Classification of Lymphocytosis from Blood Cells

Deep Learning for Medical Imaging

Clément Grisi
École des Ponts ParisTech
grisi.clement@gmail.com

Marius Schmidt-Mengin
École des Ponts ParisTech
marius.schmidt.mengin@gmail.com

Abstract

In this report, we emphasize our work for the data challenge organized as part of the Deep Learning for Medical Imaging class. The goal of the challenge is to classify lymphocytosis from blood cells. Through iterations in model definition and data representation, we managed to get 0.96623 *balanced accuracy on the public leaderboard (rank 1).*

1. Introduction

The goal of this challenge is to determine if a patient has a lymphoproliferative disorder – a type of cancer. A manifestation of this illness is an increased number of lymphocytes, called lymphocytosis. But lymphocytosis can also be caused by other conditions such as an infection or acute stress. Thus, we must determine whether a patient with lymphocytosis has cancer or not. To this end, the challenge provides a dataset of images of blood smears for 204 patiens. For each patient, we have access to dozens of images of lymphocytes, as well as the following clinical attributes: gender, age, and lymphocyte count. If we assume that a single lymphocyte image can carry information that implies that the patient has cancer, then we can cast the challenge as a multiple instance learning problem. The metric for this challenge is balanced accuracy, which is the arithmetic mean between sensitivity and specificity. This metric is better suited for imbalanced datasets than accuracy (this challenge is imbalanced as about 70% of patients have label 1).

2. Methodology

2.1. Standard MI Assumption

Under the standard MI assumption [3, 4], *positive* patients must contain at least one instance classified as *positive*, while *negative* patients, instead, must have all their instances classified as *negative*.

A simple way to model this is described on Figure 1. Each forward + backward pass can be broken down into two consecutive stages:

- the *inference* phase: during *inference*, the models' weights are frozen. Each instance is run through the network and assigned a number between 0 and 1: the more likely an instance to indicate tumoral lymphocytosis, the closer the number assigned to 1. Once all the instances of a patient have been processed, they can be ranked according to their probability of being *positive*. For each patient, the top-1 instance is selected for the second stage: the *learning* phase.
- the *learning* phase: we now allow the models' weights to be updated. Given the standard MI assumption detailed above, one can expect the top-1 instance of a *positive* patient to be *positive*, and the top-1 instance of a *negative* patient to be *negative*. Hence, as we run the top-1 instance through the network, we assign them the label of the patient they come from and update the weights accordingly: the network adjusts its weights in order to assign high probabilities to the top-1 instances coming from positive patients, while keeping low probabilities for the top-1 instances in negative patients.

At this stage, you might be wondering "*how this model converges* ?". It is true that it is not obvious as the model is initialized with ImageNet pre-trained weights, hence there's not reason to believe the model will assign high probabilities to the true positive instances, and low probabilities to true negative instances. And actually, the model does start by assigning random probabilities to each instances!

To understand how the model gradually learns to discriminate positive instances from negative ones, we need to focus on negative patients, who play a key role in the early stages of training. Under the standard MI assumption , we know these patients must have all their instances classified

