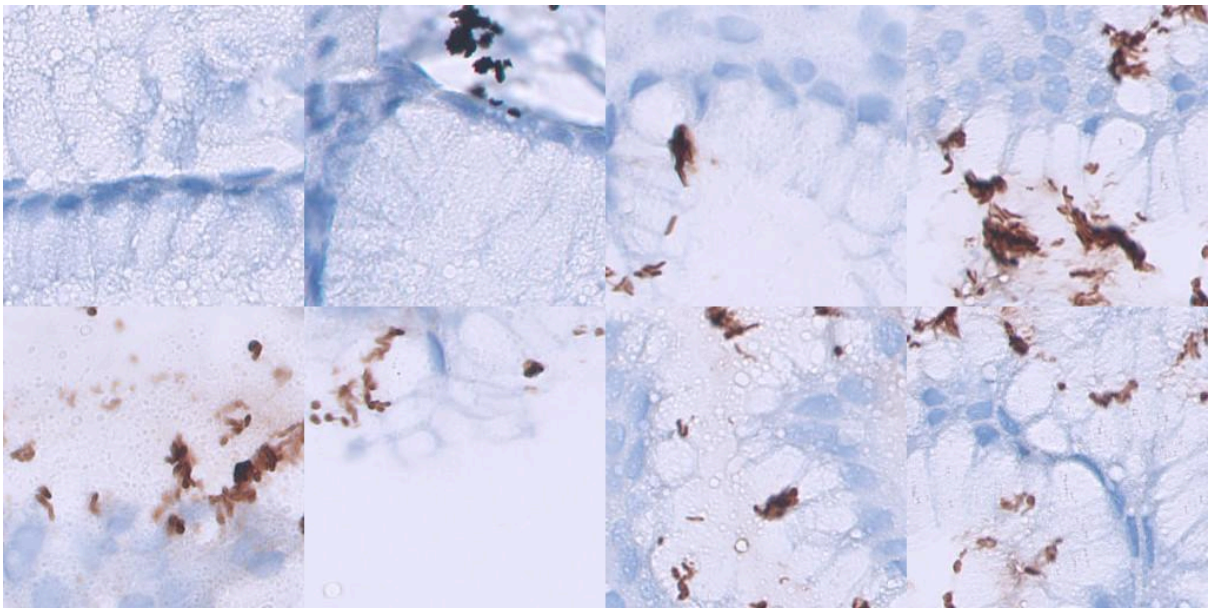
UAB
Universitat Autònoma
de Barcelona

Diagnosis of Heliobacter hypilori



Submitted by:

Marek Tutka, NIU 1

Nico Enghardt, NIU 1737026

Bogdan Mateescu, NIU 1733399

Vision and Learning

November 12th, 2024

1. Introduction

Helicobacter pylori (H. pylori) is a bacteria associated with various diseases, including gastritis, peptic ulcers, and gastric cancer. Detecting H. pylori in histological samples is essential for accurate diagnosis and effective treatment. Immunohistochemical (IHC) staining is a common technique used in pathology to identify H. pylori in tissue samples. However, the manual examination of these stained images by pathologists can be very time-consuming.

With this project we focus on developing a tool to help in the recognition of H. pylori in stained histological images. Our goal is not to replace human expertise but to enhance it, allowing for faster and more reliable diagnostic workflows.

1.1 Key challenges

The process of developing the program was not easy because of some important factors:

1. **Image Inconsistencies:** Histological images may have different staining intensities or background noise, overlapping colors and color differences
2. **Various forms:** H.plyori can appear in different shapes and sizes, some are hardly visible, causing false positives or false negatives
3. **Low number of patients**: A small patient set is sensitive to biases in classes.

1.2 Goals

The primary goal of this project is the development of a program that can detect Helicobacter pylori in histological images. More specific goals:

1. **Improve accuracy** Enhance the precision and reliability of bacteria detection and minimize false positives and negatives.
2. **Efficiency** Faster diagnoses which enable us to work on higher volumes of images
3. **Adaptability** Ensure that the model is robust and performs well across new patients never seen during training.

