Appendix for "Multiple Instance Learning via Iterative Self-Paced Supervised Contrastive Learning"

The appendix is organized as follows:

- In Appendix A, we include additional descriptions of the datasets (Appendix A.1), implementation details (Appendix A.2) and instructions on how to transform the Camelyon16 dataset for the additional experiments (Appendix A.4). In Appendix A.3, we describe the hyperparameter selection process and report results from an ablation study on the Camelyon16 dataset to evaluate the sensitivity of our approach to the choice of hyperparameters. In Appendix A.5, we provide a detailed description of the ablated versions of ItS2CLR and CE finetuning from Section 4.3.
- In Appendix B, we include additional results. In Appendix B.1, we report pseudo label accuracy measured by additional metrics between ItS2CLR and the ablated versions. In Appendix B.2, we show additional results for instance-level performance. In Appendix B.3, we report additional comparisons with end-to-end methods. In Appendix B.4, we provide additional examples of tumor localization maps.
- In Appendix C, we provide the formulation and implementation details for the different MIL aggregators used in our study.

A. Experiments

A.1. Dataset

Camelyon16 Camelyon16 is a public dataset for detection of metastasis in breast cancer. This dataset consists of 271 training and 129 test whole slide images (WSI). All the images (including both training and test) are divided into 0.25 million patches at 5× magnification. On average, each slide contains approximately 625 patches 5× magnification respectively. Each WSI is paired with pixel-level annotations indicating the position of tumors (if any are present). We ignore the pixel-level annotations during training and consider only slide-level labels (i.e. the slide is considered positive if it contains any annotated tumor regions). As a result, positive bags contain patches with tumors and patches with healthy tissue. Negative bags contain only patches with healthy tissue. The ratio between positive and negative patches in this dataset is highly imbalanced. Only a small fraction of patches in the positive slides contain tumors (less than 10%).

TCGA-LUAD TCGA for Lung Adenocarcinoma (LUAD) is a subset of TCGA (The Cancer Genome Atlas), a landmark cancer genomics program. It consists of 800 tumorous frozen whole-slide histopathology images and the corresponding genetic mutation status. Each WSI is paired with a single binary label indicating whether each gene is mutated or wild type. In this experiment, we build MIL models to detect four mutations - EGFR, KRAS, STK11, and TP53, which are sensitizing mutations that can impact treatment options in LUAD [16, 23]. We split the data set randomly into training, validation and test sets so that each patient will appear in only one of the subsets. After splitting the data, 477 images are in the training set, 96 images are in the validation set, and 227 images are in the test set.

Breast Ultrasound dataset The Breast Ultrasound Dataset includes 28,914 ultrasound exams [45]. An exam is labeled as cancer-positive if there is a pathology-confirmed malignant finding associated with this exam. In this dataset, 5,593 exams are cancer-positive. On average, each exam contains approximately 18 images. Patients in the dataset were randomly divided into a training set (60%), a validation set (10%), and test set (30%). Each patient was included in only one of the three sets. We show 5 example breast ultrasound images in Figure 4.

A.2. Implementation Details

All experiments were conducted on NVIDIA RTX8000 GPUs and NVIDIA V100 GPUs. For all models, we perform model selection during training based on bag-level AUC evaluated on the validation set.

Camelyon16 We follow the same preprocessing and pretraining steps as [35]. To preprocess the slides, we cropped the slides into tiles at 5x magnification, filtered out tiles that do not contain enough tissues (average saturation < 30), and resized the images to a resolution of 224×224 pixels. Resizing was performed using the Pillow package [15] with default settings (nearest neighbor sampling).

We pretrain the feature extractor, ResNet18 [28], with SimCLR [10] for a maximum of 600 epochs, at which we notice that the training loss converges. Each patch is represented by a 512-dimensional vector. To evaluate the learned representation

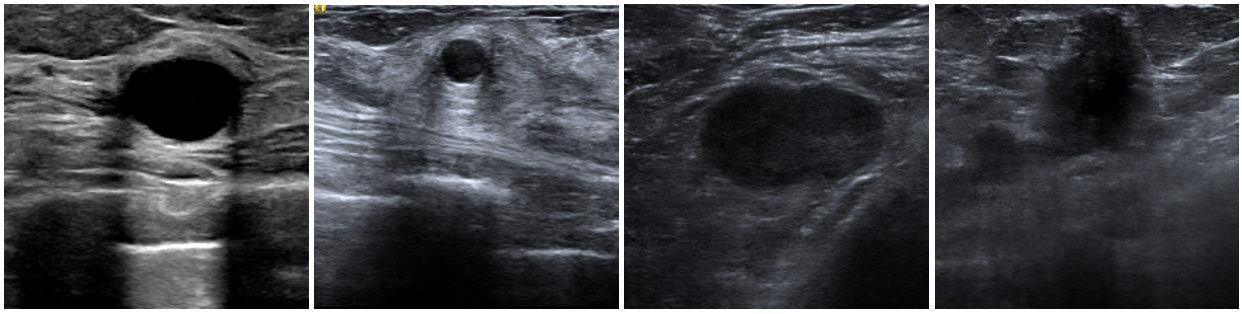

Figure 4. Example breast ultrasound images. The first two images contain a benign lesion. The second and third contain a malignant lesion. In all ultrasound images, the center object of the circular shape corresponds to the lesion of interest. The images are from different exams.

on the down-stream task, we train a MIL aggregator based on instances representations and evaluate the bag-level prediction every few epochs. We observe that the bag-level AUC on the downstream task in the validation set does not improve when pretraining for a longer period. We set the batch size to 512 and temperature to 0.5. We use SGD with the learning rate of 0.03, weight decay of 0.0001, and cosine annealing scheduler.

