Classification of ISUP grades from Whole Slide Images

Team Name: Vanilla Model

Clément Apavou* CLEMENT.APAVOU@ENS-PARIS-SACLAY.FR

Younes Belkada* YOUNES.BELKADA@ENS-PARIS-SACLAY.FR

Keywords: Deep Learning, Whole Slides Images, Prostate Cancer Diagnostic, ISUP grade.

1. Introduction

1.1. Problem Explication

ISUP grade prediction from Whole Slide Images (*wsi*) is an extremely challenging and exciting task. In practice, predicting correct ISUP grades will result in assisting doctors diagnoses, therefore assigning the right treatment to the patient. Throughout this *Kaggle* challenge, we are given a dataset of Whole Slide Images of prostate biopsies associated together with the ISUP grade of them and the goal of the contest is to predict accurately the latest. Can Deep Learning be efficiently used in this context? If yes, how?

In this challenge, we are given two *modalities* per patient. A *wsi* associated with the scan of the biopsy, together with the name of the data provider. The majority of *wsi* is linked with a segmentation mask corresponding to the labeled cancerous regions responsible of predicting the ISUP grades from doctors. These regions can be rich in information and help pre-defining regions of interests that will be used to predict the ISUP grade of the biopsy. Practically, given a whole slide image, a doctor first looks for regions of interests (that can be relatively small) and then use these regions to assign an ISUP grade for the *wsi*. Based on this intuition, the Deep Learning-based framework should inspire from this approach and apply a top-down approach consisting of firstly determining regions of interest, and classify the whole slide image on top of that.

1.2. Challenges

This kaggle contest contains various stimulating problems which are the following:

- **Image size and sparsity:** The *wsi* are large in size and resolution. The latest can be accessed in different resolutions, which are high by default. Therefore, naive end-to-end approach with the whole image would be hard to implement. In addition, the images are also extremely sparse, *i.e.*, containing a majority of white pixels. We tackle these issues by considering bag of non-white patches (with a certain threshold of white pixels density) instead of raw *wsi*.

* Contributed equally

- **Data distribution:** Lack of data (∼**380 images** in the training set) and a skewed data distribution for certain classes push us to manipulate the data with caution when training our models. We present in section 2, how we properly address some of this issue during the training.
- **Different data provider:** The Whole Slide Images of the dataset are provided by two different providers. We could notice that the images of these two providers are different with a different color distribution probably due to different medical imaging acquisition equipment. It is therefore necessary to ensure that the training set and the validation set have approximately equal proportions so that the model performs well on the data from both providers.

Deeper Data Exploration As explained previously, the dataset contains *wsi* from different data providers: *Karolinska* and *Radboud*. The color of *wsi* from *Karolinska* seems to be more saturated than the ones from *Radboud* creating a domain gap between the images of the two providers. Also, the quality of the segmentation masks are different. From the figure 2, it appears that the masks provided from *Karolinska* are less *fine-grained* than the ones provided by *Radboud* in figure 1. In terms of data distribution, a difference of ∼15% is observed in terms of number of instances (figure 5) on the training set and a difference of ∼26% is observed on the test set (figure 6). Overall, the data provider *Karolinska* corresponds to most of the input images data provider. *How about the labels?* From the figure 7 we observe that the training set is clearly imbalanced towards some classes, this is because some classes are rarely observed in real life. Let us explore in the next section how we precisely address these issues for better performances.

2. Architectures and Methodological components

2.1. Data preparation

2.1.1. Re-thinking the splitting strategy

As noticed in section 1.2, the classes of the training data are imbalanced but may refer to the *real* distribution of classes in real life (*i.e.* probably in the test set as well). Firstly when splitting the training data into a train-validation split, we ensure that the validation set follows the same distribution of classes as the training set. The resulting adjusted distribution of classes is reported in the figure 8. Oppositely to randomly splitting the dataset, by following this approach we make sure that the best model according to the validation set reflects the performance in real life, *i.e.* on the test set. This approach is applied for data providers, *i.e.* we make sure that the distribution of data providers is similar in the training set and the validation set (figure 9) in order for the model to fit the two different domains and for the evaluation to be unbiased.

