Proceedings of Machine Learning Research 1–6, 2025

DLMI: Histopathology OOD Classification Data Challenge Report

Corentin Chabanol[*1] CORENTIN.CHABANOL@STUDENT-CS.FR
Paul Tabbara[*1] PAUL.TABBARA@STUDENT-CS.FR
[1] *CentraleSupélec*

1. Introduction

This report documents the development of a deep learning model for a Kaggle data challenge on histopathology image classification. The goal was to classify medical images into two categories to decide wether or not they present a tumor. The main issue is that images from training, validation and test dataset are all acquired in different centers. The difference this induces can result in drop in performances from one set to another. The goal of this project is thus to make a model resilient to the conditions in which images are measured.

We explored several normalizations, augmentations, or ensemble methods to tackle this issue. Our best submission was obtained with a stronger focus on improving performance through fine-tuning techniques: we used a pre-trained ResNet50 (He et al., 2015) model with LoRA (Hu et al., 2021) (Low-Rank Adaptation) fine-tuning.

2. Architecture and methodological components

2.1. Dataset analysis

The dataset consists of histopathology images stored in HDF5 files, split into training, validation, and test sets. Those sets contain images from different centers (thus different sources), Fig 1 shows the source repartition in the training dataset: they all com from centers 0, 3, or 4. However, images for the validation dataset all come from center 1, while test images come from center 2. Therefore, because all centers measure their images in different conditions, the datasets have different images distributions that can lead to a drop in performances, that's what we observe in the baseline model, which achieves 94% of accuracy on the training data, drops to 87% of accuracy on the validation dataset and increases to 90% on the test dataset. The performances of the classifier rely on the dataset on which it is evaluated.

In figure 2, we observe that the dataset is perfectly balanced, as all things should be, with as many labels 0 than labels 1. Figure 3 shows the same trend in the validation dataset. This relieves the burden of tackling unbalanced datasets issues.

Furthermore, the following preprocessing steps were applied:

[*] Contributed equally

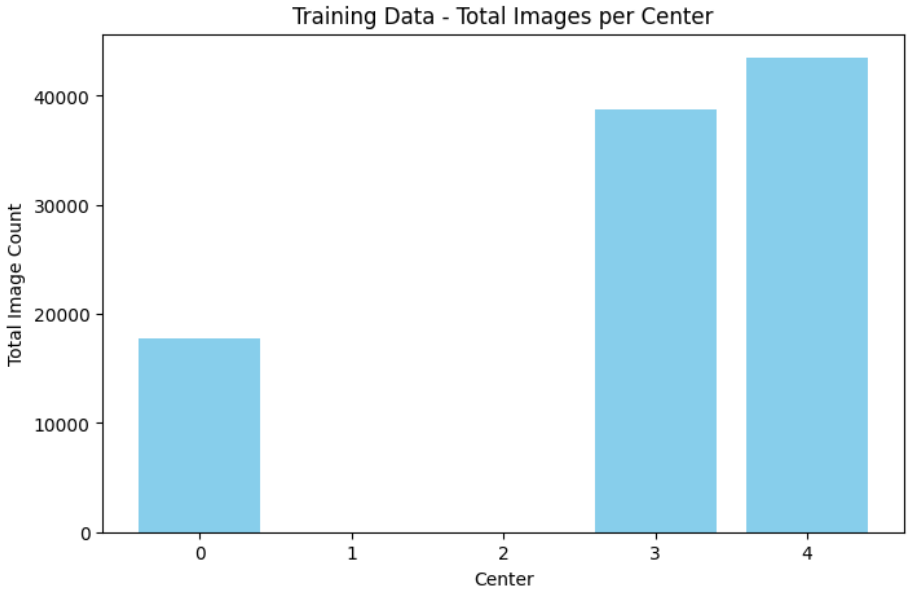

Figure 1: Repartition of sources in the training dataset.

Figure 2: Repartition of labels in the training dataset.

- **Normalization**: Images were normalized using mean $[0.485, 0.456, 0.406]$ and standard deviation $[0.229, 0.224, 0.225]$.
- **Data Augmentation**: For our best model training, random resized crops, horizontal and vertical flips, and color jitter were applied. We also experimented with augmentation consisting in random flip, rotations, and affine transform, as what we've done in the lab session on segmentation. Figure 4 illustrates such data augmentations.
- **Validation/Test Transformations**: Images were resized to 256x256 and center-cropped to 224x224.

2.2. Model Architecture

Our models are all based on pretrained models (DINOv2 (Oquab et al., 2024) and ResNet50 pre-trained on Imagenet for our best submission), to compute embeddings, on which we

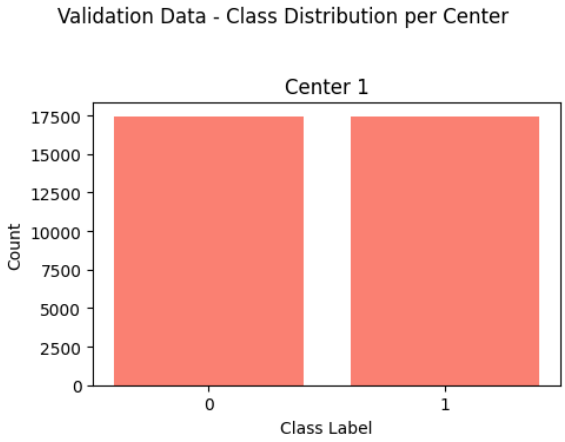

Figure 3: Repartition of labels in the validation dataset.