During finetuning the feature extractor with Its2CLR, we finetune the feature extractor for a maximum of 50 epochs. We choose the model that achieves the highest bag-level AUC on the downstream task in the validation set. The batch size is set to 512, and the learning rate is set to 10^{-2}. At the feature extractor training stage, we apply random data augmentation to each instance, including:

- random ($p = 0.8$) color jittering: brightness, contrast, and saturation factors are uniformly sampled from $[0.2, 1.8]$, hue factor is uniformly sampled from $[-0.2, 0.2]$.,
- random grayscale ($p = 0.2$),
- random Gaussian blur with a kernel size of 0.06 times the size of an image,
- random horizontal/vertical flipping with 0.5 probability.

When training the DS-MIL aggregator, we follow the settings in [35]. We use the Adam optimizer during training. Since each bag may contain a different number of instances, we follow [35] and set the batch size to just one bag. We train each model for a maximum of 350 epochs. We use an initial learning rate of 2×10^{-4}, and use the StepLR scheduler to reduce the learning rate by 0.5 every 75 epochs. Details on the hyperparameters used for training the aggregator are in Appendix C.

TCGA-LUAD To preprocess the slides, we cropped them into tiles at 10x magnification, filtered out the background tiles that do not contain enough tissues (when the average saturation is less than 30), and resized the images into the resolution of 224×224 pixels. Resizing was performed using the Pillow package [15] with nearest neighbor sampling. These tiles were color-normalized with the Vahadane method [47].

To train the feature extractor, we perform the same process as for Camelyon16.

We also use DS-MIL [35] as the aggregator. When training the aggregator, we resample the ratio of positive and negative bags to keep the class ratio balanced. We train the aggregator for a maximum of 100 epochs using the Adam optimizer with the learning rate set to 2×10^{-4} and divide the learning rate by 2 every 50 epochs.

Breast Ultrasound We follow the same preprocessing steps as [45]. All images were resized to 224×224 pixels using bilinear interpolation. We used ResNet18 [28] as the feature extractor and pretrained it using SimCLR [10] for 100 epochs, at which we notice that the training loss converges. We adopt the same pretraining and model selection approach as for Camelyon16. We used the Instance Attention-MIL as an aggregator [29]. Given a bag of images $x_1, ..., x_k$ and a feature extractor f, the aggregator first computes instance-level predictions $\hat{y}_i$ for each image x_i. It then calculates an attention score $\alpha_i \in [0, 1]$ for each image x_i using its feature vectors $f(x_i)$. Lastly, the bag-level prediction is computed as the average instance prediction weighted by the attention score $\hat{y} = \sum_i^k \alpha_i \hat{y}_i$. To optimize the aggregator, we trained it using Adam with a learning rate set to 10^{-3} for a maximum of 350 epochs. The model is selected according to the best bag-level AUC on the validation set.

