BIG-LAYERS: ENABLING END-TO-END TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training deep neural networks on extremely large inputs—such as gigapixel Whole Slide Images (WSIs) in digital pathology—poses significant challenges due to GPU memory constraints. Multiple Instance Learning (MIL) circumvents this limitation by processing patches from a WSI. However, the encoder used to get patch embeddings is usually a generic pre-trained deep neural network model. In this paper, we propose a training strategy that enables training the encoder by dynamically offloading intermediate activations of a layer to CPU RAM, allowing the layer to process inputs that do not fit in the GPU memory. We demonstrate the effectiveness of our approach on PANDA and CAMELYON datasets using popular MIL approaches. Experimental results indicate that our method improves the Quadratic Weighted Kappa (QWK) metric, on PANDA, by 7–15 percentage points compared to baselines where encoders are kept frozen. Evaluations on external test sets further suggest better generalisation. The code will be made publicly available upon publication.

1 INTRODUCTION

Advances in deep learning have revolutionised histopathology (Unger & Kather, 2024; Van der Laak et al., 2021), but some challenges in handling Whole Slide Images (WSIs) remain. One of the challenges is the enormous size of WSIs, which can be up to a few gigapixels. It prevents the use of common machine learning techniques, as these techniques require much smaller images to be directly applicable. For classification tasks, a common approach to handle such large images is to use Multiple Instance Learning (MIL) (Dietterich et al., 1997; Maron & Lozano-Pérez, 1997) in which some patches are extracted from the WSI, and it is assumed that a subset of those patches corresponds to the desired label. It is also sometimes referred to as weakly supervised learning. MIL involves three steps: first, an encoder, such as a Convolutional Neural Network (CNN), converts a patch into an embedding; second, an aggregator pools the embeddings into an aggregated embedding; and third, a classifier assigns a label to the aggregated embedding.

However, due to the large number of patches required to train a model effectively, training a model end-to-end is practically infeasible on most GPUs. Therefore, the common approach is to use a pre-trained encoder to extract the embeddings for all the patches and train only the aggregator and the classifier models (Song et al., 2023).

Recent work has attacked the GPU-memory bottleneck for whole-slide and other multi-megapixel inputs using several complementary strategies. Some authors leverage CUDA unified memory to let the runtime page very large tensors between host and device (Chen et al., 2021); others stream or tile the input so that convolutional layers run on spatial tiles and intermediate outputs are stitched (Pinckaers et al., 2020); still others reduce GPU state by offloading optimizer/parameter state to CPU (Ren et al., 2021) or running the backward pass for only a subset of patches (Skrede et al., 2020).

In this work, we introduce a training strategy that allows comprehensive end-to-end training of models by efficiently using CPU RAM and GPU RAM for layers whose input and output are too big to fit in the GPU memory, including layers that require computing statistics over the entire input. We demonstrate the utility of our approach by training models on the PANDA dataset (Bulten et al., 2022) using ResNet18 (He et al., 2016) as the encoder. Models trained using our method perform significantly better on the test set compared to baseline models where the encoder is frozen, gaining several percentage points on Cohen's quadratic weighted Kappa κ^2 (QWK) metric.

2 Related Work

Memory- and I/O-aware techniques for training on large inputs have followed several broad paradigms; we summarise each and explain how our method differs.

Unified / runtime-managed memory Chen et al. (2021) demonstrate that CUDA unified memory can enable end-to-end training on entire whole-slide images by relying on the CUDA runtime to page tensors between host and device. This approach simplifies implementation because the runtime performs paging implicitly, but it offers limited control over transfer scheduling. In practice, the explicit high-level unified-memory knobs used in older TensorFlow releases are not exposed as stable TensorFlow version 2 public APIs. Unified memory may also be suboptimal compared to methods that explicitly manage data transfers between GPU and CPU (Landaverde et al., 2014; Jarząbek & Czarnul, 2017; Alawneh et al., 2018). Our method does not depend on CUDA unified memory; instead, we perform layer-aware transfers and precisely control when and how tensors move between CPU and GPU.