2.1.2. Robustness across all the classes

Nevertheless, the model needs to be robust across all the classes. When training the models, we select the instances using a *random weighted sampler*, *i.e.* each instance is assigned to a weight which is relative to the number of observations of the corresponding label on the training set. Let us denote by x_i, y_i the i-th instance of the training set consisting of the

image x_i and the label y_i. The probability of sampling the instance x_i during the training loop $p(x_i)$ is obtained by the formula:

$$p(x_i) = \frac{1}{\sum_{j=0}^{N} \mathbb{1}_{y_j=y_i}} \tag{1}$$

Therefore, the under-represented classes will be more likely selected during the training procedure. This ensures that the model will be more robust when inferred using under-represented classes. We enable this feature by default, after observing that this leads to better generalization on the test set according to the experiment 7.

2.1.3. Training tricks to create patches

In order to accelerate the training, we split images into patches before the training and we save non-white patches (with a certain threshold of white pixels density) in `numpy` files, then we zip all the files and save them as a `wandb` artifact which allows us to download the patch dataset when the chosen patch size is the same in the following trainings. Some level 0 whole slide images blow up the ram of our hardware, so we use level 1 whole slide images which are smaller than level 0. All this is done automatically in our training code.

2.2. Evaluation procedure

Since the validation set follows the same distribution as the training set, it is imbalanced across some classes. Therefore we evaluate our models using several metrics that are robust against class imbalance: *weighted* F1 score, per-class AUROC score as well as global *weighted* AUROC score. We also compute the weighted recall, precision and accuracy. All these metrics are computed epoch-wise on the validation and the training set in order to monitor the performance of the models during training. We observed a gap of the AUROC score between the performances on the validation set and the testing set that we suspect may come from the fact that in the Kaggle challenge the methods are evaluated using *macro* AUROC score and not *weighted* as we do. We also report the performances of the tried approaches on the public test set using the score provided by the challenge.

2.3. Training Code & Experiments tracking

Training code Our github repository contains our implementation for this project. We built from scratch a very clean and easy-to-use training code in `PyTorch Lightning`. Different modes are available `Classification`, `Segmentation` and `Classif_WITH_Seg` which corresponds to classification using a semantic segmentation models trained with the mode Segmentation. Everything is automatic in the training code, there is no need to run additional scripts. More information about how our code works is available in the `README` file in our github repository.

Experiments tracking For tracking of our experiments, we used the very practical Weights & Biases (wandb) (Biewald, 2020) library. We logged for each run the configuration file with all the hyper-parameters and the global seed to be able to reproduce the results if necessary. The validation and train values of loss and metrics are logged over epochs as well as some images with predictions of the model and labels. This allows us to

evaluate both quantitatively and qualitatively our models during training. All experiments are available on the following link. An overview of the workspace can be seen in the figure 4.

2.4. Building our method step by step

In this section, we present how we built our final method step by step.

2.4.1. Multiple Instance Learning *(MIL)* - Learning to select the best patch

We argue that this challenge has to be tackled following a top-down approach, *i.e.* we first select the most relevant patches, then meaningful features are extracted from these patches and a classifier predict the ISUP grade directly using these features. Recently, (Pinckaers et al., 2021) proposes a framework for selecting the most relevant patch given a *wsi* for prostate cancer detection. The proposed approach, denoted as Multiple Instance Learning (MIL) consists of selecting N patches during training and select only the patch with a highest score. After selecting the *top* patch, a classification is performed using the features of this patch. For this approach, the patches are created in *static*, *i.e.* the patches are created following the protocol presented at 2.1.3 using a white pixel density threshold of 0.2, and a ResNet32 (He et al., 2015) is used for the feature extractor. According to the results reported on the table 4, this method outperforms the baseline. However, we think that this method is not that robust since it is not really clear how the task of selecting the best patch converges. Also, we believe that there is a bottleneck in this approach since the classification of the entire *wsi* is restricted to a single patch.

2.4.2. Concatenated patches are all you need

Data preprocessing on-the-fly We propose to tweak the input by concatenating N random non white patches in a stochastic order. Each time an instance of *wsi* is sampled, we select N patches with a random order to add stochasticity in the framework therefore making the model as *permutation-invariant* as possible and avoid overfitting. The final image is resized to a given resolution before passing it to the network. An example image is given at figure 10.

