A. Definition of Terms

We address the problem of unsupervised domain adaptation (UDA). This problem is closely related to supervised domain adaptation (SDA) and semi-supervised learning (SSL). In SSL problems both labeled and unlabeled training data is given, with the goal to learn a "better" model using all available data compared to using fully supervised learning on just the labeled data. Both labeled and unlabeled data are from the same domain (same data distribution). In the case of UDA a source domain with labels and a target domain without labels is given. Source and target domain have different data distributions, corresponding e.g. to different imaging devices or different experimental conditions in biomedical applications. The goal of UDA is to train a model that solves the same task (e.g. cell segmentation) on the target domain as on the source domain. The case of SDA is very similar, but the target domain is partially labeled (i.e. annotations are provided for a subset of the target samples). This discussion shows that all three settings are similar, with the distinction being the data distributions for labeled and unlabeled data: In SSL both labeled and unlabeled data come from the same data distribution and in UDA they come from two different data distributions (source and target data distribution). In SDA the labeled data comes from both source and target distribution (usually with the fraction of target data being significantly smaller) and the unlabeled data comes only from the target distribution. Consequently self-training methods can be generalized to all three learning problems, as has recently been demonstrated by *AdaMatch* [4].

We make use of self-training with pseudo-labels to address UDA. These terms are sometimes used with slightly different meanings in the literature. Here, we use self-training to describe methods that use a version of the model being trained to generate predictions on unlabeled data, which are then used as targets in an unsupervised loss function to again train the model. This can be understood as a student-teacher set-up, with the teacher being a version of the student (e.g. through EMA of weights or weight sharing). We use the term pseudo-labeling to describe the process of transforming the teacher predictions into targets for the unsupervised loss function, e.g. by post-processing or filtering (masking or weighting) them. Note that the literature sometimes distinguish between pseudo-labeling when the likeliest prediction is used as hard target in the unsupervised loss, and consistency regularization when the softmax output (more generally output after the last activation) is used as target. See for example https://lilianweng.github.io/posts/2021-12-05-semi-supervised/ for an in-depth discussion. However, this distinction is minor in practice and can be incorporated in the pseudo-labeling post-processing in our formulation (the function p in Eq. 1). Hence, we do not make this distinction throughout the paper.

B. Probabilistic Domain Adaptation: Training Strategies

We implement two different approaches to domain adaptation: *joint* training (model is trained jointly on labeled source and unlabeled target data, using a supervised and unsupervised loss function) and *separate* training (model is first pre-trained on the labeled source data, using only the supervised loss, and then fine-tuned on the unlabeled target data, using only the unsupervised loss). Here, we show pseudo-code for the two training routines, for *joint* training in Alg. 1 and for *separate* training in Alg. 2. For simplicity we omit validation, which is performed using the target data in both cases. Here, we use the same loss function l for both the supervised and unsupervised loss.

Input: Labeled training data $\{(x_s, y_s)\}$, unlabeled training data $\{x_u\}$, teacher and student models s, t, number of iterations N
Initialize parameters of s and t;
for $i \leftarrow 1$ *to* N **do**
  Sample mini-batch (x_s^i, y_s^i) and x_u^i;
  Compute supervised loss $L_s = l(s(x_s^i), y_s^i)$;
  Sample augmentations τ_s and τ_t;
  Compute pseudo-labels $\hat{y} = t(\tau_t(x_u))$;
  Compute unsupervised loss $L_u = l(f(s(\tau_s(x_s)), \hat{y}))$;
  Compute gradients, update parameters of s based on $L_s + L_u$;
  Update parameters of t from s;
end

Algorithm 1: Pseudo code for the joint training strategy.

The two self-training approaches we implement, *MeanTeacher* and *AdaMatch* correspond to different choices for the teacher and student augmentations τ_s and τ_t as well as the teacher update scheme. For *MeanTeacher* both τ_s and τ_t are sampled from a distribution of weak augmentations and the weights of t are the EMA of s. For *FixMatch* τ_s is sampled from a distribution of strong augmentations and τ_t from a distribution of strong augmentations, s and t share weights. The different pseudo-label filtering approaches are realized by different choices for f, where *consensus masking* corresponds to only computing gradients for pixels that have a value of 1 in the consensus response (see Eq. 3), *consensus weighting* corresponds to weighting the loss by the consensus response. In the case of no filtering f is the identity.

Input: Unlabeled training data $\{x_u\}$, pre-trained model s, number of iterations N
Copy model t from s;
for $i \leftarrow 1$ *to* N **do**
  Sample mini-batch x_u^i;
  Sample augmentations τ_s and τ_t;
  Compute pseudo-labels $\hat{y} = t(\tau_t(x_u))$;
  Compute unsupervised loss $L_u = l(f(s(\tau_s(x_s)), \hat{y}))$;
  Compute gradients, update parameters of s based on L_u;
  Update parameters of t from s;
end

Algorithm 2: Pseudo code for the separate training strategy. (Only the adaptation stage on the target domain; source training follows regular supervised learning.)