Streaming / tiled convolutional training Pinckaers et al. (2020) split the input into spatial tiles and execute convolutions tile-by-tile, stitching intermediate feature maps. They use this idea to train ResNet architectures end-to-end on large Whole Slide Images. They use gradient checkpointing (Chen et al., 2016) to avoid storing intermediate representations for those layers. While this alleviates memory constraints, it can not be used to train layers that require computing statistics over the entire input, which includes common layers such as batch normalisation. By contrast, we present a more generic method for implementing commonly used layers. Our method enables training layers that require global statistics.

Partial backward / selective gradient updates Skrede et al. (2020) reduce memory consumption by computing gradients for the encoder only for selected patches. Their approach reduces memory and computation at the cost of computing approximate gradients; they are effective when full gradients are unnecessary but can harm representational learning when end-to-end gradient fidelity matters. In contrast, we compute gradients for all the tiles processed by the encoder.

Optimizer/state offload (ZeRO-offload) Ren et al. (2021) design ZeRO-Offload to reduce GPU memory pressure by moving model state and optimizer work onto CPU. ZeRO-Offload partitions model states and keeps model parameters on the GPU while offloading averaged gradients, and the optimizer update computation to the CPU. The approach uses a highly optimized CPU Adam (Kingma, 2014) implementation and enables training large models on a single GPU. By contrast, our method targets memory arising from extremely large single-example tensors rather than the whole model state or the optimizer. Our approach enables training layers whose activation does not fit in the GPU. ZeRO-Offload and our approach are complementary and can be combined.

3 Proposed Method

3.1 Overview

Our method allows training neural networks when the input and output of one or more layers do not fit in the GPU memory. To achieve that, our method employs the following key strategies:

- **Partitioning:** Large tensors are divided into sub-tensors that fit within GPU memory.
- **Selective Offloading:** Intermediate activations that would otherwise exceed GPU capacity are stored in CPU RAM and transferred back to the GPU only when needed.
- **Layer-Specific Execution:** Compute-intensive layers (e.g., convolutional layers and batch normalisation layers) leverage the partitioning and selective offloading strategies on the GPU, while computationally cheap layers (e.g., activation layers such as ReLU and pooling layers such as MaxPool) are executed on the CPU.
- **Efficient Backpropagation:** The same data partitioning and offloading techniques are applied to gradient computations, ensuring a memory-efficient backward pass.

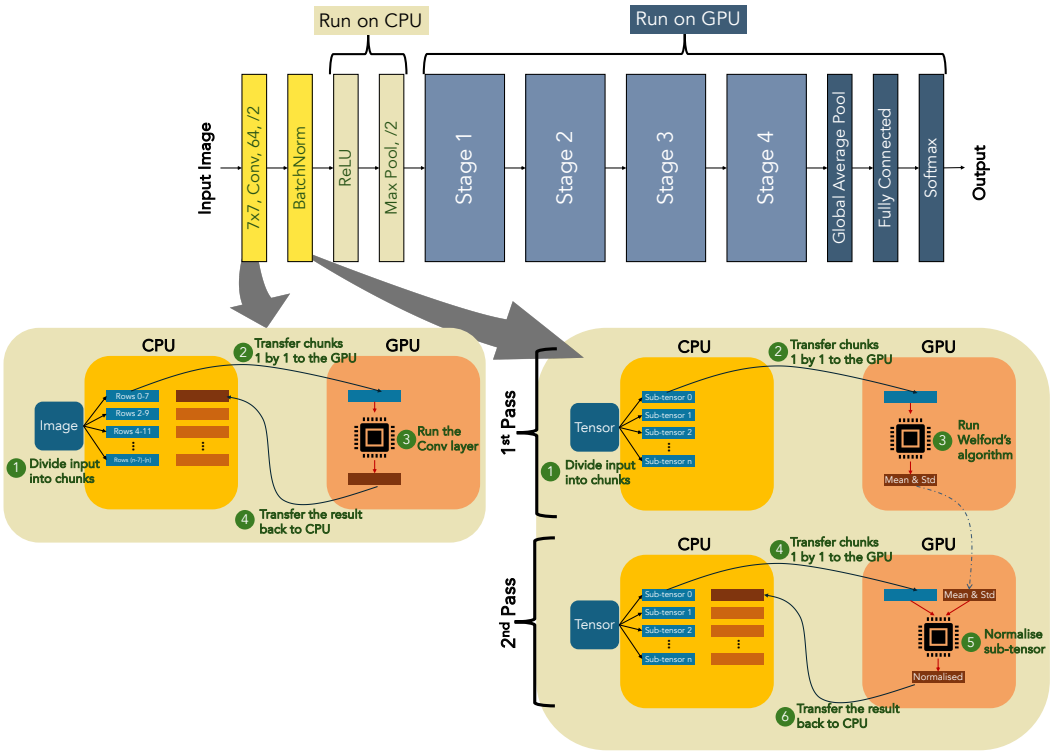

Figure 1: ResNet18 architecture implemented using our method.