Fine-tuning a CNN model Then, these images are directly used for classification using large convolutional neural networks (CNNs). We benefit from the fact that these models are *permutation-invariant* since the weights of the features are shared across all pixels. Although the baseline uses a CNN to extract features, we argue that predicting the label of the *wsi* by concatenating the features more likely leads to non *permutation-invariant* model. We train our models with a random horizontal and vertical flip on the training set only to avoid overfitting. In addition, we observed that using large CNNs such as TResNet (Ridnik et al., 2021) trained on large images leads to better performance. We use the `tresnet_xl_448` hosted on `timm` library (Wightman, 2019) pretrained on ImageNet for our experiments and randomly select N patches among available non-white patches for each *wsi*. The model is fine-tuned on our task using the cross entropy as loss function and the *Adam* optimizer (Kingma and Ba, 2014). A model trained with this method allows us to obtain an AUROC of 0.92647 on the test set on kaggle. However, results are not really

reproducible due to the randomness of the selected patches. Indeed, inference on the test set with the same model does not give the same results. We tried 4 submissions and we obtained: 0.926, 0.895, 0.877, 0.908.

2.4.3. Remove randomness: voting process at test time

Since we randomly select the patches at test time, we observed that the results may vary across several tests for the same model. We decided to not fix a specific seed since we do not want to overfit into a specific configuration and generalize well. We simply tackle the randomness issue by applying a voting procedure at test time to aggregate the results between P predictions. We have observed that this leads to a more stable framework (table 1) which allows us to obtain scores around 0.861.

2.4.4. Learning to select the best patches through semantic segmentation

Idea Our proposed method consists to randomly select patches of a *wsi* to create a "summary" image with concatenated patches allowing to resume the whole *wsi*. In order to improve the results given by our method and to remove the randomness, we propose to leverage segmentation masks giving indications of cancerous regions provided with the training images to select best patches to be concatenated according to their number of cancerous pixels using a model that has been trained to segment cancerous regions.

Training of the segmentation model Note that the segmentation masks are imperfect and different across the data providers as it is observed from 1.2. Due to different labels across providers, we train a Google DeepLab V3+ (Chen et al., 2018) on three classes (background, benign tissue and cancerous tissue) by merging the labels from both providers. We first tried to train a different model per provider but results for the provider *Karolinska* were not really good due to the less detailed mask than *Radboud*. Results are available on the table 5. So, we decided to train one model for both providers to have the best possible segmentation model covering the two different domains. For the training, we use the Dice Loss (Sudre et al., 2017) as objective function which is widely used in medical image segmentation tasks. Our best segmentation model is available here, it obtains an average IoU over all classes of 0.7858 on the validation set.

Select top-k patches For each patch of the *wsi*, we get the prediction mask from our trained segmentation model, and compute the proportion of predicted cancerous pixels. We sort the patches by their proportion of predicted cancerous pixels, retrieve the *top-k* patches, permute them randomly to avoid overfitting and then create an image which is the concatenation of these patches. With this "summary" image we apply the same method described in the section 2.4.2. We argue that if the segmentation model predicted a large proportion of cancerous pixels, the patch is most likely to be suspicious and helpful for predicting the ISUP grade of the whole WSI. The contribution of segmentation to the selection of patches allows to obtain better results, on the validation set (0.8126 vs 0.8034) and also on the test set (0.8833 vs 0.8641).

3. Model tuning and Comparison

3.1. Comparing different features extractor

Through our experiments we have identified that using large models for features extraction helps for better performances. From the observations from the table 6, we think that more parameters are needed to extract the best features to predict the correct ISUP grade.

3.2. Comparing different dataset creation strategies

3.2.1. Random weighted sampler

We compare the impact of enabling the weighted random sampler exposed in the section 1 to tackle the data imbalance issue. We train our `tresnet_xl_448` with and without the weighted sampler and compare their performance on the public score. As it can be observed on the table 7, the weighted sampler allows to achieve better results. Therefore, we enable by default the weighted random sampler in our final models.

