Prostate Cancer Diagnostic using Histopathology Images
Team GRIG

Nicolas Grevet[1] NICOLAS.GREVET@STUDENT-CS.FR
Maxime Righini[1] MAXIME.RIGHINI@STUDENT-CS.FR
[1] *Centrale-Supélec*

1. Introduction and background

Prostate cancer (PCa) is responsible for the death of 350,000 men per year. Thus, it is very important to develop more precise diagnostics.

The diagnosis of PCa is made by a pathologist and scored according to the Gleason classification system. The principle is to examine tissue samples, and to find and classify cancerous tissue into so-called Gleason patterns (3, 4, or 5). After assigning a Gleason score to the biopsy, it is converted into an ISUP grade on a 0-5 scale.
The ISUP grade is very important in determining how a patient should be treated: if it is underestimated, one may not treat a patient suffering from cancer, and if it is overestimated, one could deploy treatment resources unnecessarily.

The main issue with this scoring system is that it is very subjective: two different pathologists can easily assign a different ISUP grade to the same slide.
Thus, it seems very interesting to develop a deep learning model that would allow to bring an objective opinion on a patient's case, and to support the experts in their decision.
In this context, we are given 340 scans of prostate biopsy samples and their ISUP grade assigned by two providers (Figure 1). The goal is to use these 340 samples to predict the ISUP grade of 86 slides for which the ISUP grade is unknown.

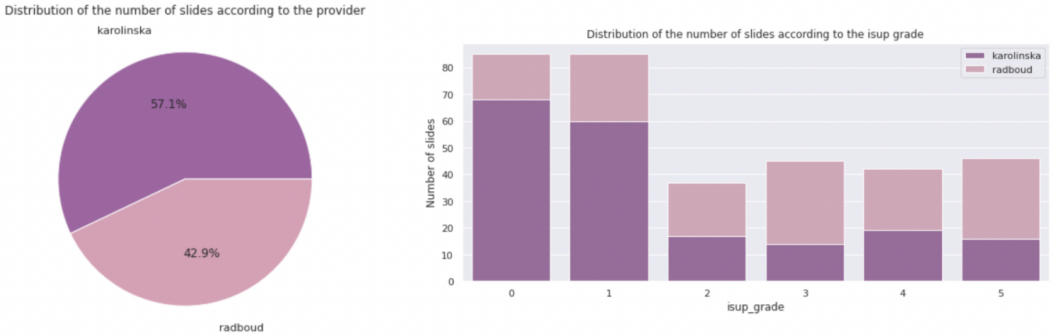

Figure 1: Some insights on the data

2. Processing of the Histopathology Images

The images that are provided to us are extremely detailed, and of very diverse sizes. Moreover, a large part of the image is occupied by background and not by tissue (Figure 2).

Figure 2: The slides are very large and of various sizes

It is unthinkable to use the images as is in a deep learning model. So we had to process them, with two goals in mind: reshape the data so every image is a square of the same size, and eliminate as many pixels as possible that do not contain tissue.

To do this, we have drawn heavily on a concat tile pooling method that we found on kaggle.
The idea is to cut the slide into several square tiles of much smaller size, and to keep only the N tiles containing the most tissue. We have adapted this by adjusting the size of the tiles to each slide (depending on the amount of tissue present).
We tested several parameters and chose to keep 100 tiles per slide. The method is efficient on all slide sizes (Figure 3).

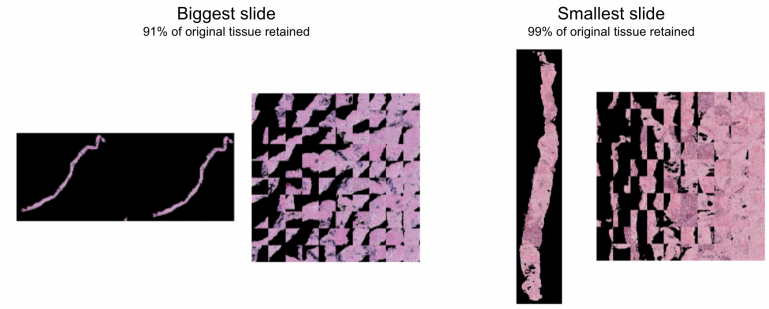

Figure 3: Tile preprocessing on the biggest and smallest slides provided to us

To make all processed images the same size, we must resize the output tiles. Keeping the resolution as high as possible gives better results, but we are limited in computing power.