Typically, the first few layers of modern CNN architectures progressively downsample the input image. These layers consume a lot of memory, but as the network deepens, the feature maps become smaller, and the memory requirements decrease substantially. Our approach utilises the CPU RAM to process the initial high-memory-demand layers. Once sufficient downsampling has occurred, the data remains on the GPU for the rest of the network. How many layers should leverage the CPU RAM can easily be adapted to the particular GPU setup using our implementation.

Algorithm 1 outlines our method. Figure 1 illustrates our implementation of ResNet18 based on Algorithm 1.

3.2 Partitioning and Selective Offloading

The core of our method for handling computationally heavy layers is to partition the input tensor into sub-tensors and incrementally compute the output using those sub-tensors. First, we divide the input tensor into (potentially overlapping) sub-tensors. Then, we transfer sub-tensors to the GPU one by one and perform the layer-specific computation with the sub-tensor present on the GPU. Some layers, like normalisation layers, require repeating the previous step to get the final output. For example, for BatchNorm, we first compute the mean and standard deviation of the input using Welford's algorithm in the first sequential transfer of sub-tensors, followed by a second transfer to normalise the input using the mean and standard deviation computed in the first pass. For the convolution layer, we transfer the sub-tensors only once, but the sub-tensors might have an overlap depending on the stride used in the layer.

Since the concrete implementation of our generic method differs from layer to layer, we detail the implementation of the forward pass for BatchNorm in Algorithm 2 and illustrate it in Figure 1, demonstrating how to use the generic method for a layer that requires global statistics.

Algorithm 1 Our method

Require: Image X, layers $\{L_i\}_{i=1}^{N}$ of the neural network
Ensure: Output tensor Y

procedure EFFICIENTFORWARD($X, \{L_i\}_{i=1}^{N}$)
  for each layer L_i in the network **do**
    if L_i is compute-intensive (e.g., Conv, BatchNorm) **then**
      Partition X into sub-tensors $\{X_j\}$ that fit in GPU memory
      for each sub-tensor X_j **do**
        Transfer X_j to GPU
        Compute $Y_j \leftarrow L_i(X_j)$ on GPU
        Transfer Y_j back to CPU if subsequent layers require partitioning
      end for
    else if L_i is computationally inexpensive (e.g., ReLU, MaxPool) **then**
      Compute $Y \leftarrow L_i(X)$ directly on CPU
    end if
    $X \leftarrow Y$ ▷ Update input for the next layer
    if tensor size has been significantly reduced **then**
      Transfer entire X to GPU for remaining layers
    end if
  end for
  $Y \leftarrow X$ ▷ Store final output
  return Y
end procedure

Algorithm 2 Memory-efficient Batch Normalisation (Forward Pass)

Require: Input tensor X, scale parameter γ, shift parameter β, running mean *r_mean*, running variance *r_var*, maximum count *max_N* of tensor elements to transfer to GPU, small constant ϵ
Ensure: Normalized output tensor Y, updated running statistics (*r_mean*, *r_var*)

procedure BIGBATCHNORMFORWARD(X, γ, β, r_mean, r_var, max_N, ϵ)
  Partition X into sub-tensors $\{X_i\}$, each with at most max_N elements
  if training **then**
    for each sub-tensor X_i of X **do**
      Transfer X_i to GPU
      Update variables in Welford's algorithm
      Transfer Y_i back to CPU memory
    end for
    Update running statistics r_mean and r_var
  else
    Set $\mu \leftarrow$ r_mean and $\sigma^2 \leftarrow$ r_var
  end if
  Normalize:
  for each sub-tensor X_i of X **do**
    Transfer X_i to GPU
    Compute:

$$Y_i \leftarrow \gamma \cdot \frac{X_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

    Transfer Y_i back to CPU memory
  end for
  return Y, r_mean, r_var
end procedure

3.3 LIMITATIONS

Our methodology involves partitioning the input tensor into sub-tensors and executing computations incrementally using these sub-tensors. Although all commonly used layers can be computed in this manner, any computationally intensive layer whose computation can not be decomposed in this way

will present a significant constraint as it will necessitate computation on the CPU, which can be prohibitively expensive.

