MASTER MVA

DEEP LEARNING FOR MEDICAL IMAGING

Kaggle Challenge

GABRIEL ATHÈNES
DINA EL ZEIN

April 10, 2022

Contents

1 Introduction

The goal of this challenge is to classify Prostate images into six states of cancer gravity. The available data consists of Histopathology images as well as masks labeling cancer tissue. Gleason scores (3, 4, or 5) are assigned to tissue which enable to compute a more synthetic ISUP grade (0 to 5) and which we will try to predict. The conversion from the gleason score to ISUP grade is explained in Figure 1.

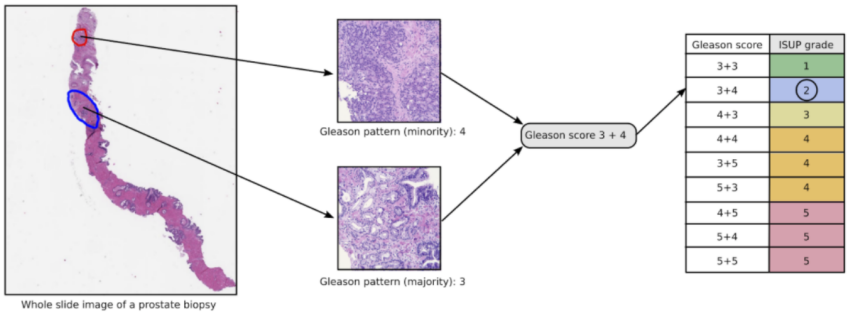


Figure 1: Conversion of Gleason score to ISUP grades

2 Data pre-processing

For preprocessing, we removed data for patients that didn't have masks for their images (2 were dropped). As the images are very large, we only used the first level of the images to get the 36 tiles of size $256 \times 256 \times 3$. For this, we used the method *get_tiles* presented in this notebook : https://www.kaggle.com/code/haqishen/train-efficientnet-b0-w-36-tiles-256-lb0-87/comments and explained in Figure 2.

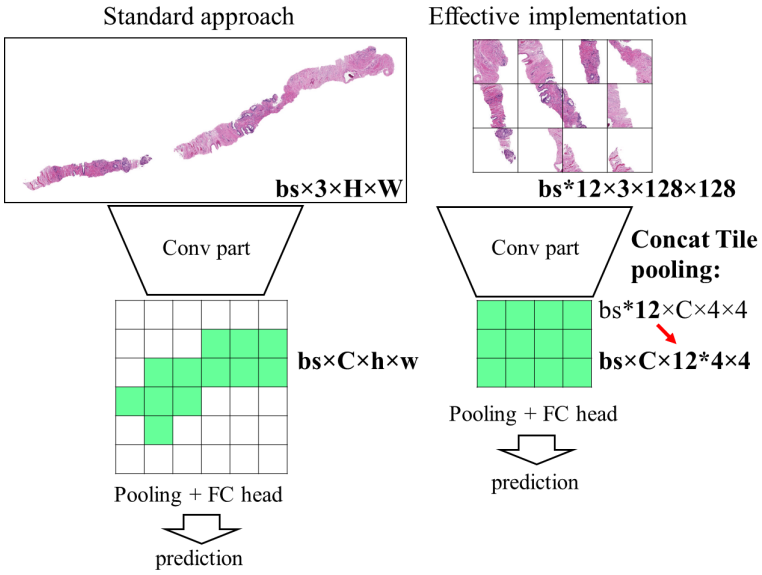


Figure 2: Tile pooling explanation taken from the notebook above

3 Architectures

In this section, we will go through the architectures that we implemented in order to tackle this challenge. We first worked on a classification neural network taking as input the tiles and predicting an isup-grade. In order to capture the information contained in the masks, we then implemented a segmentation network. We used the Unet network studied in class and trained it on image tiles to predict the mask tiles. Finally, we combined these two networks by first predicting masks for the test images with Unet, in order to classify them with EfficientNet-b0 trained with tiles of images and their masks.

3.1 Classification

We will now go through several experiments we conducted in order to find the most suited classification model for the task. We will first describe and report the results of these different experiments before displaying the classification model we chose for the challenge. Each model takes as input the 36 $(256 \times 256 \times 3)$ tiles and outputs an isup grade. The tiles are concatenated along axis 0 in order to feed our network an image of shape $9216 \times 256 \times 3$.