3.2.2. Impact of selecting patches using semantic segmentation

As we can see in the table 1, select top patches according to their segmentation allows to have more stable results on the test since we no longer select patches randomly. Thus, we remove the randomness that was present in the previous methods whose results were difficult to reproduce without the previously detailed voting method (section 2.4.3). Moreover, the results of this improved method are better than the previous method with the voting and more meaningful since we select the relevant patches that can be used to classify the ISUP grade of the whole *wsi*.

Method	Voting at test time	Test set
Concat. random patches	x	[0.8774, 0.9265]
	✓	0.8641
Concat. top patches (seg.)	x	**0.8833**
	✓	**0.8833**

Table 1: Studying the capacity of results reproducibilty of our two methods: Concatenate Random Patches and Concatenate top patches according to their segmentation. Backbone: `tresnet_xl_448`, dataloader with weighted sampler.

4. Final model

Our final method is described in the section 2.4.4. We combine this method with a good handling of the dataset to obtain good and reproducible results that can be found in the table 2 and 3. An example of an ISUP grade prediction with concatenated top patches with the segmentation can be seen in the figure 11.

AUROC Class	0	1	2	3	4	5	avg weighted	kaggle
Training set	0.9122	0.8331	0.8844	0.8764	0.9288	0.8538	0.8810	-
Validation set	0.9022	0.8077	0.675	0.775	0.7167	0.8759	0.8126	-
Test set	-	-	-	-	-	-	-	0.8833

Table 2: AUROC per class, average weighted on the training set and the validation set and AUROC on the test set of our best model.

	AUROC	F1	Recall	Precision	Accuracy
Training set	0.8810	0.5907	0.5980	0.6248	0.5980
Validation set	0.8126	0.5266	0.5294	0.5455	0.5294

Table 3: Several metrics of classification on the validation set and on the training set of our best model. The values of the metrics are the average weighted over all classes.

5. Important Links

- Our github repository with the code: https://github.com/clementapa/Prostate-Cancer-Image-Classification
- Our wandb repository with the experiments: https://wandb.ai/attributes_classification_celeba/test-dlmi?workspace=user-clementapa

References

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Hans Pinckaers, Wouter Bulten, Jeroen A. van der Laak, and Geert J. S. Litjens. Detection of prostate cancer in whole-slide images through end-to-end training with image-level labels. *IEEE Transactions on Medical Imaging*, 40:1817–1826, 2021.

Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1400–1409, January 2021.

Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 240–248. Springer International Publishing, 2017. doi: 10.1007/978-3-319-67558-9_28. URL https://doi.org/10.1007%2F978-3-319-67558-9_28.

Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Appendix A. Whole Slides Images

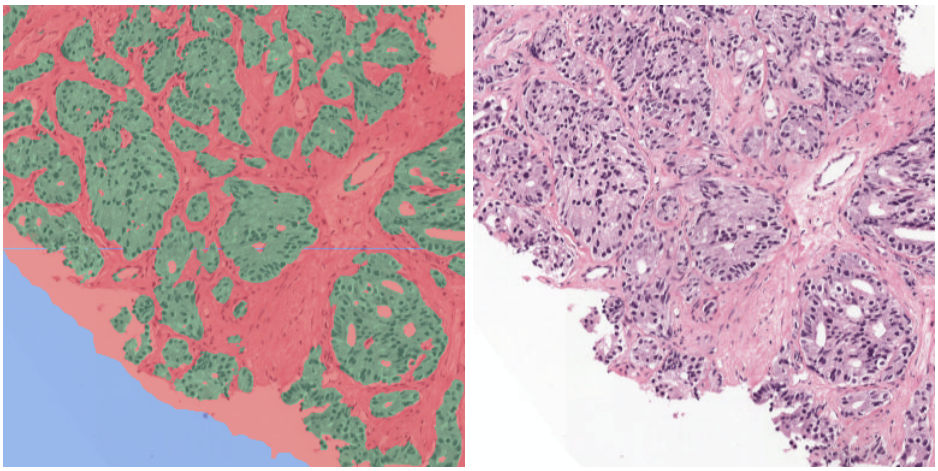

Figure 1: A *wsi* patch together with the ground truth segmentation mask, from the *radboud* data provider