Architectures that do not significantly downsample the input in the early layers or do not downsample at all (like ViTs, which do not downsample except for the initial embedding layer) require using our method for most (or all) layers in the network. This can be too slow to be practically useful, especially for very deep networks.

4 EXPERIMENTS

4.1 DATASETS

PANDA This dataset (Bulten et al., 2022) consists of 10616 WSIs of hematoxylin and eosin (H&E)-stained needle biopsy WSIs of prostate tissue from two medical centres. Each WSI carries an International Society of Urological Pathology (ISUP) grade - 0 for normal tissue and 1 to 5 for cancer, forming a 6-class classification task. All slides are in 20× resolution.

TCGA-PRAD This dataset contains 449 resection WSIs from The Cancer Genome Atlas (TCGA) repository of prostate adenocarcinoma Zuley et al. (2016). After a pathologist's review, we removed 5 WSIs lacking tumour and 23 that could not be opened. The remaining 421 WSIs represent 394 unique patients; we randomly selected one WSI per patient to use as an external test set.

CAMELYON17 This dataset (Litjens et al., 2018) comprises 1000 H&E-stained WSIs from five Dutch medical centres (with five slides per patient in the released patient-centric setup) and—when aggregated with Camelyon16—forms a collection of 1399 annotated WSIs. CAMELYON17 provides patient-level pN-stage labels (aggregating slide-level findings) and includes a subset of lesion-level manual annotations (10 annotated slides per centre in the training set) to support both slide-level classification and lesion localization tasks. All slides are in 40× resolution.

CAMELYON16 This dataset (Bejnordi et al., 2017) contains 399 WSIs of H&E-stained sentinel lymph node sections collected from two Dutch centres. The dataset splits into 270 training slides and 129 test slides; the training slides include pixel-level delineations of metastatic regions provided as XML contours and binary masks. All slides are in 40× resolution.

4.2 DATASET PREPARATION

We tile the WSIs into non-overlapping 256×256 patches. For PANDA and TCGA-PRAD, we retain only the patches with at least 60% foreground pixels. We convert each tile to greyscale and consider pixels with intensities between 3 and 230 as foreground. For CAMELYON 16/17, we follow Zhang et al. (2022) for tiling the WSIs, tiling them at 20× resolution after localising the tissue region using OTSU's threshold method (Otsu et al., 1975).

4.2.1 PANDA SPLITS

We use the training/validation/test split from Song et al. Song et al. (2024), which provides a label-stratified division of 80:10:10 after removing 1061 noisy WSIs, resulting in 7647, 954, and 954 WSIs for the training, validation, and test subsets, respectively. We train all models exclusively on the PANDA training subset and evaluate them on its test subset. Additionally, we use TCGA-PRAD as an external test set to further assess generalisability.

4.2.2 CAMELYON SPLITS

We train exclusively on the CAMELYON17 training set and follow the reprocessed binary labels proposed by Ling et al. (2025). From CAMELYON17's training set we randomly select 50 WSIs to form a validation set and use the remaining 472 WSIs for training. We evaluate on the official CAMELYON17 test set and use CAMELYON16 as an external test set.

4.2.3 METHODS

We evaluate three methods: Attention-based MIL (ABMIL) Ilse et al. (2018), Double-tier feature distillation MIL (DTFD) Zhang et al. (2022), and TransMIL Shao et al. (2021). For each method, we compare baseline models that freeze the ResNet-18 encoder with our approach that trains it. All models are initialised with a ResNet-18 encoder pre-trained on ImageNet Deng et al. (2009), and we use its final stage output as the patch embedding. We train three models per method and select the best checkpoint based on the validation QWK score.

4.2.4 TILE SELECTION

During training, we randomly sample 256 tiles per WSI to form a bag. During testing, we use all foreground tiles; for TCGA-PRAD, we additionally evaluate on 256 randomly selected tiles per WSI. As the specific 256 tiles may vary across random seeds, we run 100 tests per model using the mean QWK as the final QWK for the model.