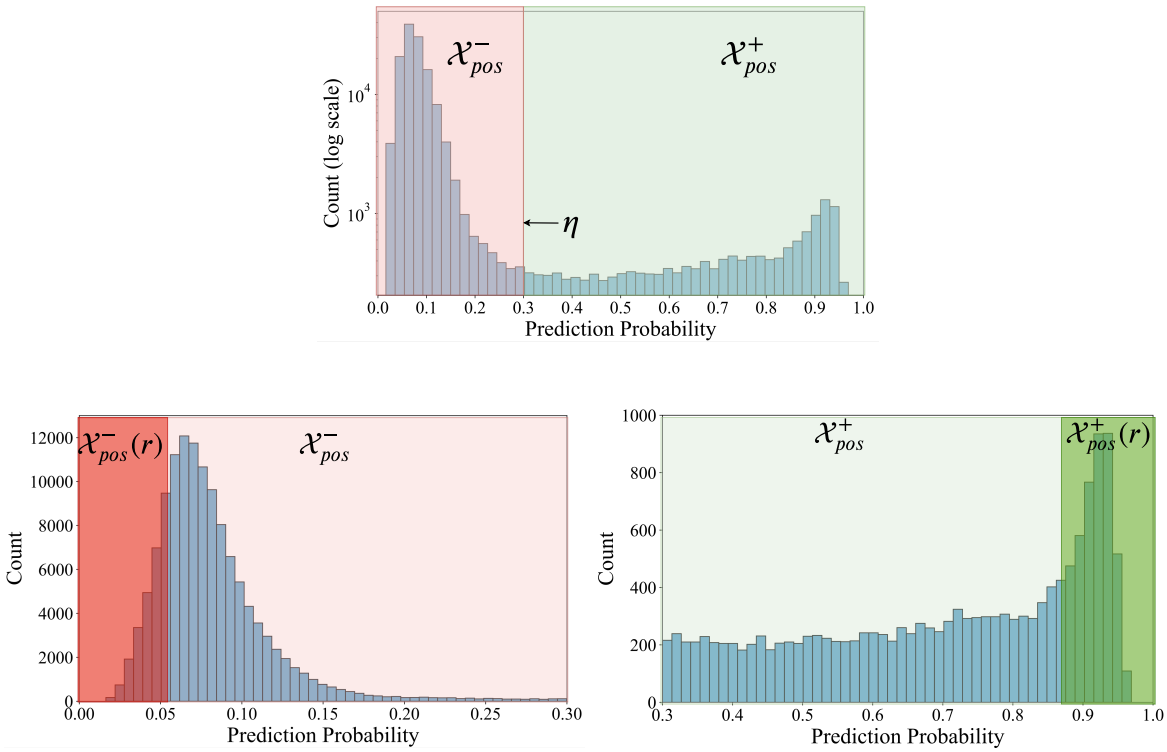

Figure 5. Illustration of our partitioning of the instances from positive bags in Section 3.2 based on the predicted probability of the instance classifier in ItS2CLR. **Top**: $\mathcal{X}^+_{\text{pos}}$ and $\mathcal{X}^-_{\text{pos}}$ are partitioned according to the thresholding parameter η. **Bottom**: The distribution of instance scores for instances with negative pseudo labels (left) and negative pseudo labels (right). A threshold r is symmetrically applied on both distributions so that the top $r\%$ instances with the lowest and highest scores are treated as confidently negative or positive, respectively. We use $\mathcal{X}^-_{\text{pos}}(r)$ and $\mathcal{X}^+_{\text{pos}}(r)$ to denote the set of instances that are deemed truly negative and positive respectively. During training, as the accuracy of the pseudo labels improves, we can increase r to incorporate more samples in these sets.

A.3. Hyperparameters for Training the Feature Extractor in ItS2CLR

Hyperparameter tuning The hyperparameters of the proposed method include: the learning rate $lr \in [1\times10^{-5}, 1\times10^{-3}]$, the initial threshold used for binarization of the prediction to produce pseudo labels $\eta \in [0.1, 0.9]$, the proportion of sampled positive anchors $p_+ \in [0.05, 0.5]$, the initial value $r_0 \in [0.01, 0.7]$ and the terminal value $r_T \in [0.2, 0.8]$ of the percentage of selected instances r in the self-paced sampling scheme. For Camelyon16, we obtain the highest bag-level validation AUC using the following hyperparameters: $\eta = 0.3$, $p_+ = 0.2$, $r_0 = 0.2$ and $r_T = 0.8$. We use the feature extractor trained under these settings in Tables 2, 3 and 4. The complete list of hyperparameters in our experiments is reported in Table 8.

Table 8. ItS2CLR hyperparameters used in our experiments.

	Camelyon16	Breast Ultrasound	TCGA-LUAD mutation			
			EGFR	KRAS	STK11	TP53
η	0.3	0.3	0.3	0.5	0.3	0.5
r_0	0.2	0.2	0.2	0.2	0.2	0.2
r_T	0.8	0.8	0.8	0.8	0.8	0.8
p_+	0.2	0.2	0.5	0.2	0.2	0.2

Sensitivity analysis We conduct a sensitivity analysis for each hyperparameter on Camelyon 16, and observe a robust performance over a range of hyperparameter values.

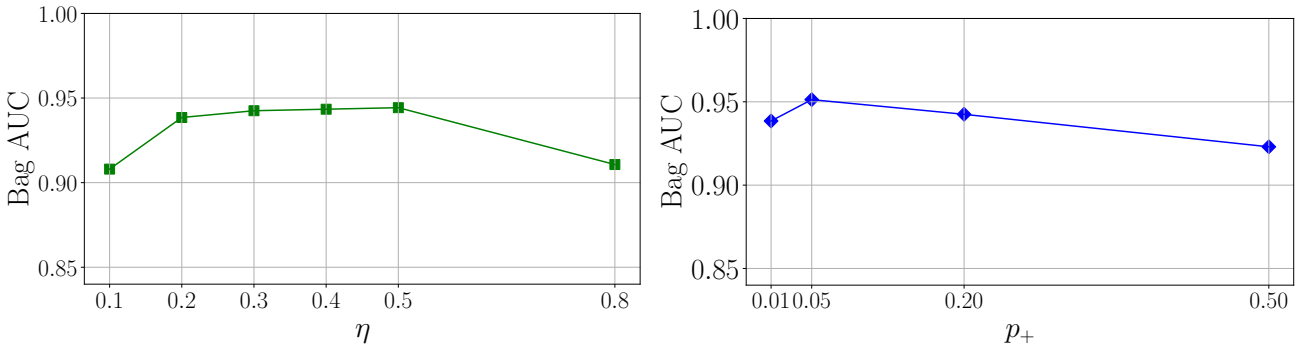

Figure 6. Sensitivity analysis for the threshold η and the ratio of positive pseudo labels used as anchor images p_+ on the Camelyon16 dataset.