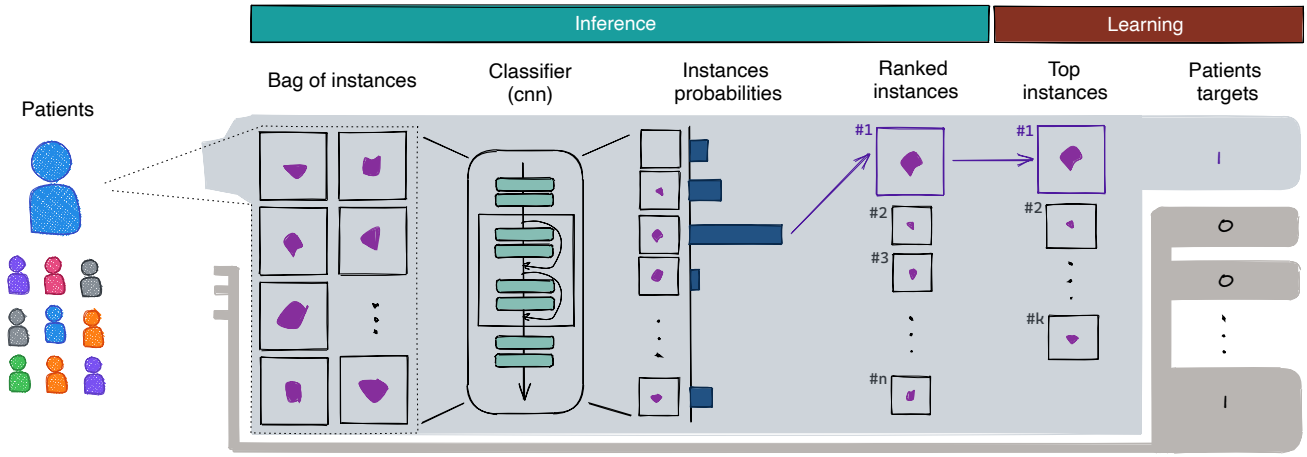

Figure 1: MIL Pipeline

as negative. At the end of the first *inference* phase, the model has assigned these instances random probabilities, which can be either high or low. Hence, the top-1 instance for negative patient is random. I claim that this doesn't matter. Let me explain you why. What's important is what happens during the first *learning* phase: for each negative patient, we have a (random) instance, which is negative. The model must adjust its weights to assign lower probabilities to each negative instances, such that at the following *inference* step, all instances that look similar to negative instances will be assigned lower probabilities. As a result, all negative instances start to be assigned low probabilities, and positive instances will srand out, with higher probabilities. At this point, the top-1 instance of a positive patient is more likely to be a true positive. And the model will adjust its weights to assign higher probabilities to this type of instance.

This simplified reasoning allows to understand how the model gently learns to discriminate suspicious instances from non-suspicious ones, and automatically infers fine-grained information from coarse-grained labels.

The standard MI assumption implies that a patient is classified positive as soon as the model identifies a suspicious instance in its corresponding bag of instances. During testing, if the top-1 instance probability is above the classification threshold τ_{class}, the whole bag is classified positive, otherwise it's classified negative. Letting the top-1 instance decide for the whole bag might be too sensitive. Immagine there exists different degrees of severity among positive instances. Looking only at the top-1 instance will correctly identify the most severe positive instances, but miss the less severe ones, which might be just as important for the final classification.

2.2. Extending the Scope of the Model

In order to deal with the limitation highlighted in the previous section, wcame up with two different idead:

- extend the standard MI assumption by introducing of a hyperparameter k_{train} which forces the model to look for at least k_{train} positive instances in positive bags, instead of only 1
- still train with top-$k = 1$, but introduce a hyperparameter k_{agg} to control the number of instances to aggregate for the final prediction.

Both ideas extend the scope of the model as we no longer let one instance decide for the label assigned to a patient. When taking more instances into account, one need to come up with a strategy to aggregate their probabilities into a single patient-level probability. As we pointed out in the previous section, applying a *max* over instances' probabilities has the adervse effect of letting the top instance decide for the whole bag. Instead, we might want to consider taking the *mean* probability of the top-k instances. Indeed, there are reason to think that n consecutive instances with high probabilities is a different biomarker than 1 instance with high probability, followed by $n - 1$ instances with probabilities close to zero. Taking the mean probability would allow to distinguish these two cases.

The experiments conducted tend to confirm that the performances of our model were sensitive to the value of both k_{train} and k_{agg} parameters. Using a fixed top-k, however, does not generalize well, since different patients have different numbers of instances, hence different numbers of suspicious instances. In order to account for this, one can introduce percentage-based k_{train} - k_{agg} values by looking at

$$\max\left(m, \lceil k\% \text{ of instances} \rceil\right)$$

where m is a new hyperparameter that controls the minimum number of instances to aggregate for the final prediction. Given the limited time of the Kaggle challenge, we didn't implent this idea, but encourage the interested reader to do so.