For the CAMELYON experiments, we adopt method-specific sampling strategies that work best for the method. For baseline models we construct training bags by randomly sampling 1024 tiles per WSI. For our models we sample 512 tiles per WSI and use a batch size of 2. During evaluation, for our DTFD and TransMIL models we sample 2048 tiles per WSI, repeat the evaluation 3 times, and use the mean of those 3 runs as the final score for a given model. For our ABMIL models and all baseline models, we use all foreground tiles at test time.

4.2.5 OPTIMISATION HYPERPARAMETER SETTINGS

PANDA We test two hyperparameter configurations per method. In the first, we select a learning rate from $1e{-}4, 5e{-}5, 1e{-}5$ and train for 20 epochs with cosine annealing (Loshchilov & Hutter, 2016), a batch size of 2, and gradient accumulation over 16 steps, inspired by Song et al. (2024). In the second, we train for 45 epochs without cosine annealing or gradient accumulation, applying exponential decay (rate 0.955). All experiments use the Adam optimiser (Kingma, 2014) with a weight decay of $1e{-}4$.

CAMELYON17 For baseline models we train with a learning rate of $1e{-}4$ for 120 epochs. For our models we train with a learning rate of $5e{-}5$ for 90 epochs. Both baseline and our models use a cosine-annealing learning-rate schedule and gradient accumulation over 2 steps. Baseline models use Adam optimiser while our models use Adam optimiser with DEMON momentum decay rule (Chen et al., 2022). Adam is used with weight decay of $1e{-}4$.

4.2.6 INPUT AUGMENTATION

We train models with and without augmentation. We employ Gaussian blur, colour jitter, random horizontal and vertical flips, and random rotation. We apply a single set of randomly selected parameters uniformly per WSI rather than augmenting each tile independently.

5 RESULTS AND DISCUSSION

5.1 PANDA AND TCGA-PRAD

Table 1 shows that end-to-end training with our method improves the QWK by 7 to 15 percentage points on the PANDA test set. On TCGA-PRAD, the baseline outperforms ABMIL and TransMIL without augmentation; however, our method with augmentation attains superior results. Baseline models that freeze the encoder do not benefit from augmentation, whereas our approach exploits it effectively.

TransMIL exhibits high variance when tested on all tiles. We hypothesise that these inferior results stem from a training-testing mismatch. Specifically, all models train on bags with 256 tiles per WSI. While the PANDA test set averages 400 tiles per WSI, with the maximum being 1400 tiles, TCGA-PRAD averages 12800 tiles with the maximum being 41000. Because TransMIL employs self-attention that directly processes inter-tile interactions, it likely performs suboptimally on WSIs with approximately 50 times more tiles than WSIs seen during training.

Table 1: Cohen's quadratic weighted kappa (QWK) on the PANDA dataset's test subset and TCGA-PRAD. We train three models per method and report the mean and standard deviation of QWK. For TCGA-PRAD, we also report QWK using only 256 randomly selected tiles per scan. We test each model 100 times when using only 256 tiles and use the mean QWK of the 100 runs as the final QWK for that model. "-Aug" indicates the use of augmentation.

Method		PANDA	TCGA-PRAD-All	TCGA-PRAD-256
ABMIL	Baseline	76.74 ± 0.43	56.34 ± 0.85	47.31 ± 1.18
	+ Ours	84.60 ± 1.15	53.53 ± 2.35	49.61 ± 1.62
DTFD	Baseline	73.64 ± 1.48	55.26 ± 1.87	47.83 ± 0.74
	+ Ours	87.13 ± 0.63	64.54 ± 2.97	56.04 ± 2.42
TransMIL	Baseline	81.87 ± 1.13	44.63 ± 3.16	46.86 ± 1.70
	+ Ours	89.60 ± 0.52	29.96 ± 9.12	52.49 ± 1.67
ABMIL-Aug	Baseline	77.89 ± 0.50	52.83 ± 0.87	51.29 ± 0.69
	+ Ours	86.91 ± 1.31	65.86 ± 1.73	63.46 ± 0.89
DTFD-Aug	Baseline	71.53 ± 1.18	56.79 ± 4.21	49.21 ± 2.44
	+ Ours	86.59 ± 0.40	69.47 ± 0.76	62.65 ± 0.54
TransMIL-Aug	Baseline	78.06 ± 1.06	36.25 ± 0.78	44.69 ± 2.54
	+ Ours	89.39 ± 0.85	45.56 ± 7.15	65.33 ± 1.14