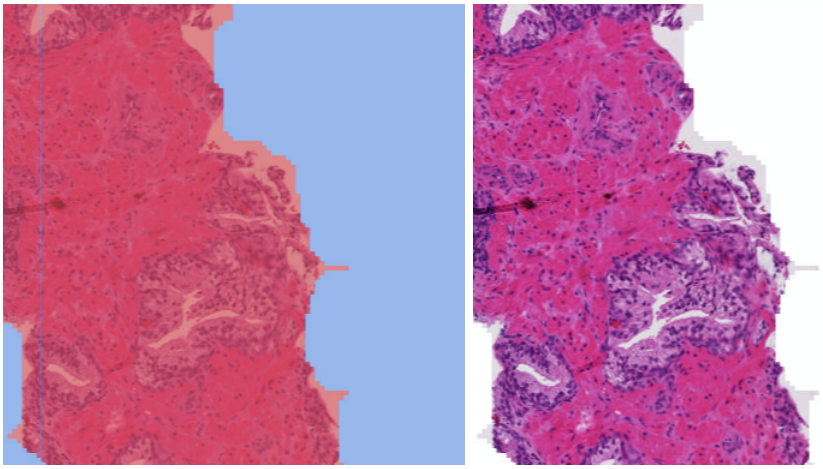

Figure 2: A *wsi* patch together with the ground truth segmentation mask, from the *karolinska* data provider

Appendix B. Baseline method

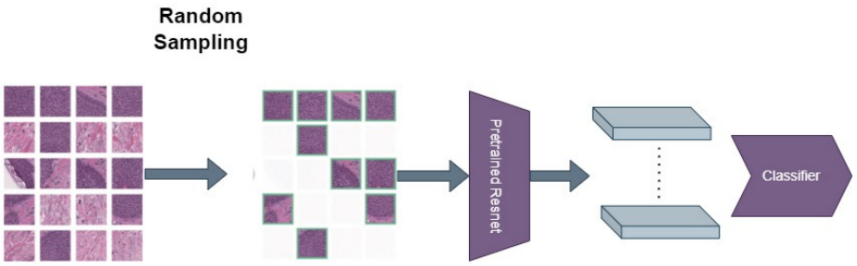

Figure 3: Overview of the pipeline for the baseline approach

Appendix C. Wandb Workspace

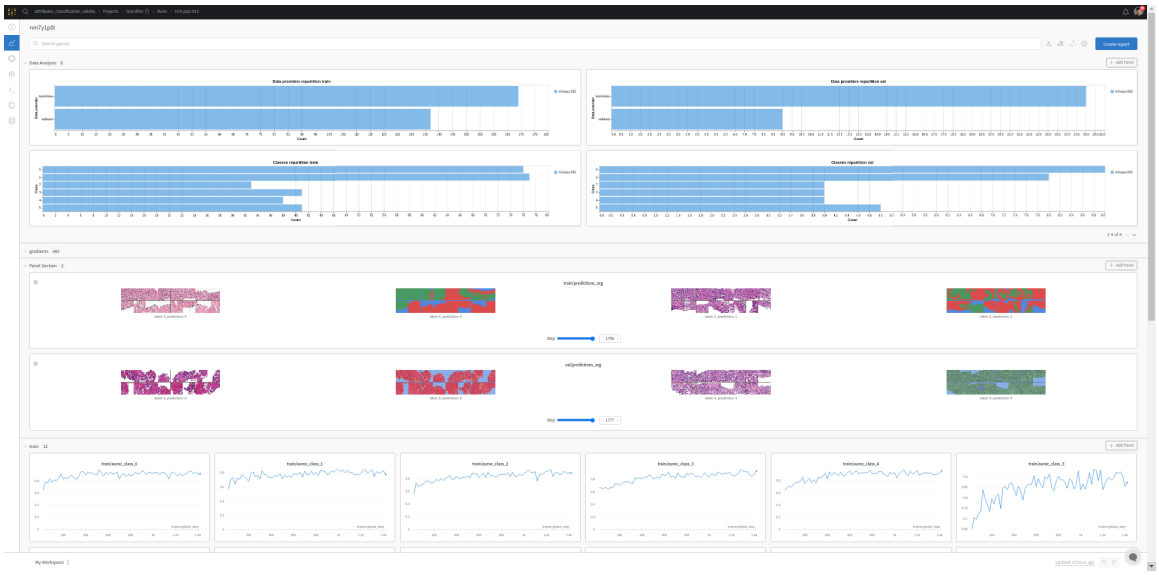

Figure 4: Overview of the wandb workspace that has been built for our project (link).