2. Methodology

2.0 Step 2: Dataset

For every suspected of carrying heliobacter pylori bacteria, we obtain one histological image with very large image size. The bacteria are coloured brown, while the tissue has blue color. Important patches have been selected algorithmically for each patient. Some patient's patches will be excluded from training and validation procedures, this is the HoldoutSet (). For all

other patients, some patches have been annotated by an expert with regards to their prevalence of heliobacter pylori. This AnnotatedSet (186 patients, 2902 patches, 2902 labels) has been augmented in order to achieve balance in the labels. We can use the full dataset of patches, called CroppedSet (186 patients, 186 diagnoses, 123.681 patches), for patient level diagnosis. For every patient, a medical diagnosis is given, which we will use to train and evaluate our diagnosis system.

We split the patients used in the AnnotatedSet and the CroppedSet using a 5-fold scheme, so that we obtain 5 different splits of patients which we will use to load 5 different splits for the AnnotatedSet and the CroppedSet. We will train and evaluate on each of the 5 splits while ensuring no data leakage in the whole pipeline.

2.1 Step 1: PatchClassifier training

To be able to find the bacteria in the patches, we first train a model (custom named PatchClassifier). For training we use annotated patches of the train patients, using 5-fold splitting, we will cover that in more detail later. The images are all resized (to standardize the input size for the model), normalized (using the ‘ torch.nn.functional.normalize(torch.Tensor(image))` function).

Our model is a custom CNN that contains:

- 2 convolutional layers that extract features from the input images
- Max-pooling layers that downsample the feature maps so that the load on the computer is lower.
- Output layer that provides a probability score (ranging from 0 to 1)

Using all these, the model will output a probability score (0 to 1) for the presence of the bacteria.

We choose a value of 0.01 as learning rate, with a batch size of 1500 images and 200 epochs. For every epoch, we have 2 loops:

In the first one, the input images are passed through the model to generate predictions, then the function BCELoss calculates the loss between predicted and true labels and using an algorithm called Adam optimization the model’s parameters are adjusted.

The second loop takes place after the training on each batch. Then we compute the validation loss to monitor how to model classifies unseen data.

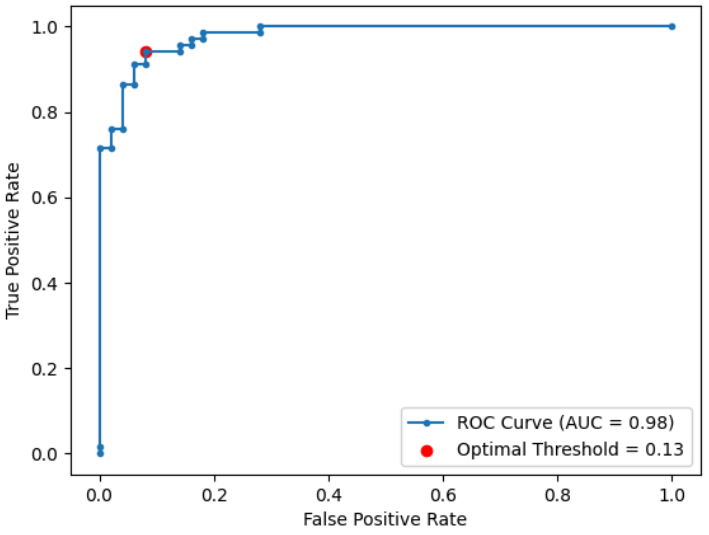
For the model training in the beginning we had issues with coming up with the correct batch size, because if the number of the epochs was too small, we would end the training too fast, leading to a not well-trained model. On the other hand, if the number was too big, we would encounter model overfitting leading to a model that does not predict the bacteria's presence very well. Therefore we have applied an optimization technique called “early stopping” that allows us to set a higher number of epochs without being afraid of overfitting. Our implementation of the mentioned stopping technique works like this: we monitor the validation loss after every epoch looking for improvement. After every epoch that had

improvement we save the model's state as a checkpoint. If there are no improvements for 50 consecutive epochs, the training is stopped and the last checkpoint is saved as the final model. Based on our test runs, we have added a warm up period of 30 epochs to make sure the model has sufficient time to learn before applying early stopping. This way we can prevent overfitting our model.