Table 2: Accuracy on the CAMELYON17 dataset's test subset and CAMELYON16 whole set. We train three models per method and report the mean and standard deviation of accuracy. For DTFD and TransMIL, we also report accuracy for our models using only 2048 randomly selected tiles per scan. We test each model 3 times when using only 2048 tiles and use the mean accuracy of the 3 runs as the final accuracy for that model. "-Aug" indicates the use of augmentation.

Method		CAMELYON17	CAMELYON16
ABMIL-Aug	Baseline	87.71 ± 0.92	81.35 ± 1.19
	+ Ours	89.27 ± 0.96	90.59 ± 0.40
DTFD-Aug	Baseline	87.64 ± 0.44	82.21 ± 0.98
	+ Ours	88.06 ± 0.79	85.87 ± 1.71
TransMIL-Aug	Baseline	87.29 ± 0.56	81.52 ± 0.15
	+ Ours	91.03 ± 0.07	83.33 ± 1.56

We validate this hypothesis by testing on TCGA-PRAD using a subset of 256 randomly selected tiles per WSI. To handle variability from random selection of tiles, we run the evaluation 100 times and use the mean QWK as the final QWK for each model. Under these conditions, TransMIL improves notably, reaching QWK values comparable to other methods. Moreover, our approach with augmentation also maintains superior generalisability on TCGA-PRAD.

5.2 CAMELYON

Table 2 reports accuracy on the CAMELYON17 test subset and the CAMELYON16 whole set. End-to-end training with our method increases accuracy by up to 9 percentage points. Many whole-slide images in both CAMELYON datasets contain substantially more tiles—up to an $80\times$ increase—than the 512-tile bags we use for training; in CAMELYON we observe high evaluation variance for both DTFD and TransMIL under full-tile evaluation. To mitigate this effect, we evaluate DTFD and TransMIL by randomly sampling 2048 tiles per WSI (the same number use in validation for checkpoint selection), repeat each evaluation three times, and use the mean accuracy across the three runs as the final score for a given model. For ABMIL, using the full set of foreground tiles at test time consistently improves performance on CAMELYON16, which makes ABMIL the most robust approach in this regard.

5.3 SPEED

We compare the speed of the baseline method with our method. For the baseline, we compute the embeddings on the fly on GPU and do not run the backward pass for the encoder, while for our method, backpropagation is run for the encoder as well. When training on PANDA, baseline models take about 45 minutes for one epoch, while models trained using our approach take about 200 minutes for one epoch on NVIDIA RTX 3090.

Benchmark protocol We performed an *incremental-stage* sweep in which we increase the image side length (starting at 7,680 px, step 256 up to a maximum of 13,824 px) and for each image side attempt progressively more memory-intensive configurations of the network until a configuration fits on the device. Key parameters of the sweep are:

- batch size: 1,
- warmup iterations per size: 5,
- measured repeats per size: 10 (reported mean and standard deviation),
- optimizer: AdamW (LR $= 10^{-5}$, weight decay $= 10^{-4}$),
- mixed precision: automatic AMP (CUDA float16),
- device: NVIDIA RTX 3090 (24 GB).

Figure 2 shows mean wall times (±1 standard deviation shaded) as a function of image side; the top x-axis shows the total pixel count (side^2). The plot was produced from the raw sweep output and highlights how larger images and different stage configurations affect throughput.

