21] M. Law and A. Chung, "Three dimensional curvilinear structure detection using optimally oriented flux," in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 368–382.

[22] K. K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool, "Deep retinal image understanding," in *Proc. Int. Conf. Med. Image Comput.-Assist. Intervent.*, 2016, pp. 140–148.

[23] L. Zhang *et al.*, "Learning deep structured active contours end-to-end," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8877–8885.

[24] G. Mattyus, W. Luo, and R. Urtasun, "DeepRoadMapper: Extracting road topology from aerial images," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 3458–3466.

[25] S. Min, X. Chen, Z. Zha, F. Wu, and Y. Zhang, "A two-stream mutual attention network for semi-supervised biomedical segmentation with noisy labels," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4578–4585.

[26] V. Mnih, "Machine learning for aerial image labeling," Ph.D. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2013.

[27] V. Mnih and G. E. Hinton, "Learning to detect roads in high-resolution aerial images," in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 210–223.

[28] A. Mosinska, M. Kozinski, and P. Fua, "Joint segmentation and path classification of curvilinear structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 6, pp. 1515–1521, Jun. 2020.

[29] A. Mosinska, P. Marquez-Neila, M. Kozinski, and P. Fua, "Beyond the pixel-wise loss for topology-aware delineation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 3136–3145.

[30] D. Oner, M. Kozinski, L. Citraro, N. C. Dadap, A. G. Konings, and P. Fua, "Promoting connectivity of network-like structures by enforcing region separation," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Apr. 21, 2021, doi: 10.1109/TPAMI.2021.3074366.

[31] H. Peng *et al.*, "Virtual finger boosts three-dimensional imaging and microsurgery as well as terabyte volume image visualization and analysis," *Nature Commun.*, vol. 5, no. 1, pp. 4342–4355, Sep. 2014.

[32] H. Peng, Z. Zhou, E. Meijering, T. Zhao, G. A. Ascoli, and M. Hawrylycz, "Automatic tracing of ultra-volumes of neuronal images," *Nature Methods*, vol. 14, pp. 332–333, Mar. 2017.

[33] K. Ramnath, S. Baker, I. Matthews, and D. Ramanan, "Increasing the density of active appearance models," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2008, pp. 1–8.

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput.-Assist. Intervent.*, 2015, pp. 234–241.

[35] M. Seyedhosseini, M. Sajjadi, and T. Tasdizen, "Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2013, pp. 2168–2175.

[36] A. Sironi, E. Türetken, V. Lepetit, and P. Fua, "Multiscale centerline detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 7, pp. 1327–1341, Jul. 2016.

[37] D. Terzopoulos, A. Witkin, and M. Kass, "Constraints on deformable models: Recovering 3D shape and nonrigid motion," *Artif. Intell.*, vol. 36, no. 1, pp. 91–123, Aug. 1988.

[38] E. Turetken, C. Becker, P. Glowacki, F. Benmansour, and P. Fua, "Detecting irregular curvilinear structures in gray scale and color imagery using multi-directional oriented flux," in *Proc. Int. Conf. Comput. Vis.*, Dec. 2013, pp. 1553–1560.

[39] A. Van Etten, D. Lindenbaum, and T. M. Bacastow, "SpaceNet: A remote sensing dataset and challenge series," 2019, *axiv:1807.01232*. [Online]. Available: https://arxiv.org/pdf/1807.01232.pdf

[40] G. Wang *et al.*, "A noise-robust framework for automatic segmentation of COVID-19 pneumonia lesions from CT images," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2653–2663, Aug. 2020.

[41] W. Wang, K. Yu, J. Hugonot, P. Fua, and M. Salzmann, "Recurrent U-Net for resource-constrained segmentation," in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 1–10.

[42] J. D. Wegner, J. A. Montoya-Zegarra, and K. Schindler, "A higher-order CRF model for road network extraction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2013, pp. 1698–1705.

[43] C. Wiedemann, C. Heipke, H. Mayer, and O. Jamet, "Empirical evaluation of automatically extracted road axes," *Empirical Eval. Techn. Comput. Vis.*, vol. 12, pp. 172–187, Jun. 1998.

[44] D. Wu *et al.*, "A learning based deformable template matching method for automatic rib centerline extraction and labeling in CT images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 980–987.

[45] Z. Yu *et al.*, "Simultaneous edge alignment and learning," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2018, pp. 388–404.

[46] Z. Zhang, D. Marin, E. Chesakov, M. M. Maza, M. Drangova, and Y. Boykov, "Divergence prior and vessel-tree reconstruction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10216–10224.

[47] Z. Zhou, X. Liu, B. Long, and H. Peng, "TReMAP: Automatic 3D neuron reconstruction based on tracing, reverse mapping and assembling of 2D projections," *Neuroinformatics*, vol. 14, no. 1, pp. 41–50, Jan. 2016.

[48] H. Zhu, J. Shi, and J. Wu, "Pick- and-learn: Automatic quality evaluation for noisy-labeled image segmentation," in *Proc. Int. Conf. Med. Image Comput.-Assist. Intervent.*, 2019, pp. 576–584.