So we decided to use tiles of size $(128, 128)$ (so an image of size $(1280, 1280)$) which was the best compromise.

3. Prediction pipeline

3.1. Selected model

This is an image processing problem, and we have very little data. So we decided to use transfer learning.
The images to be processed by the model are very large and we are limited in computing power: we had to find a compromise between the performance of the model and its size.
After testing several models, the best performing one that does not crash the google colab RAM is efficientnet-b0. This model still performs well when compared to other much larger models (Figure 4).

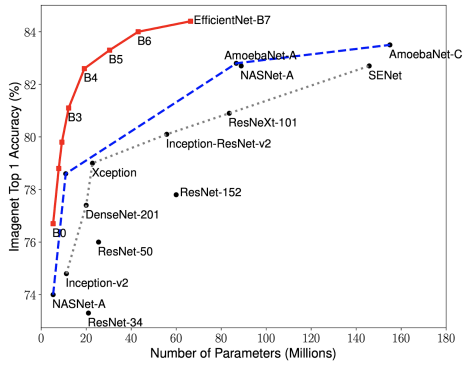

Figure 4: Performance and size comparison of different transfer learning models

3.2. Generators and data augmentation

The data is much too heavy for us to use all at once in our model. So we had to use generators. Also, since we have very little data, it is important to use data augmentation. We decided to apply rotations and symmetries both at the tile level and at the full image level on the train data (Figure 5), and no augmentation on the validation data.

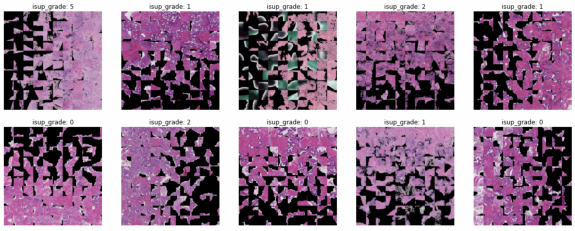

Figure 5: Illustration of the output images of the generators we have created

We have used this approach for all our prediction notebooks.

3.3. Label cleaning

The challenge instructions state that the labels are not perfect, so we wanted to come up with a method to try to identify those images with a wrong label.

The general idea is to split the training data into five folds, and for each of these fold train an efficient-b0 regression model four times (three folds for training and one fold for validation each time). We then predict the ISUP grade of each image of the fold by averaging the predictions of the four models trained on the other folds (in fact this is cross-validation). Then, if the label and the prediction are "too far" from one another, the label will be considered incorrect.

We chose a regression model to have some flexibility on the threshold to be used to determine incorrect labels.
To illustrate, let's say we set a threshold at 1.7. If a label 2 is predicted at 2.8, we keep it ($2.8 - 2 = 0.8 < 1.7$). If a label 4 is predicted at 1.3, we do not keep it ($4 - 1.3 = 2.7 > 1.7$). We precisely chose a threshold of 1.7 because it eliminates a little more than 10% of the training data, which we think is coherent.

3.4. Prediction

For our prediction on the test set, we decided to train four models over a five-fold cross-validation:

- A regression model on the complete dataset,
- A regression model on the dataset without the slides eliminated in the previous part,
- A classification model on the complete dataset,
- A classification model on the dataset without the slides eliminated in the previous part.

In all cases, we still use the efficientnet-b0 pre-trained model. In the case of regression, we use the loss mean squared error, and in the case of classification the binary cross-entropy with logits.

The last step is to combine the predictions and round them up.
We noticed during the label cleaning step that the models tend to underestimate the ISUP grade. Thus, we customized our rounding as follows: if the prediction is above $x.35$, we round up. Otherwise we round down. This rounding allowed us to slightly improve our results.

Here is a summary of our noteworthy predictions :

Table 1: Noteworthy predictions

Model	**Score on the kaggle public leaderboard**
Mean of all predictions	0.87179
Mean of predictions on full dataset	0.86538
Mean of predictions on clean dataset	0.88461
Best combination found	0.89102

We managed to exceed this best score, but the predictions were in float form instead of integers, and we decided to exclude these predictions.

4. Conclusion

This challenge allowed us to discover the difficulties of processing very high resolution images. This pushed us to find an uncommon preprocessing pipeline, and to adapt to the limitations of computing power.

We could certainly have achieved a better score with more data or more computing power. We also limited ourselves to the weights of the pretrained efficientnet-b0 model, we could perhaps have achieved a better score by customizing the model a little more (which would have lengthened the already very long calculation times).