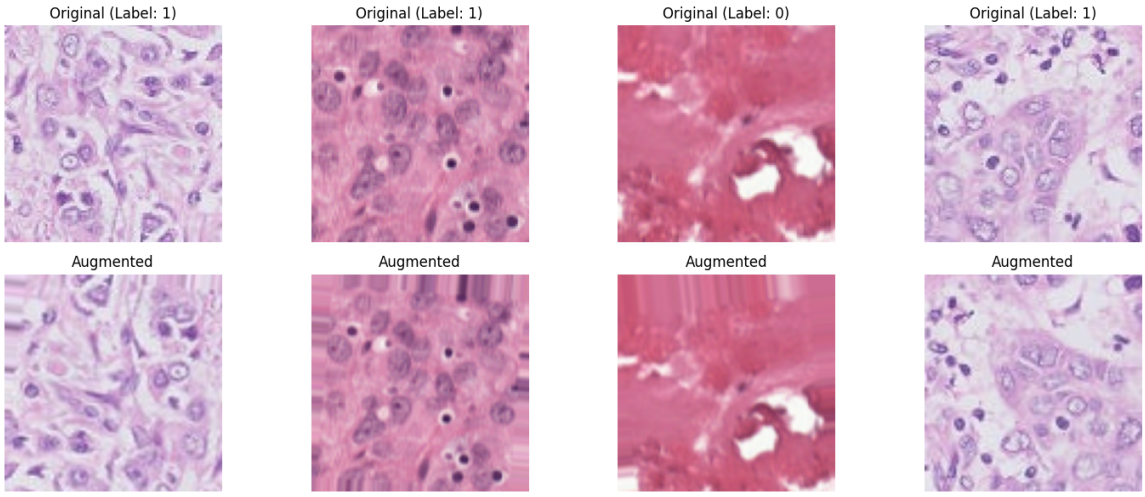

Figure 4: Example of augmented data with random flips, and affine transform.

added a classifier head consisting of a single linear layer followed by a sigmoid activation. Two fine-tuning approaches were explored:

2.2.1. Baseline Fine-Tuning

- The final fully connected layer was replaced with a dropout layer (p=0.5) followed by a linear layer with a single output.
- Only the new layers were trained, while the rest of the model was frozen.

We tested different approaches for this tuning of the last layer: we tried to train it with data augmentation, to train it on images normalized per center (that is, each center dataset is treated separately and normalized to keep relative information inside a center), and

ensemble methods for predictions (8 augmented versions of the test images where gathered for the prediction).

2.2.2. LoRA Fine-Tuning

- LoRA layers were added to all convolutional layers in the model.
- The rank of the LoRA matrices was set to 8.
- The original convolutional layers were frozen, and only the LoRA parameters and the final classifier were trained.

3. Model Tuning and Comparison

3.1. Training

The models were trained with the following hyperparameters:

- Batch size: 32
- Learning rate: 1×10^{-4}
- Weight decay: 1×10^{-4}
- Loss function: BCEWithLogitsLoss with class balancing
- Optimizer: Adam
- Scheduler: ReduceLROnPlateau with patience=5
- Early stopping: Patience=10

3.2. Results

The performance of the models on the validation set is summarized below (Table 3.2):

Model	**Val. Loss**	**Val. Accuracy**	**Test Accuracy**
Baseline (DINOv2)	0.3383	0.8745	0.9056
Baseline (ResNet50)	0.3830	0.8285	0.8981
Baseline (DINOv2)+DataAug	0.3204	0.8803	/
Baseline (DINOv2)+CenterNorm+Ensemble	0.2927	0.8935	0.9350
LoRA Fine-Tuning	0.2252	0.9391	0.9451

Table 1: Presentation of performances of the different methods used. Test accuracies are derived from the public Kaggle scores.

The LoRA fine-tuned model achieved significantly better performance, demonstrating the effectiveness of the approach.

We note that center normalization yielded good results with the ensemble method for predictions. However, we could not reproduce this method with the LoRA finetuned model because of computational resources and time constraints, as generating 8 samples per training example took a great amount of time to compute.

Moreover, because all centers had balanced samples, we thought it could be the same for the test dataset. We thus tried to adapt the threshold for classification to enforce 50% of each label in the predictions. With `Baseline (DINOv2)+CenterNorm+Ensemble`, this resulted in a test accuracy of 0.9432, which is a gain of almost 0.1 in accuracy, which is not negligible and could have helped improve our results on the LoRA model.

3.3. Conclusion

The LoRA fine-tuning approach outperformed the baseline method, achieving a validation accuracy of 0.9391. This highlights the potential of LoRA for adapting large pre-trained models to specific tasks with limited computational resources. Future work could explore different LoRA ranks and layers to further optimize performance. Adding previously described methods such as ensemble inference or balanced predictions enforcement could also be good directions for future improvements.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv:1512.03385.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, October 2021. URL http://arxiv.org/abs/2106.09685. arXiv:2106.09685 [cs].

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning Robust Visual Features without Supervision, February 2024. URL http://arxiv.org/abs/2304.07193. arXiv:2304.07193 [cs].

Appendix: Code

The code for this project implementing LoRA is available in the Jupyter notebook `lora_submission.ipynb`. Key components include:

- Custom dataset class (`HistoDataset`) for efficient data loading.
- LoRA layer implementation (`LoRAConv2d`).

- Training loops with progress tracking and model saving.

Simple data augmentation on DINOv2, as well as data visualization can be found in `data_aug_dinov2.ipynb`.
The pipeline exploring normalizing by centers and ensemble predictions is implemented in `normalize_source.ipynb`. The code to balance prediction is in `change_csv.py`.