2.3. Taking More Instances into Consideration

According to [5], bottom-k instances are the regions "*which best support the absence of the class*" and could be just as useful as top-k instances to determine the patient label. Indeed, we can expect the bottom-k instances of a positive patient to have a different morphological aspsect than that of a negative patient, which could help refining the lymphocytosis diagnosis. Based on these findings, we decided to take the bottom-k probabilities into account as well when deriving a patient's probability, computing the average probability on the k_{agg} top instances plus the k_{agg} bottom intances.

Results presented in Section 3.3 suggest aggregating top-k with as bottom-k instances to determine the patient label doesn't help. Instead of trying to hand-craft an aggregation function to identify which instances might be useful to determine the patient label, we design a novel approach to let the model identify useful instances for classification, potentially taking all instances into consideration.

2.4. Taking All Instances into Consideration

Instead of only aggregating the model's output on a subset of instances into a single patient-level quantity, we design a custom aggreagtion module that learns how to best take instances into consideration.

Let us denote all the predictions (logits) for patient j by ℓ_i^j, $i = 1 \dots n_j$. To aggregate these predictions, we seek to define some weights $\alpha_i^j \geq 0$ such that we can compute the final prediction logit for patient j by

$$\ell^j = \sum_{i=1}^{n_j} \alpha_i^j \ell_i^j$$

(we assume that $\sum_i \alpha_i^j = 1$). The confidence score is then given by applying the sigmoid function: $y_j = \sigma(l^j)$. Two very common aggregation functions are the arithmetic mean and the maximum, where we would respectively set $\alpha_i^j = 1/n_j$ and

$$\alpha_i^j = \begin{cases} 1 \text{ if } \ell_i^j = \max_k \ell_k^j \\ 0 \text{ otherwise} \end{cases}$$

We believe that these two aggregation function suffer from different problems. Let us express the gradient of the cross-entropy loss with respect to the logits ℓ_i^j:

$$\frac{\partial L_{\text{CE}}}{\partial \ell_i^j} = \alpha_i^j \left(y^j - \bar{y}^j\right)$$

The maximum function results in sparse gradients (only one α_i^j is nonzero), as only the image that effectively generated the maximum produces a nonzero gradient. On the other hand, the mean function generates equal gradients for all images (all α_i^j are equal to $1/n_j$), meaning that for a positive patients, the "pull" generated by the gradient on negative images (if there are any) is equal to the pull applied to positive images. We developped an aggregation function that intuitively overcomes this problem. The idea behind it is similar to focal loss [6], which scales the weight so as to put more gradient on predictions that are deemed important in the context of a heavy class imbalance. Our aggregation function defines the weights α_i^j by batch normalization [7] followed by the sigmoid function:

$$\alpha_i^j = \sigma\left(\beta + \gamma \frac{\widetilde{\ell_i^j} - \mu}{\sqrt{V + \epsilon}}\right)$$

where $\widetilde{\ }$ denotes the stop gradient operation (as in focal loss, no gradients are propagated through the weights), γ and β are trainable parameters, ϵ is a small constant and μ and V are respectively the mean and variance of the logits ℓ_i^j over a batch of patients:

$$\mu = \frac{1}{n_1 + \dots + n_B} \sum_{j=1}^{B} \sum_{i=1}^{n_j} \ell_i^j$$

$$V = \frac{1}{n_1 + \dots + n_B} \sum_{j=1}^{B} \sum_{i=1}^{n_j} \left(\ell_i^j - \mu\right)^2$$

where B is the batch size. Intuitively, α_i^j is small when ℓ_i^j is small compared to the rest of the batch, i.e, when it is likely to be a negative sample. Figure 2 shows the histogram of the logits as well as the weighting function.

```
logits = conv(features) # 1 channel
weights = bn(logits.detach())
weighted_logits = weights * logits
y = sigmoid(weighted_logits.sum()/weights.sum())
```

Pseudo-code 1: PyTorch-like pseudo-code for our custom aggregation function

3. Results

We start with the simple model that only looks at the top-1 instance, then we evaluate how the different ideas successively described in Section 2 impact our performances. All our methods share a common structure: we