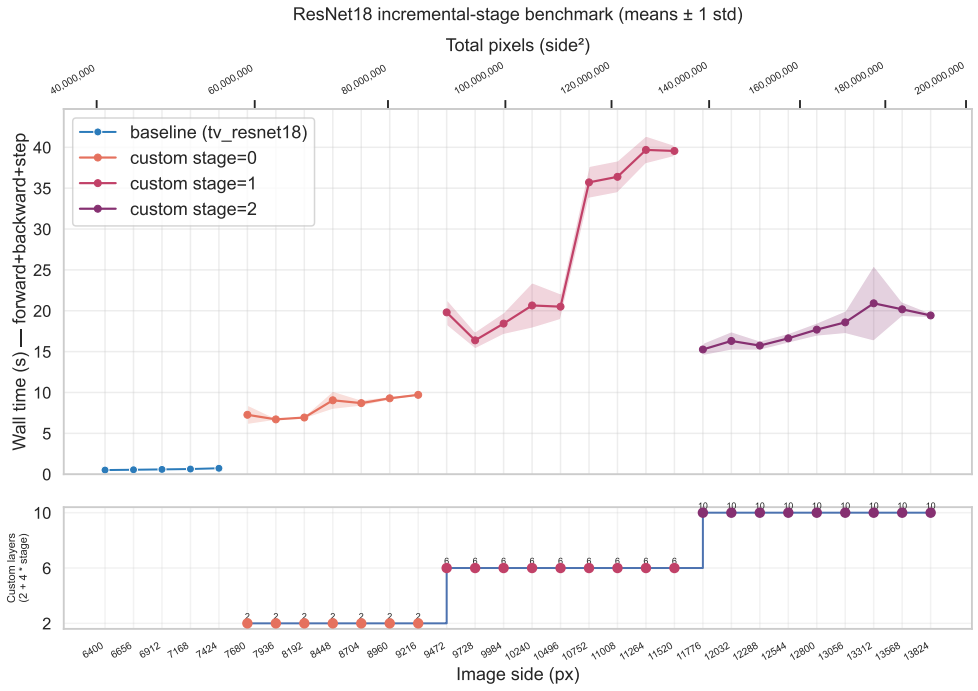

Figure 2: Incremental-stage benchmark: mean wall time per training iteration (forward + backward + step) as a function of image side. Shaded bands show ±1 standard deviation across measured repeats. The top axis reports total pixel count (side^2). The sweep attempts multiple configuration strategies per side (see text). A stage refers to a group of 2 basic blocks.

5.4 Discussion

While we focus on MIL for histopathology in this work, our method is broadly applicable. It allows training on WSIs without dividing them into patches, which can be helpful in applications where global context is essential.

The memory constraints we address are not unique to histopathology. For instance, remote sensing images can have a large image size, which poses challenges in tasks like segmentation Huang et al. (2018) and object detection Li et al. (2022). Our approach can be useful in these and other applications as well.

6 Conclusion

We introduce a novel method to address the memory constraint encountered on GPU while processing large inputs such as WSIs. By leveraging CPU RAM as an auxiliary memory resource, our approach enables processing input tensors that exceed the GPU memory capacity.

We train various MIL models end-to-end on the PANDA dataset using our approach and observe significant improvement over baseline MIL models in both the PANDA test subset and an external dataset. The results demonstrate the usefulness of our approach.

References

Luay Alawneh, Emad Rawashdeh, Mahmoud Al-Ayyoub, and Yaser Jararweh. Gpu parallelization of sequence segmentation using information theoretic models. *Simulation Modelling Practice and Theory*, 86:11–24, 2018.

Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes Van Diest, Bram Van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen AWM Van Der Laak, Meyke Hermsen, Quirine F Manson, Maschenka Balkenhol, et al. Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. *Jama*, 318(22):2199–2210, 2017.

Wouter Bulten, Kimmo Kartasalo, Po-Hsuan Cameron Chen, Peter Ström, Hans Pinckaers, Kunal Nagpal, Yuannan Cai, David F Steiner, Hester Van Boven, Robert Vink, et al. Artificial intelligence for diagnosis and gleason grading of prostate cancer: the panda challenge. *Nature medicine*, 28(1):154–163, 2022.

Chi-Long Chen, Chi-Chung Chen, Wei-Hsiang Yu, Szu-Hua Chen, Yu-Chan Chang, Tai-I Hsu, Michael Hsiao, Chao-Yuan Yeh, and Cheng-Yu Chen. An annotation-free whole-slide training approach to pathological classification of lung cancer types using deep learning. *Nature communications*, 12(1):1193, 2021.

John Chen, Cameron Wolfe, Zhao Li, and Anastasios Kyrillidis. Demon: improved neural network training with momentum decay. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3958–3962. IEEE, 2022.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71, 1997.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Bohao Huang, Daniel Reichman, Leslie M Collins, Kyle Bradbury, and Jordan M Malof. Tiling and stitching segmentation output for remote sensing: Basic challenges and recommendations. *arXiv preprint arXiv:1805.12219*, 2018.