Appendix D. Data exploration

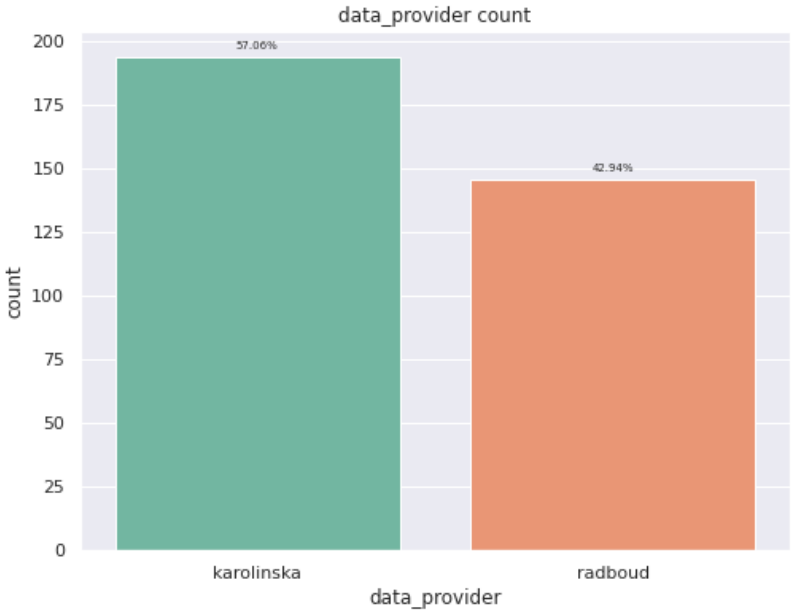

Figure 5: Data distribution on the train dataset of the challenge, sorted by the data provider

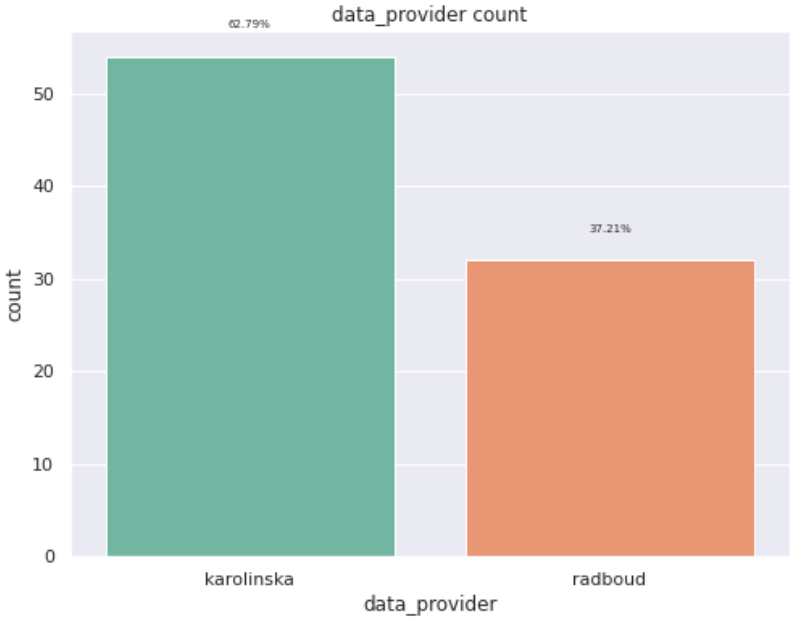

Figure 6: Data distribution on the test dataset of the challenge, sorted by the data provider