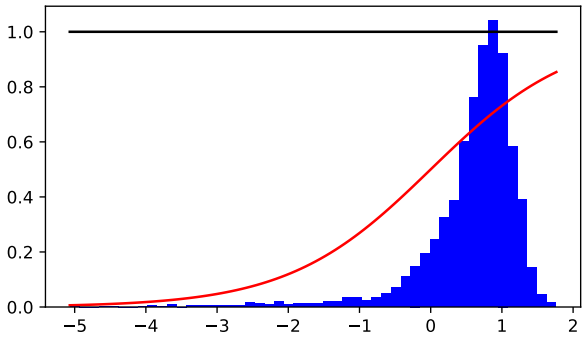

Figure 2: The distribution of the batch normalized logits over a bag is shown in blue. The red sigmoid shows the weight that is associated to each logit. The smaller the logit, the smaller the weight.

adopt a simple and explainable architecture composed of a ResNet [8] backbone followed by an aggregation module. The ablation study is solely conducted for the custom aggreagtion module we designed, as it ended up outperforming all other approaches by a significant marging.

To compare the different methods fairly, we always run 10 different trainings. Each training has its own seed for initializing the model and splitting the dataset ($50/50$ split). The 10 seeds are kept the same for each configuration. Each model is trained for 40 epochs with a batch size of 16, resulting in 5 batches per epoch. Evaluation if performed at the end of every epoch. We always optimize with Adam and a learning rate of 10^{-4}. When we report a metric for any configuration, we average the 10 best values of each training, and again average this over all 10 trainings (the average is thus taken over 100 values). We always report 95% confidence intervals.

3.1. Augmentations

As the images are centered on a lymphocytes, we always apply a center crop of size 112 to all images (training, validation and testing). Initial experiments showed that this significantly improves the accuracy and reduces the computational resources. We use vertical and horizontal random flipping. We tried to use more agressive augmentations such as affine transforms and color jittering but this heavily impaired the accuracy.

3.2. Backbone Choice

We compared three backbones: ResNet18, ResNet34 and ResNet50 [8]. We also tried a more recent backbone, EfficientNet [9], but did not obtain good results. Results for the custom aggregation are summarized in Figure 3. ResNet18 achieves the best validation balanced accuracy. This is also what we observe for other aggregation functions. We use ResNet18 for all our subsequent experiments.

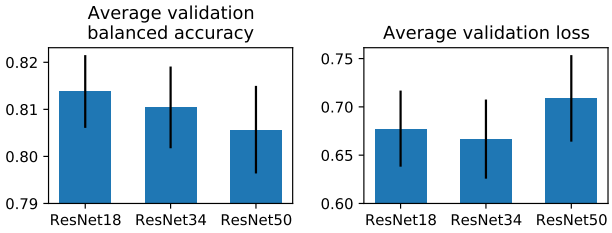

Figure 3: Impact of ResNet depth on validation loss and balanced accuracy. The black bars indicate 95% confidence intervals. Shallower seems to be better but the results are not very significative.

3.3. Working on a Subset of Instances

We present here the results we manage to achieve when training only on a subset of instances. As explained in Section 2.1, we start by only backpropagating gradients on the top instance for each patient. This perfoms relatively poorly, as we're only using a tiny subset of the data at hand (Figure 4, leftmost bar). This is why we try training training on more instances, backpropagating gradients on top-k instances instead. We mean the top-k instances' probabilities to infer the patient label. Experiments reveals $k = 10$ gives the best results.

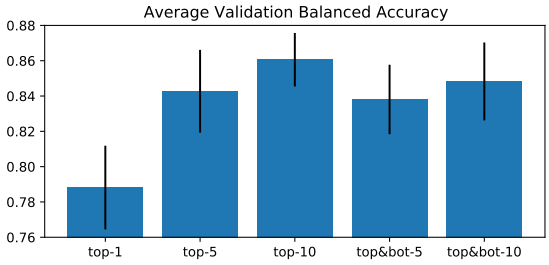

Figure 4: Training on different subsets of instances.

Eventually, we also try to take bottom-k instances into account when aggregating probabilities from the instance-level to the patient-level. However, this doesn't help achieve better performances, as shown on Figure 4.

