Exploiting Label-Independent Regularization from Spatial Patterns for Whole Slide Image Analysis

Supplementary Material

7. Attention Issues

To better understand this issue and its implications, we conducted a detailed analysis on the Camelyon16 dataset, which provides pixel-level annotations that allow us to obtain patch-level labels for evaluation. The attention values in the ABMIL method tend to be highly skewed, as discussed in the main text and its Figure 3. Often, only a few instances receive significant attention, while most instances are neglected. Many existing WSI visualization techniques, such as the one adopted in CLAM [31], apply a percentage transformation formula when visualizing attention scores:

$$P(s_i) = \frac{R(s_i)}{n} \times 100$$

where $P(s_i)$ is the percentile for score s_i, $R(s_i)$ represents the rank of score s_i. This ranking method artificially transforms the distribution of attention weights to be uniform (with equal intervals). While this transformation allows for the creation of some ideal visualizations, it alters the original attention distribution in a way that cannot be reversed and obscures the real attention distribution issues. To better understand the attention distribution and its implications, instead of visualizing transformed attention weights, we visualize the raw attention weights of a negative example in Figure 5. The instance with the highest attention weight (red) accounts for 91.3% of the total attention, while the remaining 1738 instances (blue) share the remaining 8.7%.

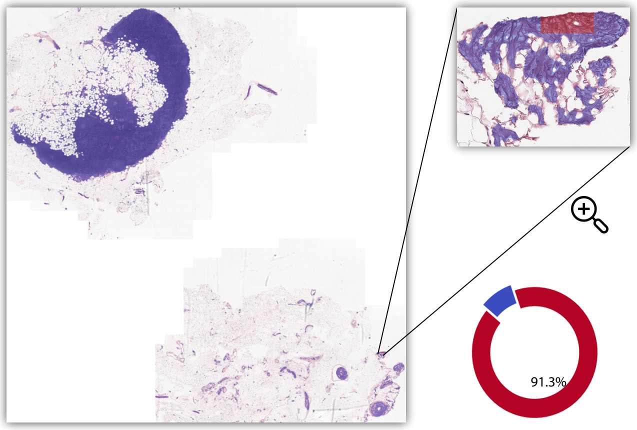

Figure 4. Visualization of ABMIL attention for a negative bag.

Ideally, in negative cases, all patches should be learned as negative and have uniform weights to ensure that information from each patch is reasonably considered during gradient updates. However, the highly skewed attention weights in ABMIL allow only a small number of instance features in each bag to be used for updating the model. The model remembers the bag by memorizing the features of only a few instances, leading to underutilization of information in negative instances. This increases the possibility of the model overfitting negative instances into positive instances to remember positive bags.

8. Datasets

We evaluate our proposed method on three diverse and challenging histopathology WSI datasets, each representing a unique classification task within the domain:

- **CAMELYON-16 [3]:** This dataset is utilized for binary classification to detect tumor presence within lymph node sections. It consists of 270 training slides and 129 testing slides, with a total of 4,647,505 patches. The task assesses the model's proficiency in differentiating normal and tumor tissues. As CAMELYON-16 lacks a predefined validation set, we allocate 15% of the training set for validation purposes.
- **The Cancer Genome Atlas (TCGA) Lung Cancer:** This dataset aims at classifying Lung Squamous Cell Carcinoma (LUSC) versus Lung Adenocarcinoma (LUAD). It contains 1,044 slides, from which 17,407,256 patches are extracted. This dataset evaluates the model's ability to distinguish between related cancer subtypes. In the absence of a predefined train-test split, we designate two-thirds of the samples for training, reserving 15% for validation and the remaining one-third for testing.
- **BRACS [4]:** This dataset focuses on a three-class classification challenge involving normal tissue, atypical ductal hyperplasia, and malignant tumors, with 6,297,595 patches derived from 395 training, 65 validation, and 87 testing slides. It tests the model's capability in classifying tissue pathologies along a continuum of severity from benign to potentially malignant to unequivocally cancerous.

9. Implementation Details

Our training protocol utilized the AdamW optimizer with an initial learning rate of 1×10^{-4} under cosine annealing scheduling for 100 epochs with weight decay set to 1×10^{-5}. The best checkpoint on the validation set was used for test set evaluation.

We carefully considered dataset splitting strategies to ensure rigorous evaluation and prevent potential information leakage. For the three datasets used in our study:

- **TCGA-Lung**: Without official splits, we randomly divided the dataset with a seed value of 42: one-third for testing and 15% of the remaining data for validation.
- **BRACS**: We utilized the official training, validation, and test splits provided by the dataset authors, which maintain proper patient-level separation.
- **Camelyon16**: We used the official training and test splits, and randomly allocated 15% of the training set for validation using a seed value of 42.

When using official splits (BRACS and Camelyon16), we benefit from proper patient-level separation, where all WSIs from the same patient are confined to a single split. This separation is crucial for realistic performance evaluation, as mixing WSIs from the same patient across different splits could lead to patient-level information leakage and potentially overestimate the model's performance in real clinical applications. For TCGA-Lung where official splits were not available, we maintained consistency in our splitting approach using fixed random seeds to ensure reproducibility. For all experiments, we report test set performance using the model checkpoint that achieved the best performance on the validation set.