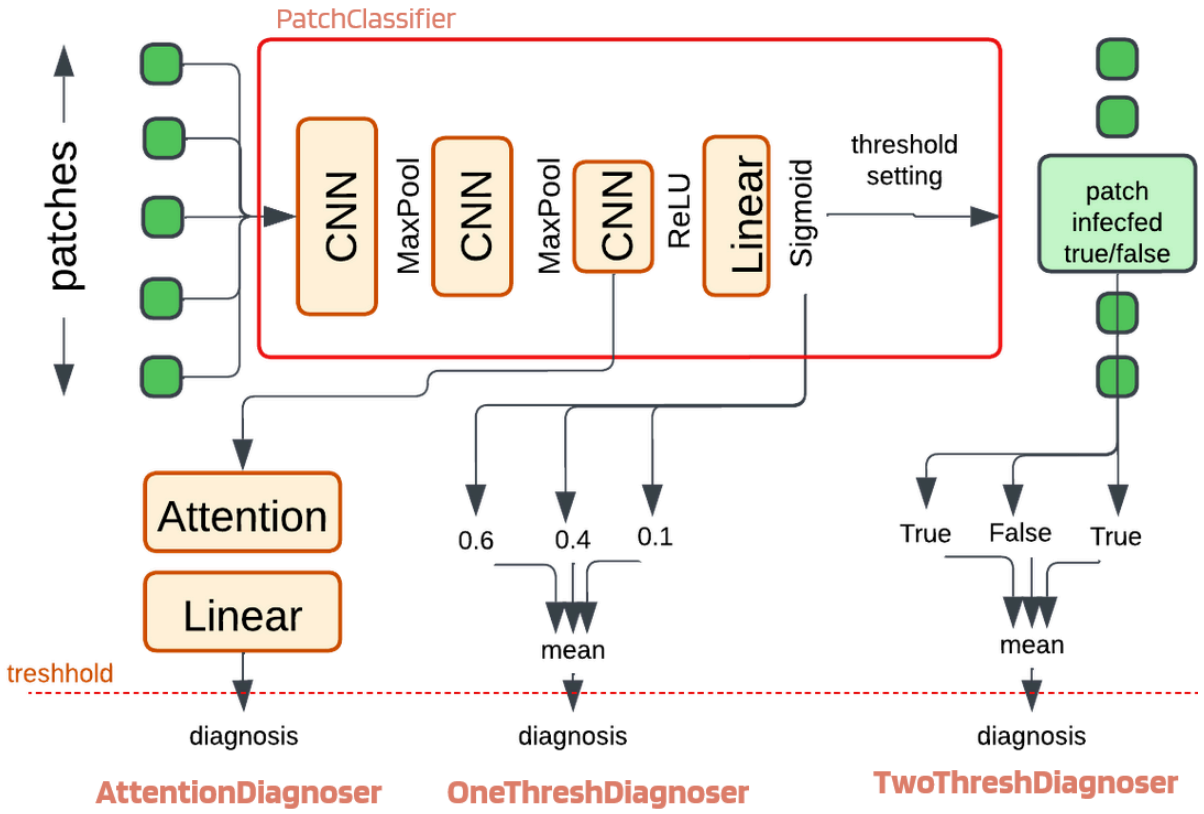
The last very important part is the usage of 5-fold cross-validation. It gives us a way more robust evaluation on how the model handles unseen data by training and validating it on different data. Our dataset is split into 5 subsets and the model is trained on 4 of them while the fifth is used for validation. This process is repeated for all folds. In the end we get a different model for each of the 5 folds.

2.2 Adaptive Tresholding

Suppose, we have a list of scores in range [0,1] and associated binary labels. For every possible threshold, we can map the scores to binary values and compute the confusion matrix with regards to the labels. For every threshold in range [0,1] we will plot the true positive rate over false positive rate and obtain a diagram like the one on the right. We then select the point closest to (true_pos = 1, false_pos = 0) as the optimal threshold. We will use this method in several points of the diagnosing.



2.3 Step 2: Diagnosing process



After training 5 different PatchClassifiers, one for each fold, we prepare 4 diagnosis systems on each of the classifiers using the training split of the CroppedSet. We will then evaluate the performance of each pipeline using the test split of the CroppedSet and compute further evaluation scores for the HoldOutSet.

2.3.1 AttentionDiagnoser

We feed all a patients patches through the PatchClassifier until the ReLU layer, resulting in 10 features for every patch. We aggregate a (features x patches)-matrix and run in through a Self-Attention Layer, which removes the patch-dimension. After forwarding through a linear layer and sigmoid activation function, we obtain a single diagnosis score for the patient.

We train the Attention- and Linear Layer on the train-CroppedSet, while fixing the PatchClassifier layers. we achieve this by precomputing the feature output of the Classifier..

2.3.2 OneThreshDiagnoser

The classifier output (sigmoided score) for all patches of a patient are averaged to generate a diagnosis score.