3.4. Working on a All Instances

We start from a simple baseline and we evaluate different modifications. For each configuration, we run 10 trainings. Each training has its own seed for initializing the model and splitting the dataset ($50/50$ split). The 10 seeds are kept the same for each configuration. Each model is trained for 40 epochs with a batch size of 16, resulting in 5 batches per epoch. Evaluation if performed at the end of every epoch.

We always optimize with Adam and a learning rate of 10^{-4}. When we report a metric for any configuration, we average the 10 best validation values obtained during each training, and again average this over all 10 trainings (the average is thus taken over 100 values). The reported values thus slightly underestimate the best performance of the network, obtained by taking the top-1 value of each training instead of the top-10. We always report 95% confidence intervals over the 100 values recorded during this process. We now describe the practical implementation of the custom aggregation module discussed in Section 2.4. An overview of the pipeline is given in Figure 5.

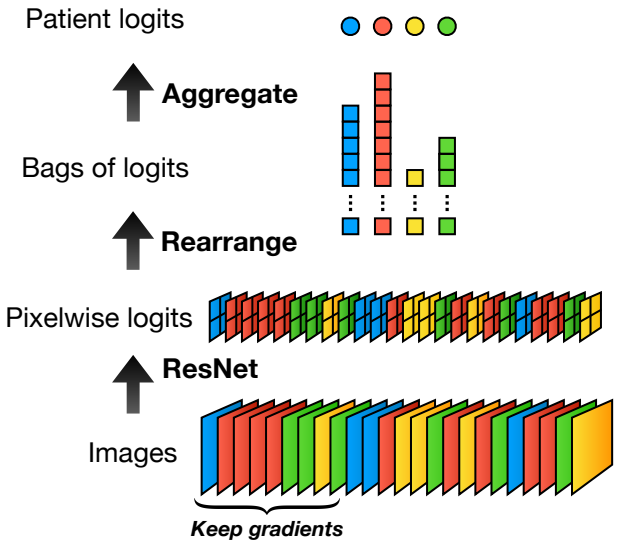

Figure 5: Custom Aggregation Pipeline

For a given batch of patients, we take all their images and stack them into a batch. This batch is passed independently the backbone, without global pooling. A 1×1 convolution with one output channel is then applied to the resulting feature maps to obtain pixelwise prediction logits. These logits are rearranged into bags of scores, each bag corresponding to all prediction for one patient. Finally, the aggregation module transforms each bag of scores into a final patient prediction.

It is common that the features are aggregated before being linearly mapped to the final prediction score. In contrast, we apply the aggregation after reducing each feature to a single prediction logit. We did not find this to impact the results (and in the case where the aggregation function is linear, both methods are equivalent). Doing so makes the model more interpretable as each pixel (in the feature space) of each image can be given a score.

As the number of images passed through the backbone is often large, we keep the gradients for only a random subset of these images. Pseudo-code for this is provided by Algorithm 1.

```
B, 3, H, W = image_batch.shape
max_forward_size = 512
max_backward_size = 512
features = []
i = 0
# forward images without keeping gradients
with torch.no_grad():
while i < B - max_backward_size:
j = min(i+max_forward_size,
num_images-max_backward_size)
features.append(backbone(images[i:j]))
i = j
# forward image with gradients
features.append(backbone(images[i:]))
# Note: the images are supposed to be shuffled
# so that the subset for which the gradients
# are kept is random.
```

Pseudo-code 2: PyTorch-like pseudo-code for propagating the gradients through only a subset of the images, in order to not run out of GPU memory.

Then, we experiment by no longer taking all the images of each patient and aggregate their logits, but only using a small random subset of image per patient during each training step. We hypothesize that selecting only a small subset of images provides some regularization to the training process. Indeed, assume that for a given positive patient, one image is easily identified by the network as positive. In this case, the network can ignore the other images. By selecting only a small random subset of images, we avoid this problem, as the easy positive image has a high probability of not being selected. Note that we implicitely make the assumption that in the case of a positive patient, a large number of images contains positive information. This idea is similar to hard negative mining, and could be called "easy positive avoidance".

Our winning submission was obtained with this strategy but Figure 6 shows that it actually worsened the results when evaluated over 10 runs [1].