3.1.1 Experiments

We decided to test five classical convolutional neural networks : *Resnet18, Alexnet, Vgg16, ResNext*, and *Efficientnet-b0*. In terms of hyperparameters, the model is optimized using Adam with a learning rate of 0.0003 on 20 epochs with batch size = 2.

Pretrained models: We first tested the models that were pretrained on the Imagenet dataset that we imported from the torchvision library. We replace the last layer of each model by a layer adapted to our data and train it. Below we report the performance in terms of execution time and kappa-score of these models .

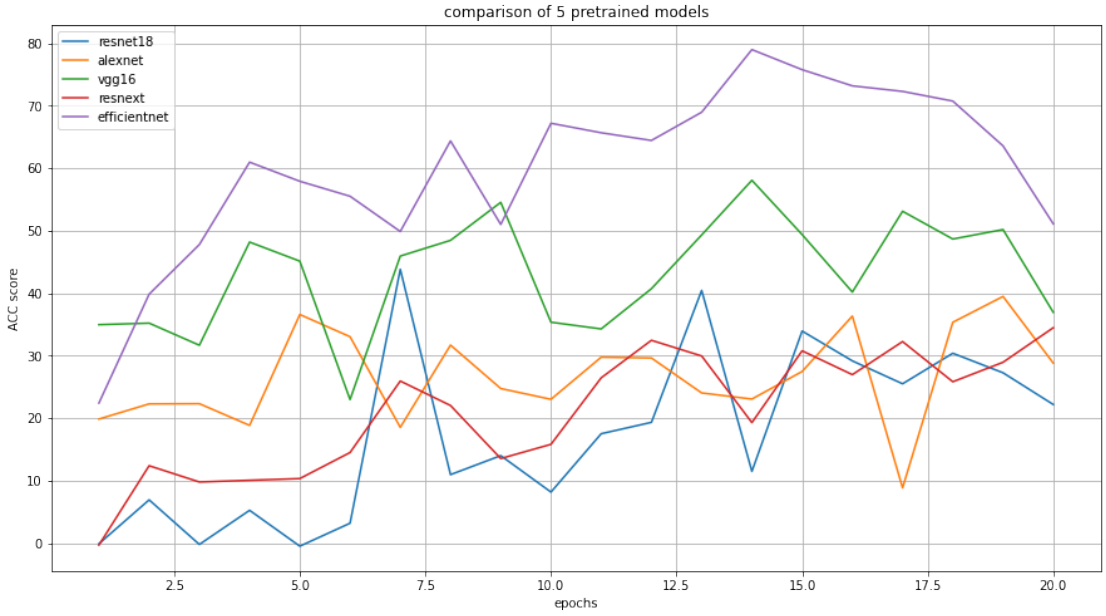


Figure 3: Comparison of the ACC scores on the valid set of the 5 pretrained models

Trained from scratch models : We then tested these models without taking pretrained weights.