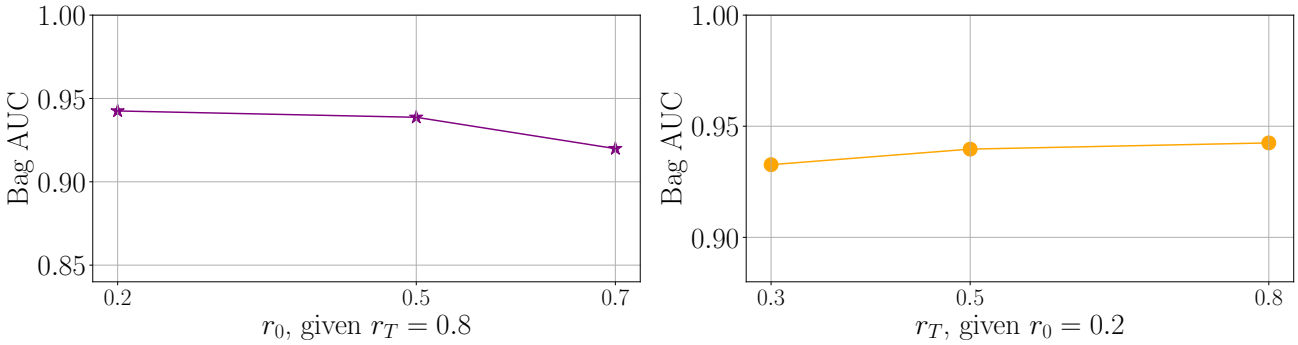

Figure 7. Sensitivity analysis for the hyperparameters r_0 and r_T of the proposed self-paced learning scheme on Camelyon16.

- *Threshold* η: The choice of η influences the instance-level pseudo labels. As shown in Figure 5, the outputs of the instance-level classifier are mostly close to 0 or 1, so the pseudo labels do not dramatically vary for a wide range of η. We analyze the importance of η. The left panel of Figure 6 shows that ItS2CLR is quite robust to the value of η, except for extreme values. If η is too small (e.g. 0.1), it can introduce a significant number of false positives. If η is too large (e.g. 0.8), it can mistakenly exclude some useful positive samples, causing a drop in the performance. In the main paper, since negative instances are more prevalent than positive instances, a threshold of 0.3 (less than 0.5) can increase the recall for the positive instances.

- *Sampling ratio of anchor instance over pseudo labels*: We use p_+ to denote the percentage of positive anchor instances sampled during the contrastive learning stage. The right panel of Figure 6 shows that it is desirable to choose a relatively small p_+. Since there are far fewer positive instances than negative instance, keeping the ratio of positive anchors low can avoid repetitively sampling from a limited number of positive instances. Also, since the instance pseudo labels $\mathcal{X}^-_{\text{neg}}$ must be correct by the definition of negative bags, the negative instance pseudo labels are more accurate than the positive ones.

- *The initial rate* r_0 *and final rate* r_T *for the self-paced sampling scheduler*: Figure 7 shows that ItS2CLR is also generally robust to the values of r_0 and r_T. However, an extremely large initial rate r_0 (high confidence in the pseudo labels) may introduce more samples with incorrect labels during training and hurt the performance. Conversely, extremely small r_T (low confidence in the pseudo labels) may hinder the model from using more data, also hurting performance.

- *Sampling during warm-up:* During the warmup phase, we sample anchor instances from $\mathcal{X}^-_{\text{neg}}$. An alternative choice can be sampling the anchor instance from $\mathcal{X}^+_{\text{pos}}$ and the corresponding set $\mathcal{D}_x$ from $\mathcal{X}^-_{\text{neg}}$. However, our experiments show that the resulting bag-level AUC drops to 90.91 under this setting, which is significantly lower than 94.25 by the proposed method. This comparison demonstrates the importance of using clean negative instances as anchor images during warmup.

A.4. Experiments on Synthetic Versions of Camelyon16

Simulation of witness rates (WR)

Since the ground truth instance-level labels are available for Camelyon16, we can conduct controlled experiments on synthetic versions of the dataset. We manipulate the prevalence of positive instances in the bag (the *witness rate*) and study its impact on the performance of the proposed approach and the baselines, as reported in Section 2. To increase the witness rate, we randomly drop the negative instances at a fixed ratio in each bag; to reduce the witness rate, we randomly drop the positive instances at a fixed ratio in each bag. The percentage of retained instances and the resulting witness rates are reported in Table 6.

Downsampled version of Camelyon16 for end-to-end training

In order to enable end-to-end training, we downsample each bag in Camelyon16 to around 500 instances so that it fits in the memory of a GPU. To achieve this, we divide large bags which have more than 500 instances into smaller bags.

For negative bags, we randomly partition the instances within the original bag into same-sized sub-bags with around 500 instances.

For positive bags, we randomly partition the positive instances within the original bag into the desired number of sub-bags. We adjust the number of sub-bags so that it cannot be less than the number of positive instances. We then combine the positive instances and the negative instances to form sub-bags. This ensures that the bag-level label is correct and the witness rate for each positive sub-bag remains similar to the original bag.

A.5. Description of the ablation study

Details for CE finetuning with/without iterative updating

- *CE without iterative updating:* we use the same initial pseudo labels and pretrained representations as our ItS2CLR framework. Concretely, we label all the instances in negative bags as negative. We label the instances in positive bags using the instance prediction obtained from the aggregator. When finetuning the aggregator, the pseudo labels are kept fixed.
- *CE + iterative updating:* based on CE with iterative updating, the pseudo labels are updated every few epochs, which is in turn used to guide the finetuning of the feature extractors.