4. Conclusion

In this report, we highlight the main process we have chosen to address the data challenge organized as part of the Deep Learning for Medical Imaging course. An emphasis over the iterations we followed for the assumptions, representations of the data set and models enables us to pinpoint the main challenges to be able to correctly classify be-

[1] When training with 10 images per patient per batch, we trained for 120 epochs instead of 40, but performed evaluation only every third epoch (so that the number of evaluations is kept the same).

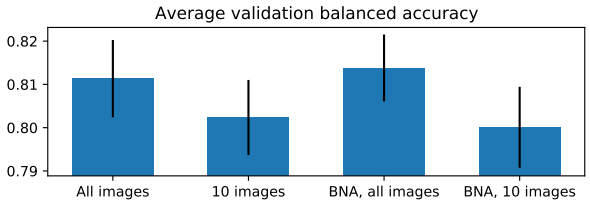

Figure 6: The performance when training with subsets of 10 images was generally worse.

tween reactive and tumoral lymphocytosis. Our best model reached 0.96623 score on the public academic leaderboard.

References

[1] Le Hou, Dimitris Samaras, Tahsin M. Kurç, Yi Gao, James E. Davis, and Joel H. Saltz. Efficient multiple instance convolutional neural networks for gigapixel resolution image classification. *CoRR*, abs/1504.07947, 2015.

[2] Boris Babenko. Multiple instance learning: algorithms and applications. 2008.

[3] Thomas G. Dietterich, R. Lathrop, and Tomas Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell.*, 89:31–71, 1997. 1

[4] Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998. 1

[5] Charlie et al. Saillard. Predicting survival after hepatocellular carcinoma resection using deep learning on histological slides. *Hepatology*, 72, 02 2020. 3

[6] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007. IEEE Computer Society, 2017. 3

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. 3

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 4

[9] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. 4

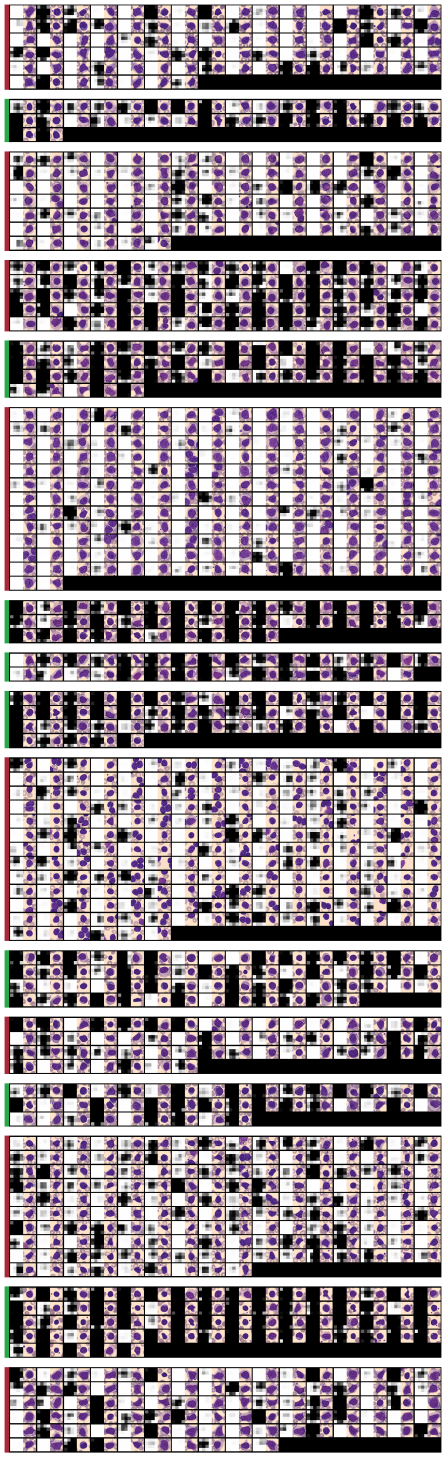

Figure 7: Predictions and images for 16 patients from the validation set. The patients where not cherry-picked. For each patient, a red left border indicates a positive ground truth label while green indicates a negative one. For each lymphocyte image, we provide on the left the 4×4 predictions of the network. White means a score of one and black a score of 0.