For preprocessing, we adapted CLAM's pipeline [31] with several modifications. Non-overlapping patches of size 224×224 were extracted, departing from CLAM's original 256×256 patch size. We introduced an additional grid alignment step where patch boundaries were extended to the nearest multiple of the patch size, ensuring consistent mapping between physical coordinates and the model's input grid. This alignment eliminates potential rounding errors in patch coordinate mapping, though it may result in a slight increase in the total number of patches compared to the original CLAM preprocessing.

Algorithm 1 Expand Contours to Fixed Step-size Grid

```
global step_size = 224
def extend_contour(start_x, start_y, w, h):
    w += start_x % step_size
    h += start_y % step_size
    start_x -= start_x % step_size
    start_y -= start_y % step_size
    return start_x, start_y, w, h
# contour: (start_x, start_y, w, h)
contour = extend_contour(contour)
```

For the reconstruction task, we use cosine distance as the loss function ($\mathcal{L}_{recon}$ in Algorithm 2). Unlike pixel reconstruction tasks, which commonly use Mean Squared Error (MSE) as the loss function due to the fixed distribution and value range of input features, our feature reconstruction task involves input scales that might vary significantly among different feature extractors. Therefore, using cosine distance provides a more scale-invariant objective compared to MSE loss.

The models were trained using an AdamW optimizer with a learning rate of 1e-4 and a weight decay of 1e-5 for 100 epochs, coupled with cosine annealing for learning rate scheduling. The hyperparameters for the loss function were set as follows: $\lambda_{recon} = 1.8$, $\lambda_{corr} = 0.1$, and $\lambda_{comp} = 0.1$. This choice was guided by the goal of approximately balancing the gradient magnitudes between losses at the start of training, so that neither dominates the optimization. This joint optimization allows the feature-induced stream to regularize the encoder and support the formation of a robust latent space, while maintaining discriminative capability through label guidance. We used these weights across all experiments.

Algorithm 2 Simplified Training & Evaluation Process

```
# Training
model.train()
for $\mathcal{G}$ in data_loader do
    $\mathcal{G}_m$ = $\mathcal{G}$
    $\mathcal{G}_m$[mask_idx] = mask_token
    corr_latent = model.encoder($\mathcal{G}_m$)
    # reconstruction task on the corrupted view
    $\mathcal{G}_{recon}$ = model.decoder(corr_latent)
    $\mathcal{L}_{recon}$ = cosine_dist($\mathcal{G}$[mask_idx], $\mathcal{G}_{recon}$[mask_idx])
    # classification task on the corrupted view
    $\hat{y}_{corr}$ = model.classifier(corr_latent)
    $\mathcal{L}_{corr}$ = cross_entropy($y$, $\hat{y}_{corr}$)
    # classification task on the complete view
    $\hat{y}_{comp}$ = model.classifier(model.encoder($\mathcal{G}$))
    $\mathcal{L}_{comp}$ = cross_entropy($y$, $\hat{y}_{comp}$)
    loss = $\lambda_{recon}$ $\mathcal{L}_{recon}$ + $\lambda_{comp}$ $\mathcal{L}_{comp}$ + $\lambda_{corr}$ $\mathcal{L}_{corr}$
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
scheduler.step()
# Evaluation
model.eval()
for $\mathcal{G}$ in data_loader do
    $\hat{y}_{comp}$ = model.classifier(model.encoder($\mathcal{G}$))
    $\mathcal{L}_{comp}$ = cross_entropy($y$, $\hat{y}_{comp}$)
    loss = $\mathcal{L}_{comp}$
```

10. Efficiency

As shown in Table 4, our SRMIL framework achieves competitive computational efficiency, requiring on average 0.0785 seconds per slide for training and 0.0232 seconds for inference. While ATTMIL is faster due to its lightweight architecture, SRMIL provides a favorable trade-off between

Table 4. Computational efficiency analysis. We report the average time per WSI on the Camelyon16 dataset. All experiments were conducted on a single NVIDIA A6000 GPU.

Method	Training (s/WSI)	Inference (s/WSI)
ATTMIL	0.0148	0.0035
RRTMIL	0.0548	0.0098
TransMIL	0.1210	0.0198
SRMIL (Ours)	0.0785	0.0232

speed and accuracy.

11. More Ablations

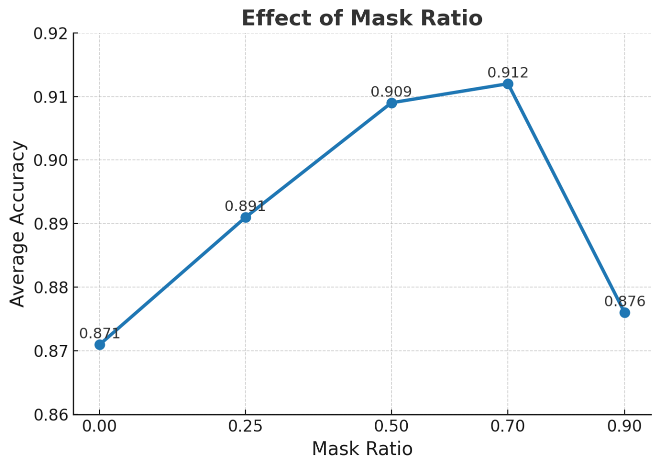

Figure 5. Visualization of the effect of different mask ratios.

When the mask ratio is set to 0, the performance (~ 0.874) is close to that observed without applying the reconstruction loss in Tab. 3. Moreover, we observe that mask ratios in the range of 0.5–0.7 yield the most effective regularization, leading to consistently improved accuracy. However, further increasing the mask ratio results in performance degradation, suggesting that excessive masking cannot introduce too much useful information for latent space regularization.