Automatic Gleason Grading

Luca Pagano, Ernesto Natuzzi

June 2, 2023

POLITECNICO
MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

Abstract

Prostate Cancer (PCa) is the sixth most common and second deadliest cancer among men worldwide. The aggressiveness of prostate cancer is measured by Gleason grading, a system based on the appearance of cancer cells. It is usually performed via visual inspection (with a microscope) of the prostate tissue by expert pathologists. However, this is a time-consuming task and suffers from very high inter-observer variability. Automatic computer-aided methods have the potential for improving the speed, accuracy, and reproducibility of the results. This challenge aims at the automatic Gleason grading of prostate cancer from H&E-stained histopathology images through the use of a neural network, performing multiclass semantic segmentation.

1 Analysing the Dataset

Data used in this challenge consists of a set of tissue micro-array (TMA) images. We have two datasets: one having labels, for training purposes, consisting of 244 patients, and one without them, to test them, made of 86 images, . Each TMA image is annotated in detail by several expert pathologists. Most of the images feature 6 labels, each interpretation of an experienced doctor. Label pixels range from 0 (background) to 6, where 1-6 represents the Gleason Grading. Then we proceeded to rename the entire dataset with the following convention:

- maps: `slideXXX_coreYYY_classing_nonconvex.png` → `sXXX_cYYY_MAPn.png`
- train images: `slideXXX_coreYYY.jpeg` → `sXXX_cYYY.png`

Then we checked for a size mismatch between the images and their corresponding map in the dataset. In the end, we came with the following results:

1. The maps/images vary in size, with the dominant one remaining at 5120x5120.
2. All train_imgs have at least one map (usually 5, 4, or 6).
3. The corresponding label of an image has the same resolution.
4. In all mask directories, except for the fifth doctor's directory, there are corresponding images: `s007_c137` and `s007_c145` that do not have a respective image.

Therefore, we decided to delete those maps, since we don't know what they refer to.

2 Preprocessing

2.1 Combining the masks

Since we have 6 different mask for each training image, each one made by an expert pathologist, we have decided to combine them through the use of the STAPLE algorithm, implemented in the SimpleITK library.

STAPLE [3](Simultaneous Truth and Performance Level Estimation) is an algorithm used in medical image processing, specifically in the field of medical image segmentation. It is commonly used for combining multiple segmentation results obtained from different algorithms or observers to improve the overall accuracy and reliability of the segmentation.

The primary goal of STAPLE is to estimate the true segmentation of an image while taking into account the performance characteristics of the individual algorithms or observers. It assumes that each algorithm or observer has a certain level of performance, which may vary in terms of accuracy and reliability.
Below, a comparison between the six masks and the STAPLE one.

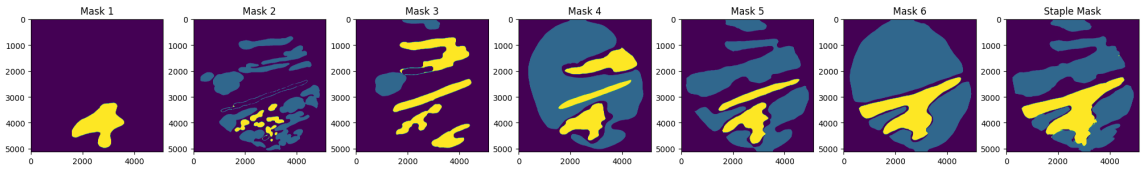


Figure 1: Comparison between the original maps and STAPLE

In case of "undecidedLabel", a pixel value assigned when the algorithm doesn't know how to combine the pixel values, our code fills the information by copying the pixel from the *Map1* directory. In addition, we changed the mask label pixels from "0, 1, 3, 4, 5, 6" to "0, 1, 2, 3, 4, 5"

2.2 Dividing into patches

To fit server hardware we'll apply some transformation to our images. Our first approach was to resize the images to 2048x2048 and divide them in 64 patches, with an overlapping percentage of 25%. The patches approach was inspired from a paper [1]which adoperates the same procedure.
However, we noticed the model could not learn from the labels due to the

lack of context caused by the excessive fractioning of images.

At the end our strategy was to:

1. Resize the images to a 1024x1024 format
2. Divide them into patches of 512x512, with 50% overlapping

Having overlapped patches ensures that object boundaries that may not align perfectly with patch boundaries are captured, and they provide additional context, helping the model understand spatial relationships between regions and improving segmentation accuracy.