Details for ItS2CLR with/without SPL

- *ItS2CLR without iterative updating:* we keep everything the same as the full Its2CLR procedure (including the SPL strategy), but we do not apply steps 7, 8 and 9 in Algorithm 1. As a result, the pseudo labels are fixed to the initial set of pseudo labels.
- *ItS2CLR without SPL:* we keep everything the same as the full Its2CLR procedure (including iterative updating), but modify step 10 in Algorithm 1. We do not utilize the pseudo label to train the model in a self-paced learning way as in Section 3.2. We utilize all the pseudo-labeled data from the beginning of the finetuning.

B. Additional Results

B.1. Learning Curves

F1-Score plot corresponding to Figure 1: In Figure 8, we show the max F1 score curve corresponding to the right side of Figure 1. This plot confirms the importance of self-paced learning and iterative updating in ItS2CLR.

Instance-level AUC during training comparison with cross-entropy finetuning: Figure 9 compares ItS2CLR with an alternative approach that finetunes the feature extractor using cross-entropy (CE) loss on the Camelyon16 dataset. Without iterative updating, CE finetuning rapidly overfits to the incorrect labels. Iterative updating prevents this to some extent but does not match the performance of ItS2CLR, which produces increasingly accurate pseudo labels as the iterations proceed.

B.2. Instance-level Evaluation

In order to evaluate instance-level performance, we report values of classification metrics including AUC, F1-score, AUPRC and Dice score for localization.

The Dice score is defined as follows:

$$\text{Dice} = \frac{2\sum_i y_i p_i}{\sum_i y_i + \sum_i p_i}, \tag{4}$$

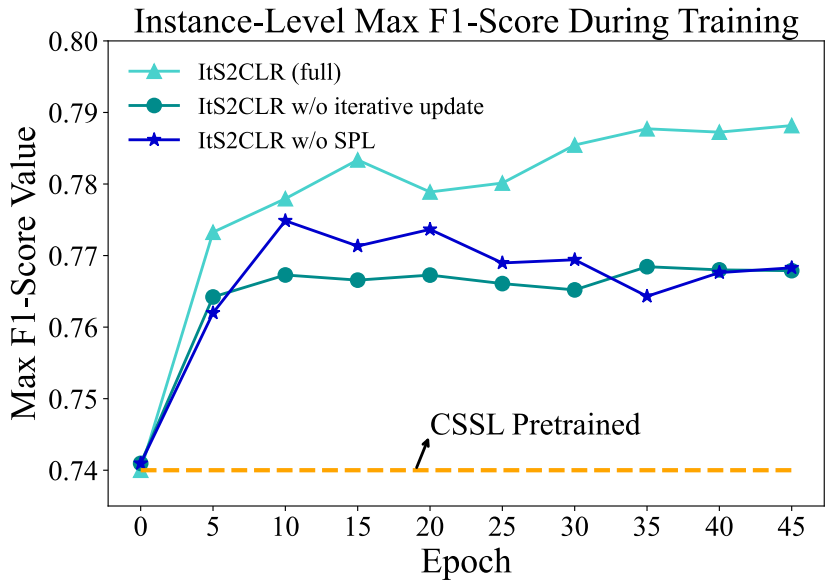

Figure 8. Comparison of max F1 score on instance pseudo labels. ItS2CLR updates the features iteratively based on a subset of the pseudo labels that are selected according to the self-paced learning (SPL) strategy. On Camelyon16, this gradually improves the accuracy of the pseudo labels in terms of instance-level max F1 score. Both the iterative updates and SPL are important to achieve this.

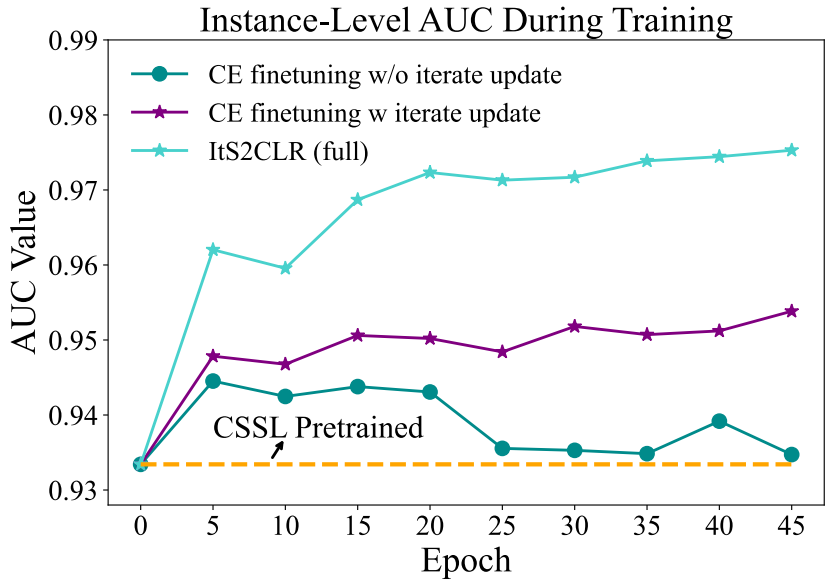

Figure 9. Comparison of instance-level AUC during training between ItS2CLR and an alternative approach that finetunes the feature extractor using cross-entropy (CE) loss on the Camelyon16 dataset. Iterative updating improves performance for CE finetuning, but ItS2CLR produces more accurate pseudo labels.

where y_i and p_i are the ground truth and predicted probability for the ith sample. The predicted probability is computed from the output of the MIL model s_i via linear scaling:

$$p_i = \sigma\left(as_i + b\right), \tag{5}$$

where $a \in [-5, 5]$ and $b \in [0.1, 10]$ are chosen to maximize the Dice score on the validation set.