2.3.3 TwoThreshDiagnoser

We perform Adaptive thresholding on the train-AnnotatedSet after training of the classifier. When preparing the Diagnoser, we use the train-CroppedSet to set a threshold as follows. For each patient, we compute the tresholded classifier output and average over all patches. This averaged diagnosis score is used in Adaptive Trehsholding with regards to the patient diagosis labels.

2.3.3 DistributionDiagnoser

Because this pipeline is new, it's not included in the diagram above.

For the train-CroppedSet we compute the sigmoided scores of all patches for a patient. We us this to compute the average and further statistical moments of this score distribution. We compile them to a diagnosis score with another linear layer and sigmoid, which we train during preparation of the diagnoser.

3. Experimental Design

We run 5 versions of each of the above described pipelines using the 5-fold split. Therefore, we obtain 5 PatchClassifier models and can compute average and standard deviation of all of the following metrics:

- Accuracy (true predictions / all datapoints)
- Precision (true positives / all positive predictions)
- Recall of positive scores (true positives / all true labels)
- Recall of negative scores (true negatives/ all false labels)
- F1-score (harmonic mean of Precision and Positive Recall)
- Confusion Matrix
- Confusion Matrix / all_datapoints

We prepare the 4 Diagnosers 5-fold and compute the same metrics on their outputs.

4. Results

4.1 PatchClassifier Evaluation (test-AnnotatedSet)

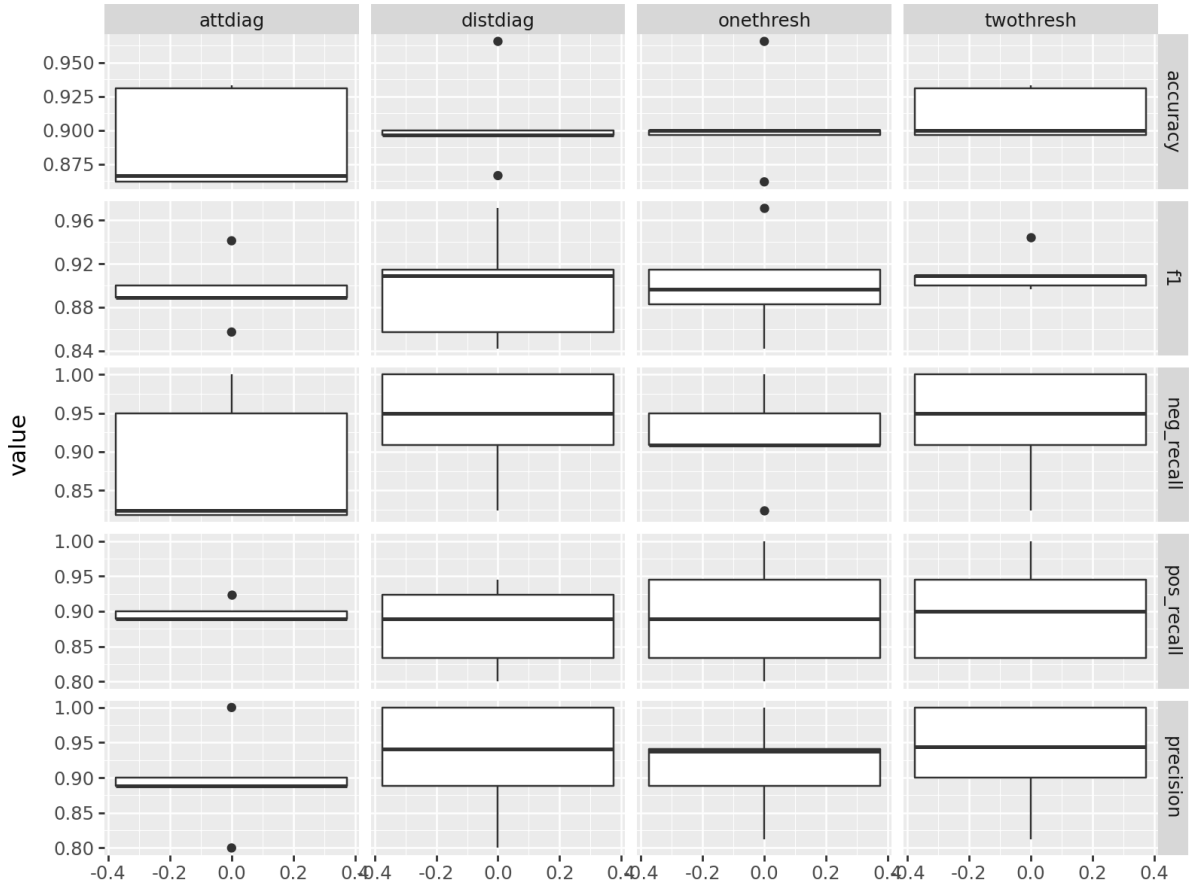
Averaged over 5 folds	PatchClassifier
Accuracy	94 % ± 4%
Precision	95% ± 2 %
Positive Recall	93% ± 7 %
Negative Recall	95% ± 3%
F1	94% ± 3%