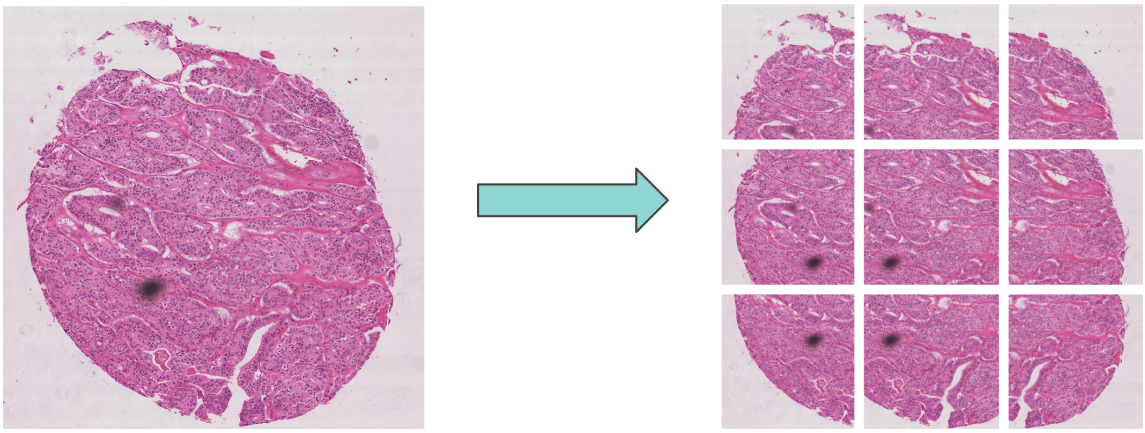


Figure 2: `s001_c003` patient image to patches (50% overlapping)

3 Training the model

3.1 Dividing the dataset

Firstly, we split the portion of dataset having respective labels with the following convention:

1. 80% Training
2. 20% Validation

Since each image consists of 9 patches, the training dataset has a total of 1755 images, and the validation dataset a total of 441.

3.2 Model Choice

Considering our hardware limitations, for the model we decided to adopt a UNet, with an EfficientNetB4 for the backbone.

EfficientNet [2] uses a technique called compound coefficient to scale up models in a simple but effective manner. Instead of randomly scaling up width, depth or resolution, compound scaling uniformly scales each dimension with a certain fixed set of scaling coefficients. Using the scaling method and AutoML, the authors of efficient developed seven models of various dimensions, which surpassed the state-of-the-art accuracy of most convolutional neural networks, and with much better efficiency.

Figure 3 shows the performance of EfficientNet compared to other network architectures. The biggest EfficientNet model EfficientNet B7 obtained state-of-the-art performance on the ImageNet and the CIFAR-100 datasets. It obtained around 84.4% top-1/and 97.3% top-5 accuracy on ImageNet. Also, the model size was 8.4 times smaller and 6.1 times faster than the previous best CNN model. It obtained 91.7% accuracy on the CIFAR-100 data set and 98.8% accuracy on the Flowers dataset.

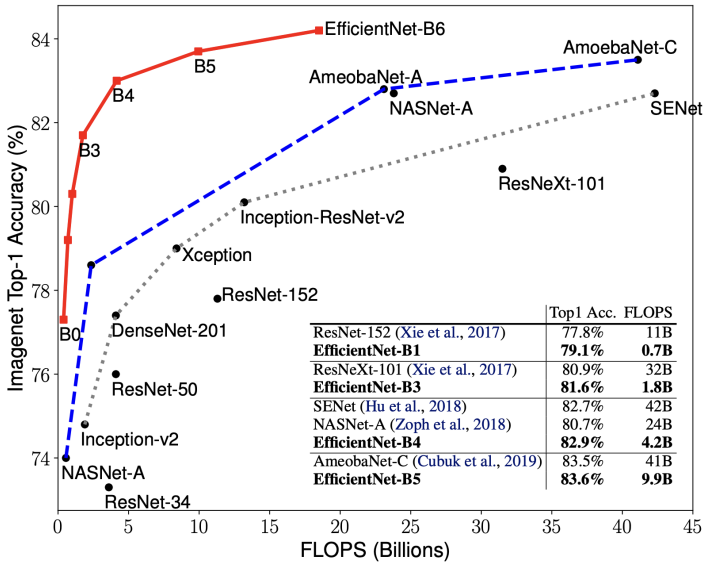


Figure 3: Comparison of the models

After some comparison with other alternatives, we opted for **EfficientNetB4** since it is the best compromise between performance and accuracy as showed in Figure 3.