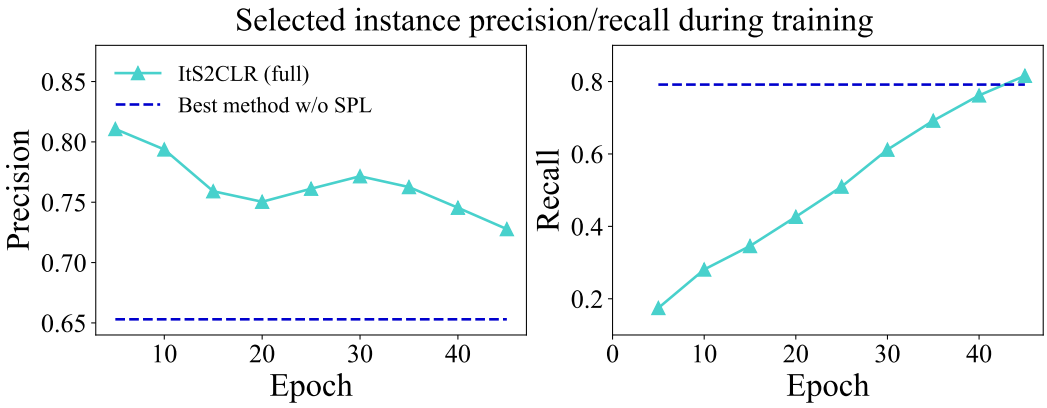

Figure 10. The precision and recall of pseudo labels from the selected instances during fine-tuning. The proposed ItS2CLR approach achieves high precision in generating pseudo labels from selected instances during fine-tuning through self-paced sampling. Since instance-level AUC improves during training (as shown in Fig 1), gradually including more instance candidates leads to higher recall while maintaining significant precision.

Table 9. Comparison of instance-level performance for the models in Table 3, using a max pooling aggregator.

$(\times 10^{-2})$	SimCLR (CSSL)	Ground-truth finetuning	CE finetuning		ItS2CLR			
			w/o iter.	*iter.*	*w/o both*	*w/o iter.*	*w/o SPL*	*Full*
AUC	91.53	97.58	93.17	94.48	92.69	94.55	94.43	**96.25**
F1-score	78.45	88.24	85.26	85.83	**86.77**	86.05	87.52	86.75
AUPRC	79.94	85.50	85.79	86.73	85.50	84.54	86.80	**89.99**
Dice	31.21	63.01	43.90	44.76	46.57	55.30	52.55	**57.82**

Table 10. Comparison of instance-level performance for the models in Table 3, using a linear classifier trained on the frozen features produced by each model. In addition, we produce bag-level predictions using the maximum output of the linear classifier for each bag.

$(\times 10^{-2})$ Instance-level	SimCLR (CSSL)	Ground-truth finetuning	CE finetuning		ItS2CLR			
			w/o iter.	*iter.*	*w/o both*	*w/o iter.*	*w/o SPL*	*Full*
AUC	96.13	97.56	96.94	96.88	96.64	97.25	96.92	**97.27**
F1-score	85.29	87.69	87.34	86.94	86.00	87.07	87.6	**87.92**
AUPRC	82.65	85.94	79.96	78.02	78.17	**84.56**	77.90	82.09
Dice	49.56	61.39	55.40	54.66	51.85	54.87	55.11	**60.13**
Bag-level (max-pooling)								
AUC	86.25	97.37	87.53	90.54	89.97	93.09	92.81	**97.47**

Max pooling aggregator: In Table 4, we show that our model achieves better weakly supervised localization performance compared to other methods when DS-MIL is used as the aggregator. In Table 9, we show that the same conclusion holds for an aggregator based on max-pooling.

Linear evaluation: In Table 10, we report results obtained by training a logistic regression model using the features obtained from the same approaches in Table 4, following a standard linear evaluation pipeline in representation learning [10]. ItS2CLR again achieves the best instance-level performance. We also produce bag-level predictions using the maximum output of the linear classifier for each bag, which again showcases that better instance-level performance results in superior bag-level classification.

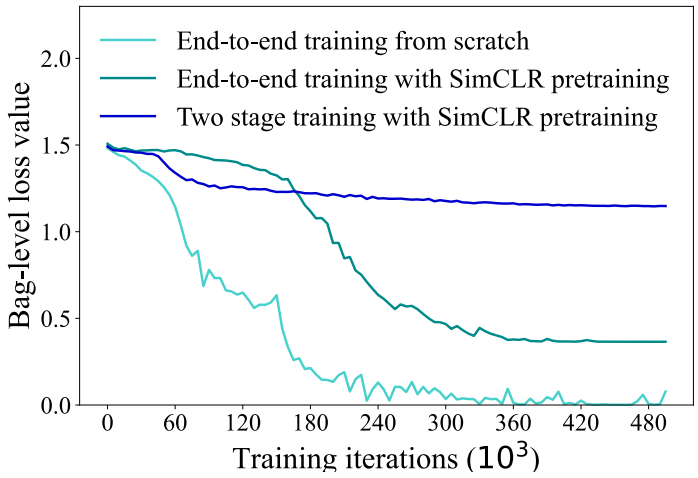

Figure 11. Comparison between end-to-end training and two-stage training on the downsampled version of the Camelyon16 dataset. End-to-end models overfit rapidly. Note that the unit of the training iterations here is 1k.