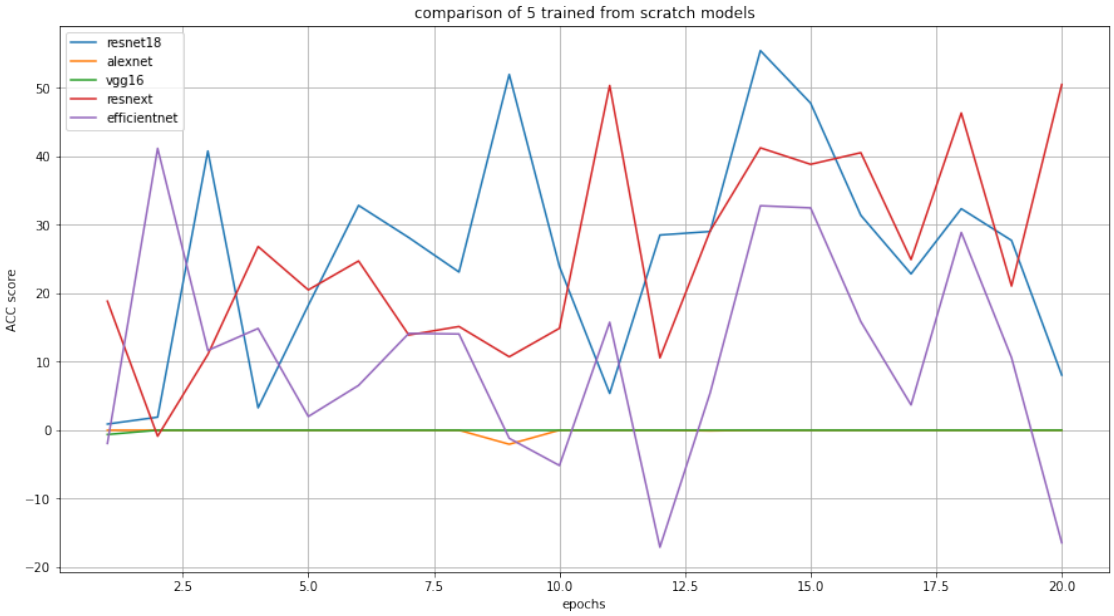


Figure 4: Comparison of the ACC scores on the valid set of the 5 trained from scratch models

Given the different results we obtained we opted for a pretrained Efficientnet-b0. We then implemented two different variants of this model.

Variant 1: We tried one-hot encoding the labels. Vector [0,0,0,0,0] represented label 0, [0,0,0,0,1] label 1 and so on. This had as benefit to reduce the number of parameters of the model but we did not notice any significant change of performance.
Variant 2: Instead of giving our model batches of shape $batchsize \times 9216 \times 256 \times 3$, we gave our model image batches of shape $36 \cdot batchsize \times 256 \times 256 \times 3$. There are therefore more batches but the images are smaller. The label batches are therefore also of shape $36 \cdot batchsize \times 6$ and we classify an image by assigning to it the majoritary label of the 36 tiles. This variant decreased our score of a few percents and we therefore did not pursue with any of these two approaches.

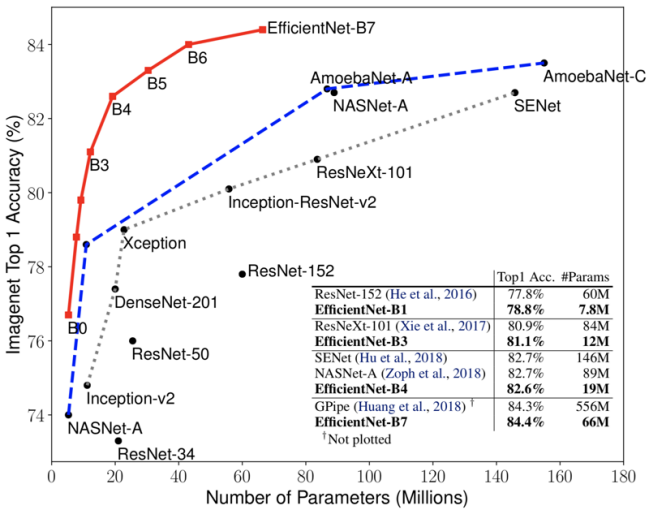
Summary

This table presents the execution time and ACC scores on the valid set for the different architectures we tried.

Model	Pretrained	Max valid Acc	Execution time (s)
Resnet18	Yes	44%	474
Alexnet	Yes	37 %	313
Vgg16	Yes	55 %	1181
ResNext	Yes	35 %	1384
Efficientnet-b0	Yes	79 %	1536
Resnet18	No	55%	789
Alexnet	No	0%	386
Vgg16	No	0%	2694
ResNext	No	50%	3079
Efficientnet-b0	No	41 %	1188

3.1.2 Efficientnet-b0

The model we chose as a result of the previous experiments is the **pretrained Efficientnet-b0**. Let's dive a bit deeper into how this model operates.



In [1], Mingxing Tan and Quoc V present Efficientnet as the result of a novel model scaling method that uses a simple yet highly effective compound coefficient to scale up CNNs in a more structured manner. Unlike conventional approaches that arbitrarily scale network dimensions, such as width, depth and resolution, their method uniformly scales each dimension with a fixed set of scaling coefficients and enables them to produce the EfficientNet models which superpass state-of-the-art accuracy with up to 10x better efficiency. As we can see on the left, EfficientNet gives state of the art performance on the ImageNet dataset while providing less parameters.

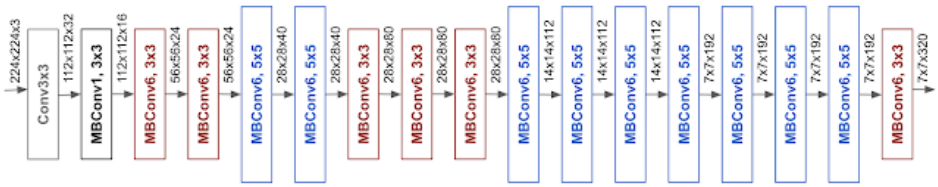


Figure 5: Architecture of EfficientNet-b0

Once we were satisfied with our classification architecture, we turned our attention on the information contained in the masks of the images.