For the model we used the Segmentation Model library, providing in addition useful metrics and losses.

3.3 Loss Choice

After various experiments to evaluate the impact of parameters in data augmentation, we focused on choosing the best possible loss function. Being the class dataset not perfectly balanced, we believed that the CategoricalFocalLoss, which put more emphasis on less represented classes, combined with a Dice Loss, could be beneficial for the overall model performance.

3.3.1 Difference between losses

Categorical Cross-Entropy loss is traditionally used in classification tasks. It quantifies the degree of uncertainty in the model's predicted value for the variable. The sum of the entropies of all the probability estimates is the cross entropy.

$$Entropy = -p_i log_b(p_i) \tag{1}$$

$$CrossEntropy = -\Sigma_{i=1}^{i=n} Y_i \ log_b(p_i) \tag{2}$$

Where Y is the true label and p is the predicted probability.

Class imbalance inherits bias in the process. The majority class examples will dominate the loss function and gradient descent, causing the weights to update in the direction of the model becoming more confident in predicting the majority class while putting less emphasis on the minority classes.

Focal loss focuses on the examples that the model gets wrong rather than the ones that it can confidently predict, ensuring that predictions on hard examples improve over time rather than becoming overly confident with easy ones.

Focal loss achieves this through something called *Down Weighting*. Down weighting is a technique that reduces the influence of easy examples on the loss function, resulting in more attention being paid to hard examples. This technique can be implemented by adding a modulating factor to the Cross-Entropy loss.

$$FocalLoss = -\Sigma_{i=1}^{i=n} (i - p_i)^{\gamma} \ log_b(p_i) \tag{3}$$

Where γ (Gamma) is the focusing parameter to be tuned using cross-validation.

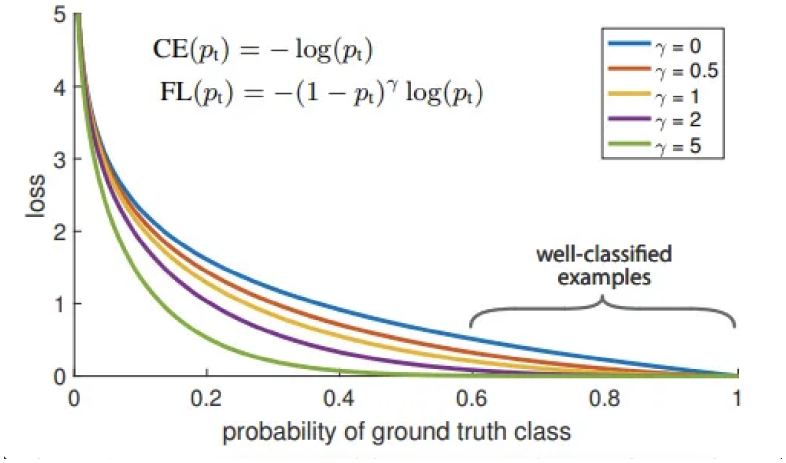


Figure 4: Focal Loss as the variation of gamma

In the end, after some experiments with the CategoricalFocalLoss, the DiceLoss, and the combination of the two, with or without weights, the best overall performance was obtained with the state of the art **Categorical-CrossEntropy Loss**, since the class were not so imbalanced to justify using other losses.

4 Results

4.1 Training History

In the end, we trained for 30 epochs with batches of 4 sizes and included data augmentation with the following parameters in the data augmentation for the `ImageDataGenerator`.