Table 11. Results on the downsampled version of the Camelyon16 dataset.

	End-to-end (scratch)	End-to-end (SimCLR)	SimCLR + DS-MIL	ItS2CLR
Bag AUC	64.52	66.71	80.96	**88.65**
Instance AUC	78.32	81.29	93.94	**95.58**
Instance F-score	51.02	55.71	85.93	**87.01**

Table 12. Results on the Breast Ultrasound dataset.

	SimCLR + Aggregator	End-to-end MIL	ItS2CLR
Bag AUC	80.79	91.26	**93.93**
Bag AUPRC	34.63	58.73	**70.30**
Instance AUC	62.83	82.11	**88.63**
Instance AUPRC	10.58	31.31	**43.71**

B.3. Comparison with End-to-end Training

In this section, we provide additional results to complement Table 5, where ItS2CLR is compared to end-to-end models. The end-to-end training is conducted with the same aggregators for each dataset as described in Section 4 and Appendix A.2.

Camelyon16 Figure 11 shows that an end-to-end model trained on the downsampled version of Camelyon16 described in Section A.4 rapidly overfits when trained from scratch and from SimCLR-pretrained weights. The two-stage model, on the other hand, is less prone to overfitting. Table 11 shows that the two-stage learning pipeline outperforms end-to-end training, and is in turn outperformed by ItS2CLR.

Breast Ultrasound dataset Table 12 shows that end-to-end training outperforms the SimCLR+Aggregator baseline for the breast-ultrasound dataset, but is outperformed by ItS2CLR.

B.4. Tumor Localization Maps

Figure 12 provides additional tumor localization maps.