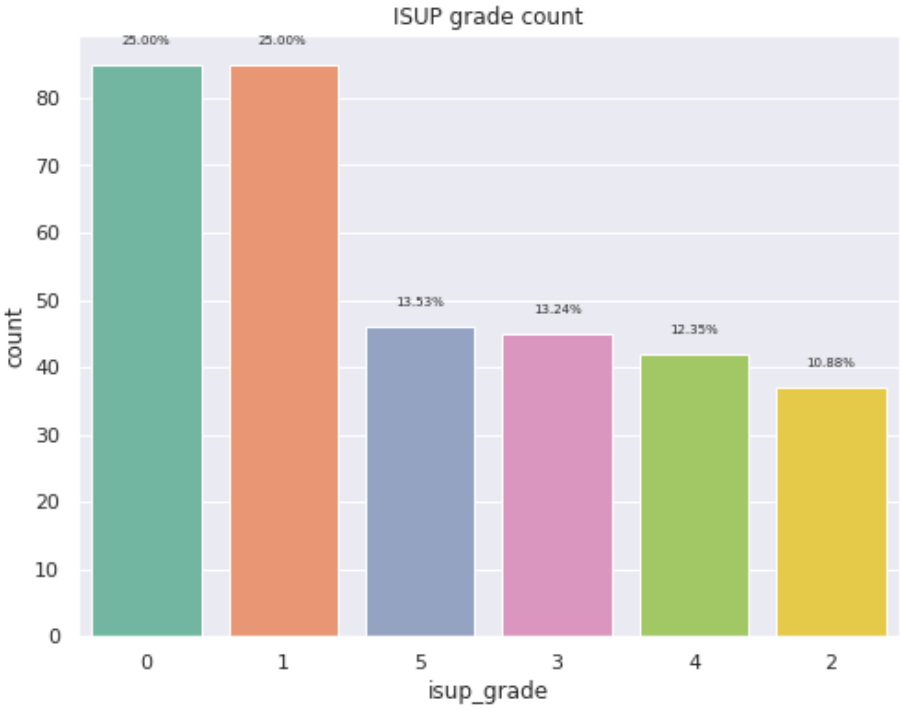

Figure 7: Data distribution on the train dataset of the challenge, sorted by the label

Appendix E. Re-thinking the splitting strategy

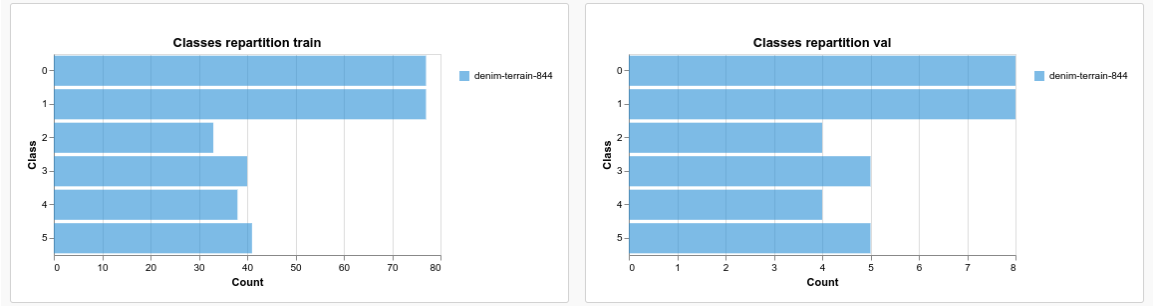

Figure 8: Classes repartition between the training set and the validation set

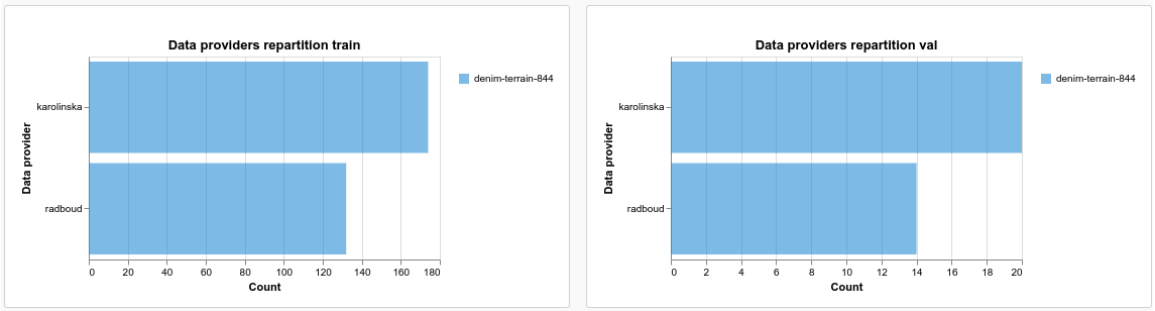

Figure 9: Data providers repartition between the training set and the validation set