Algorithm 1 Data augmentation

```
data_gen_args = dict(shear_range=0.05,
                     zoom_range=0.2,
                     horizontal_flip=True,
                     vertical_flip=True,
                     fill_mode='reflect')
```

Algorithm 2 Training

```
def create_model():
    model = sm.Unet("efficientnetb4", classes = num_classes, activation =
```

```
3
4
5       total_loss = sm.losses.CategoricalCELoss()
6
7       metrics = [sm.metrics.IOUScore(threshold=0.5),
8       sm.metrics.FScore(threshold=0.5)]
9
10      model.compile(
11          'Adam',
12          loss = total_loss,
13          metrics= metrics,
14      )
15
16      return model
17
18 model = create_model()
19
20 callbacks = [
21      keras.callbacks.ModelCheckpoint(
22      '/content/drive/MyDrive/checkpoint/saveBestModel.h5',
23      save_best_only=True, mode='min'),
24      keras.callbacks.ReduceLROnPlateau(),
25 ]
26
27 history = model.fit(train_dataset, steps_per_epoch=steps_per_epoch,
28      epochs=num_epochs,
29      callbacks=callbacks,
30      validation_data = validation_dataset,
31      validation_steps = valid_steps
32      )
```

In the end, we produced the results of Figure 5, reaching overfitting around the 17th epoch.

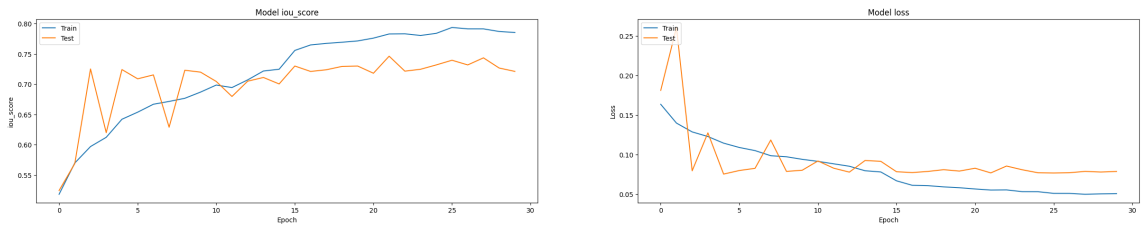


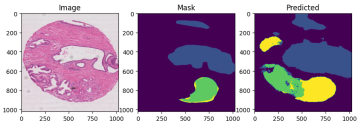
Figure 5: Training history

4.2 Comparison between original and predicted

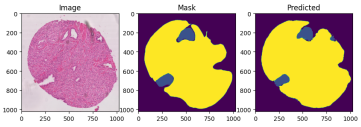
We picked some interesting cases that best shows the that best summarise the strengths and weaknesses of our model. We reconstructed the predicted images by adapting an algorithm which aims at avoiding visual discontinuities along the edges of the patches. It works by taking advantage of 50% overlapping and using a Gaussian filter to eliminate differences along the edges. In the table below are shown the difference in metrics (F1 score, IOU Score) between the patches and the reconstructed image.

We can see a strong improvement in metrics between patches and reconstructed images. This can be explained by the fact that overlapping is used in reconstructing images and, therefore, more than one image is used concurrently to construct an image portion. This leads our reconstruction to be more accurate then the one of the single patches simply recombined, without any smoothing. In Figure 6 and Figure 7 can see some examples.

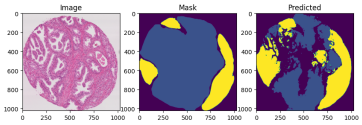
	Patches		**Reconstructed**	
Metrics	**IOU Score**	**F1 Score**	**IOU Score**	**F1 Score**
Best	0.7936	0.9844	0.9693	0.9844
Worst	0.5181	0.5378	0.3678	0.5378
Average	0.6766	0.8765	0.7981	0.8765



(a) `Patient s001_c012`

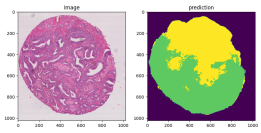


(b) `Patient s001_c039`

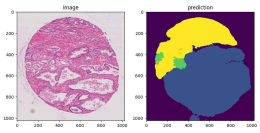


(c) `Patient s003_c055`

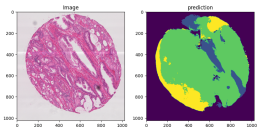
Figure 6: Some examples from the validation dataset



(a) `Patient s001_c132`



(b) `Patient s001_c143`



(c) `Patient s002_c075`

Figure 7: Some examples from the test dataset

References

[1] Aashis Khanal and Rolando Estrada. *Dynamic Deep Networks for Retinal Vessel Segmentation*. 2019. arXiv: `1903.07803 [cs.CV]`.

[2] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: `1905.11946 [cs.LG]`.

[3] Wells WM Warfield SK Zou KH. "Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation". In: 2004 July. URL: `https://pubmed.ncbi.nlm.nih.gov/15250643/`.