confusion matrix	Neg Pred	False Pred
Neg truth	53 % ± 12 %	3,5 % ± 3,5 %
Positive truth	2,4 % ± 1,1 %	41 % ± 10%

4.1 Diagnoser Evaluation (test-CroppedSet and test-Holdout)

test-Cropped / HoldOut	Attention Diagnoser	OneThresh Diagnoser	TwoThresh Diagnoser	Distribution Diagnoser
Accuracy	90% ± 5% / 84% ± 2%	92% ± 3% / 85% ± 2%	92% ± 2% / 85% ± 1%	91 % ± 3 % / 84 % ± 2 %
Precision	89% ± 8% / 95% ± 3%	92% ± 6% / 93% ± 3%	93% ± 7% / 94% ± 3%	93 % ± 8 % / 95 % ± 5 %
Positive Recall	92% ± 5% / 73% ± 4%	92% ± 4% / 76% ± 3%	91% ± 4% / 75% ± 3%	88 % ± 5 % / 73 % ± 3 %
Negative Recall	87% ± 11% / 96% ± 3%	92% ± 6% / 94% ± 3%	94% ± 7% / 95% ± 3%	94 % ± 7 % / 96 % ± 4 %
F1	90% ± 5% / 82% ± 2%	92% ± 3% / 83% ± 2%	92% ± 2% / 84% ± 2%	90 % ± 5 % 82 % ± 2 %

The same metrics are displayed in boxplots here (just test-CroppedSet)



5. Discussion and Conclusions

We had some issues with utilizing the potential of the GPU cluster. In our first runs we didn't specify functions with the ".to(device)" which resulted in operations running on the CPU rather than GPU. Once we fixed that issue, one training run with batch size 700 and about 100 epochs (number estimated, because different runs had different learning curve and early stopping mechanism stopped them at different epoch) was running for 25-30 minutes for one fold, leading to the whole training with 5-folds taking 2,5 hours (new training with different fold was executed only after the previous has stopped).

In the last week after consulting the professor we have changed the way we are handling the image loading. At first in the patient class we were storing only the path to the image and when needed, our "__getitem__(image_index)" loaded the image from the path stored in the patient.images array and returned it. Using this approach we didn't use much memory, because we were storing only the path string. Loading images from the disk looks relatively fast, however when we take into account that those images were loaded at every epoch and there were thousands of images to load, it adds up. So we changed the class to preload all images when the patient object is created, therefore in the patient.images array we stored the loaded image and function "__getitem__(image_index)" simply returned the image from the array. We also changed the batch size from 700 to 1500.

As a result, one training run with batch size 1500 and about 200 epochs took only about 3 minutes for one fold, leading to the whole training with 5-folds taking approximately 15 minutes.

The metrics of the PatchClassifier training are as high as we hoped for. This confirms our CNN approach. There is little left to be optimized here, which is why we focused our efforts on different Diagnoser mechanisms. It's challenging to aggregate knowledge of an unknown number of patches into one single diagnosis, and it was not obvious to us, which would be the best approach, therefore we evaluated a family of different diagnosers.

We see no advantage of diagnosers with trained layers to the ones that only use thresholding: The AttentionDiagnoser has slightly worse performance on some metrics, while the DistributionDiagnoser is on par with the Treshold Diagnosers. Overall the ThresholdDiagnoser has the best performance, which hints to the fact that there is meaning to a patch-level threshold: Does this patch contain heliobacter pylori or not.

The system could be used now to diagnose patients in hospitals, in cooperation with doctors. Then new confidence thresholds should be defined, above which a patient is considered surely sick and below which a patient is definitely healthy. All unsure cases have to be reviewed by medical personell.

6. References

We used the Attention-Module given in Campus Virtual for the AttentionDiagnoser.