Appendix F. Concatenated patches

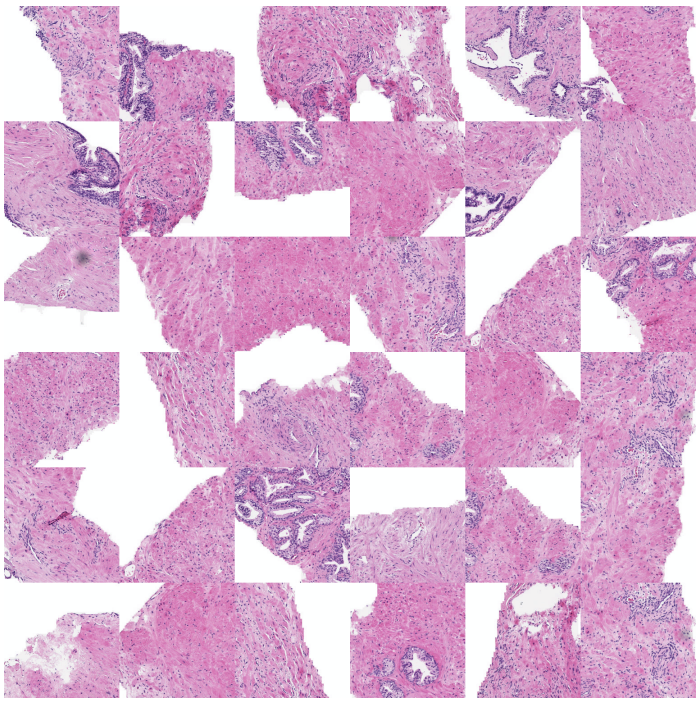

Figure 10: An example image used for training in the method 2.4.2

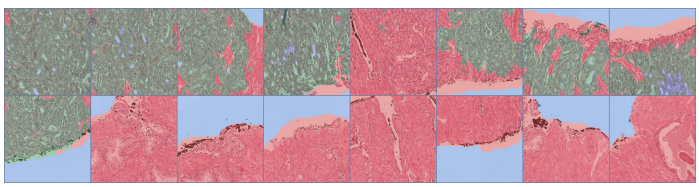

Figure 11: An example image used for training in the method with its segmentation predictions. Prediction: 4, Label: 4

Appendix G. Model tuning and Comparison

Proposed Method	Test set
Baseline	0.675
MIL (Pinckaers et al., 2021)	0.825

Table 4: Provided baseline against the method proposed by (Pinckaers et al., 2021).

Model	Features Extractor	Data provider	Patch size	Level	mIoU validation
DeepLabv3Plus	`resnet152`	All	384	1	0.7858
	`resnet34`	Radboud	512	0	0.7029
		Karolinska			0.5958

Table 5: Segmentation models trained to segment cancerous tissues.

Method	Features Extractor	# parameters	Test set
Baseline	`resnet -`	-	0.675
Concat. random patches	`resnet34`	21.3**M**	0.816
	`resnet101`	42.5**M**	0.764
	`tresnet_xl_448`	75.8**M**	0.926

Table 6: Studying the impact of chosing large CNN models. Larger models leads to better performance.

Method	Weighted Sampler (1)	Test set	wandb run
Concat. random patches	x	0.897	feasible-shape-757
	✓	**0.926**	denim-terrain-844

Table 7: Studying the impact of the weighted sampler when training our methods. Backbone: `tresnet_xl_448`.

Method	% White pixels	Image size	# patches	Test set
Concat. random patches	0.5	1024×1024	6×6	[0.8774, 0.9265]
Concat. random patches	0.2	512×512	4×4	0.888
Concat. top patches (seg.)	0.5	1024×1024	6×6	0.767
Concat. top patches (seg.)	0.2	512×512	4×4	0.884

Table 8: Studying the impact of varying the Image size as well as the number of patches. Backbone: `tresnet_xl_448`, dataloader with weighted sampler.