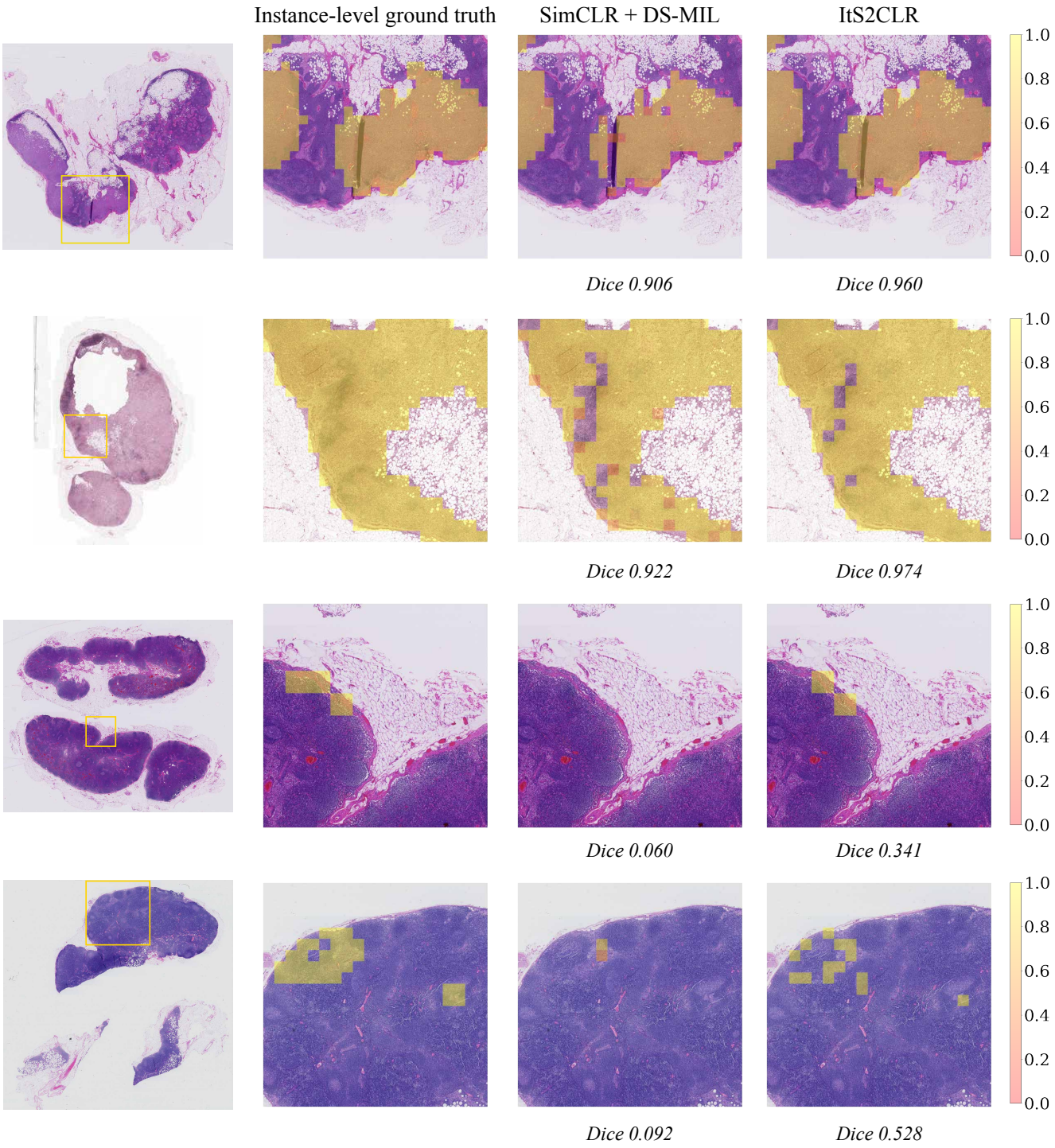

Figure 12. Additional tumor localization maps for histopathology slides from the Camelyon16 test set. Instance-level predictions are generated by the instance-level classifier of the DS-MIL trained on extracted instance-level features.

C. MIL Aggregators

C.1. Formulation of MIL Aggregators

In this section, we describe the different MIL aggregators benchmarked in Section 4.2 and Table 3.

Let $\mathcal{B}$ denote a collection of sets of feature vectors in $\mathbb{R}^d$. The bags of extracted features in the dataset are denoted by $\{H_b\}_{b=1}^B \subset \mathcal{B}$. An aggregator is defined as a function $g : \mathcal{B} \to [0, 1]$ mapping bags of extracted features to a score in $[0, 1]$.

There exist two main approaches in MIL:

1. *The instance-level approach*: using a logistic classifier on each instance, then aggregating instance predictions over a bag (e.g. max-pooling, top k-pooling).

2. *The embedding-level approach*: aggregating the instance embeddings, then obtaining a bag-level prediction via a bag-level classifier (e.g. attention-based aggregator, Transformer).

We denote the embeddings of the instances within a bag by $H = \{h_k\}_{k=1}^K$, where K is the number of instances.

Max-pooling obtains bag-level predictions by taking the maximum of the instance-level predictions produced by a logistic instance classifier ϕ, that is

$$g_\phi(H) = \max_{k=1,\cdots,K} \{\phi(h_k)\}. \tag{6}$$

Top-k pooling [46] produces bag-level prediction using the mean of the top-M ranked instance-level predictions produced by a logistic instance classifier ϕ, where M is a hyperparameter.

Let top$M$ (ϕ, H) denote the indices of the elements in H for which ϕ produces the highest M scores,

$$g_\phi(H) = \frac{1}{M} \sum_{k \in \text{top}M(\phi,H)} \phi(h_k). \tag{7}$$

Attention-based MIL [29] aggregates instance embeddings using a sum weighted by attention weights. Then the bag-level estimation is computed from the aggregated embeddings by a logistic bag-level classifier φ:

$$g_\varphi(H) = \varphi\left(\sum_{k=1}^K a_k h_k\right), \tag{8}$$

where a_k is the attention weight on instance k:

$$a_k = \frac{\exp\left(w^T \tanh(V h_k^T)\right)}{\sum_{j=1}^K \exp\left(w^T \tanh(V h_j^T)\right)}, \tag{9}$$

where $w \in \mathbb{R}^{l\times 1}$ and $V \in \mathbb{R}^{l\times d}$ are learnable parameters and l is the dimension of the hidden layer.

DS-MIL combines instance-level and embedding-level aggregation, we refer to DS-MIL [35] for more details on this approach.

Transformer [9] proposed an aggregation that uses an L-layer Transformer to process the set of instance features H. The initial set $H^{(0)}$ is set equal to H. Then it goes through the Transformer as follows:

$$\begin{aligned} H'^{(l)} &= \text{MSA}\left(H^{(l-1)}\right) + H^{(l-1))} \\ H^{(l)} &= \text{MLP}\left(H'^{(l-1)}\right) + H'^{(l-1)} \end{aligned} \tag{10}$$

for $l = 1, \cdots, L$, where MSA is multiple-head self-attention, MLP is a multi-layer perceptron network. Then the processed vectors $H^{(l)}$ are fed to **Attention-based MIL** [29] to obtain the bag-level predictions

$$g_\varphi(H^l) = \varphi\left(\sum_{k=1}^K a_k h_k^l\right). \tag{11}$$

Here a_k is the attention weight on instance k:

$$a_k = \frac{\exp\left(w^T \tanh(V (h_k^l)^T)\right)}{\sum_{j=1}^K \exp\left(w^T \tanh(V (h_j^l)^T)\right)}, \tag{12}$$

where $w \in \mathbb{R}^{p\times 1}$ and $V \in \mathbb{R}^{p\times d}$ are learnable parameters and p is the dimension of the hidden layer.

C.2. Implementation Details

Top-k pooling We select the ratio in Top-k pooling from the set $\{0.1\%, 1\%, 3\%, 10\%, 20\%\}$.

DS-MIL The weight between the two cross-entropy loss functions in DS-MIL is selected from the interval $[0.1, 5]$ based on the best validation performance.

Attention-based MIL. The hidden dimension of the attention module to compute the attention weights is set equal to the dimension of the input feature vector (512).

Transformer. We add light-weighted two-layer Transformer blocks to process instance features. We did not observe improvement in performance with additional blocks.