Maximilian Ilse, Jakub Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *International conference on machine learning*, pp. 2127–2136. PMLR, 2018.

Łukasz Jarząbek and Paweł Czarnul. Performance evaluation of unified memory and dynamic parallelism for selected parallel cuda applications. *The Journal of Supercomputing*, 73:5378–5401, 2017.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Raphael Landaverde, Tiansheng Zhang, Ayse K Coskun, and Martin Herbordt. An investigation of unified memory access performance in cuda. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6. IEEE, 2014.

Zheng Li, Yongcheng Wang, Ning Zhang, Yuxi Zhang, Zhikang Zhao, Dongdong Xu, Guangli Ben, and Yunxiao Gao. Deep learning-based object detection techniques for remote sensing images: A survey. *Remote Sensing*, 14(10):2385, 2022.

Xitong Ling, Yuanyuan Lei, Jiawen Li, Junru Cheng, Wenting Huang, Tian Guan, Jian Guan, and Yonghong He. Comprehensive benchmark dataset for pathological lymph node metastasis in breast cancer sections. *Scientific Data*, 12(1):1381, 2025.

Geert Litjens, Peter Bandi, Babak Ehteshami Bejnordi, Oscar Geessink, Maschenka Balkenhol, Peter Bult, Altuna Halilovic, Meyke Hermsen, Rob Van de Loo, Rob Vogels, et al. 1399 h&e-stained sentinel lymph node sections of breast cancer patients: the camelyon dataset. *GigaScience*, 7(6):giy065, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. *Advances in neural information processing systems*, 10, 1997.

Nobuyuki Otsu et al. A threshold selection method from gray-level histograms. *Automatica*, 11 (285-296):23–27, 1975.

Hans Pinckaers, Bram Van Ginneken, and Geert Litjens. Streaming convolutional neural networks for end-to-end learning with multi-megapixel images. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1581–1590, 2020.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564, 2021.

Zhuchen Shao, Hao Bian, Yang Chen, Yifeng Wang, Jian Zhang, Xiangyang Ji, et al. Transmil: Transformer based correlated multiple instance learning for whole slide image classification. *Advances in neural information processing systems*, 34:2136–2147, 2021.

Ole-Johan Skrede, Sepp De Raedt, Andreas Kleppe, Tarjei S Hveem, Knut Liestøl, John Maddison, Hanne A Askautrud, Manohar Pradhan, John Arne Nesheim, Fritz Albregtsen, et al. Deep learning for prediction of colorectal cancer outcome: a discovery and validation study. *The Lancet*, 395 (10221):350–360, 2020.

Andrew H Song, Guillaume Jaume, Drew FK Williamson, Ming Y Lu, Anurag Vaidya, Tiffany R Miller, and Faisal Mahmood. Artificial intelligence for digital and computational pathology. *Nature Reviews Bioengineering*, 1(12):930–949, 2023.

Andrew H Song, Richard J Chen, Tong Ding, Drew FK Williamson, Guillaume Jaume, and Faisal Mahmood. Morphological prototyping for unsupervised slide representation learning in computational pathology. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11566–11578, 2024.

Michaela Unger and Jakob Nikolas Kather. Deep learning in cancer genomics and histopathology. *Genome medicine*, 16(1):44, 2024.

Jeroen Van der Laak, Geert Litjens, and Francesco Ciompi. Deep learning in histopathology: the path to the clinic. *Nature medicine*, 27(5):775–784, 2021.

Hongrun Zhang, Yanda Meng, Yitian Zhao, Yihong Qiao, Xiaoyun Yang, Sarah E Coupland, and Yalin Zheng. Dtfd-mil: Double-tier feature distillation multiple instance learning for histopathology whole slide image classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 18802–18812, 2022.

Margarita L Zuley, Rose Jarosz, Bettina F Drake, Danielle Rancilio, Aleksandra Klim, Kimberly Rieger-Christ, and John Lemmerman. Radiology data from the cancer genome atlas prostate adenocarcinoma [tcga-prad] collection. *Cancer Imaging Arch*, 9(10.7937):K9, 2016.