C. Implementation

We use the same UNet and PUNet architecture for all experiments, using an encoder-decoder architecture following the respective implementations of [17] and [11]. We increase the number of channels from 64 to 128, 256 and 512 in the encoder, and decrease it accordingly in the decoder. We use a 2d segmentation network, hence both architectures make use of 2d convolutions, 2d max-pooling and 2d upsampling operations. The UNet is trained using the Dice Error (1. - Dice Score) as loss function. For the PUNet we use a similar formulation for the loss function as in [11], but use the Dice Error for the reconstruction term instead of the cross entropy. We use a dimension of 6 for the latent space predicted by the prior and posterior net of the PUNet. We use the Adam optimizer, relying on the default PyTorch parameter settings, except for the learning rate, and we use the *ReduceLROnPlateau* learning rate scheduler. For *joint* and source model trainings we train for 100k iteration, for the second stage of *separate* trainings we train for 10k iterations. We use different patch shapes, batch sizes and learning rates depending on the dataset and method; these values were determined by exploratory experiments.

For LIVECell:

- $UNet$: patch shape: (256, 256); batch size: 4; learning rate: 1e-4
- $PUNet$: patch shape: (512, 512); batch size: 4; learning rate: 1e-5
- $PUNet_{trg}$, MT_s: patch shape: (512, 512); batch size: 2; learning rate: 1e-5
- FM_s: patch shape: (256, 256); batch size: 2; learning rate: 1e-7
- FM_j, MT_j: patch shape: (256, 256); batch size: 2; learning rate: 1e-5

For mitochondria segmentation in EM:

- $UNet$, $PUNet$, MT_s, MT_j, FM_j: patch shape: (512, 512); batch size: 4; learning rate: 1e-5
- FM_s: patch shape: (512, 512); batch size: 4; learning rate: 1e-7

For lung segmentation in X-Ray:

- $UNet$: patch shape: (256, 256); batch size: 2; learning rate: 1e-4
- $PUNet$, MT_s, MT_j: patch shape: (256, 256); batch size: 2; learning rate: 1e-5

We use gaussian blurring and additive gaussian noise (applied randomly with a probability of 0.25, and with augmentations parameters also sampled from a distribution) as weak augmentations, and gaussian blurring, additive gaussian noise and random contrast adjustments (applied randomly with a probability of 0.5, and sampling from a wider range compared to the weak augmentations) as strong augmentations.

Our implementation is based on PyTorch. We use the PUNet implementation from https://github.com/stefanknegt/Probabilistic-Unet-Pytorch. All our code is available on GitHub at https://github.com/computational-cell-analytics/Probabilistic-Domain-Adaptation. Please refer to the README for instructions on how to run and install it.

D. Datasets

LIVECell Dataset We use the LIVECell dataset from [6]. This dataset contains about 5000 phase contrast microscopy images with instance segmentation ground-truth and predefined train-, test-, and validation-splits. We binarize the instance segmentation ground-truth to obtain a semantic segmentation problem. The dataset contains images of 8 different cell lines: A172, BT474, BV2, Huh7, MCF7, SHSY5Y, SkBr3 and SKOV3. These cell lines show significant difference in appearance and morphology of cells as well as spatial distribution such as cell density and cell clustering. Hence, we treat all 8 cell types as different domains, and study the adaptation from one cell line as source domain to the seven other target domains for all 8 cell lines. The columns in Tab. 2 show the average dice score for one source applied to the seven target domains.

Mitochondria EM Segmentation For mitochondria segmentation in EM we use the dataset of [7] as source dataset. This dataset contains two EM volumes, one of human neural tissue, the other of rat neural tissue, imaged with scanning EM. Each volume contains 400 images with instance annotations for training, and 100 images with instance annotations for testing. We binarize the instance segmentation ground-truth to obtain a semantic segmentation problem. We study domain adaptation with [7] as source for

two different target datasets. The first is *Lucchi* [5], which contains two volumes of tissue from the murine hippocampus imaged with FIBSEM that both contain mitochondria instance annotations. We use one of the volumes for training the domain adaptation methods (either via joint or separate training), and the other for evaluation. And *VNC* [8], which contains two volumes from the ventral nerve cord of a fruit fly, imaged with serial section transmission EM. Only one of the two volumes contains instance annotation, it is used for evaluation, the other does not, it is used for training the domain adaptation methods (which does not require labels). *UroCell* [33] contains annotated volumes of tissue from mice urinary bladders imaged with FIBSEM that contain multiple intracellular compartment annotations, including mitochondria. We use four volumes for training the domain adaptation methods (either via joint or separate training), and reserve one volume for evaluation.

Lung X-Ray Segmentation For the lung segmentation task we use four different datasets of chest radiographs, following the experiment set-up of [23]. The datasets are: *NIH*, which contains chest X-Ray (CXR) images with various severity of lung diseases, *Montgomery* [9], which contains images of patients with and without tuberculosis, and *JSRT* [20], which contains images of patients with and without lung nodules. The *JSRT* dataset is split into two subsets: *JSRT1* with normal CXR images (60 images, 50 train, 10 test), and *JSRT2* with inverted CXR images (247 images, 199 train, 48 test). The *NIH* dataset contains 100 images (we use 80 for training and 20 for testing) and the *Montgomery* dataset contains 138 images (113 are used training, 25 for testing) respectively. All datasets contain binary lung annotations; we discard additional annotations in the case of *JSRT2*. We treat each dataset as a separate domain, and perform domain adaptation for all pairs of domains.