3.2 Segmentation with Unet

We used the same Unet model than the one implemented in class as the problem was very similar. It is constitued of an encoder and a decoder. The encoder downsamples by 2 the size of the input at each block with a "Max-Pooling". The decoder upsamples progressively the extracted features using Transpose Convolution layers. To keep information of high resolution, Unet uses skip-connections to pass information from the encoder part of the network to the decoder part.

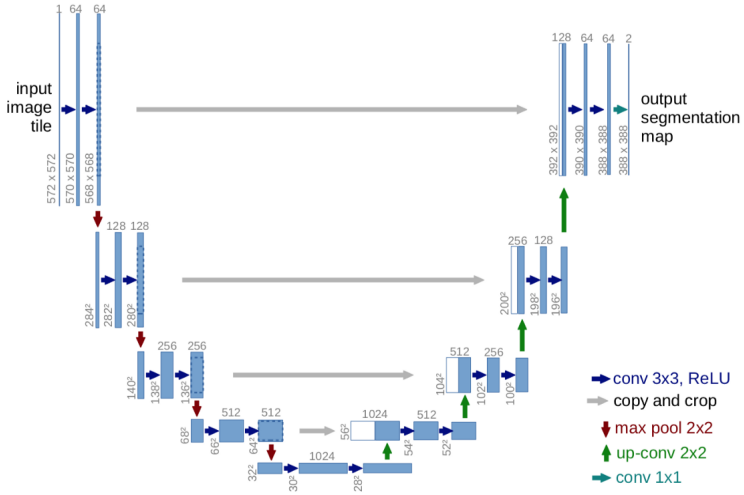


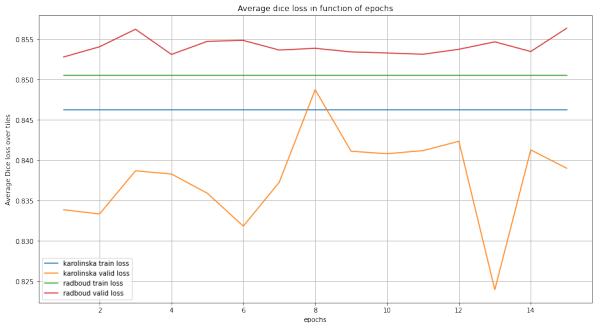
Figure 6: Unet architecture

Here is a detailed explanation of the architecture we implemented.

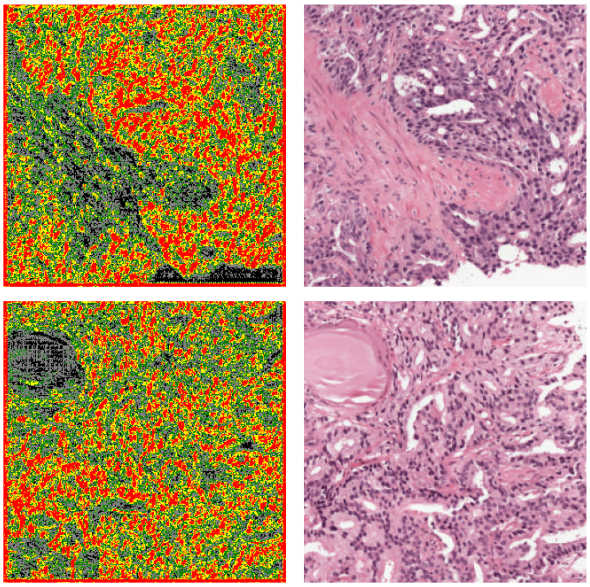
- Architecture : As both UNet and the images have an important size, we took the same architecture than the one implemented in TP, removed the 4th downsampling block and first upsampling block. As this

architecture was already using the full GPU Ram that we had available, we did not attempt to increase the number of blocks or parameters and kept this memory-optimal architecture for the rest of the challenge.

- Input : 36 tiles of shape $256 \times 256 \times 3$
- Output : 36 tiles of $L \times 256 \times 256$ with L the number of labels for the masks.
- Mask label : A mask tile has shape 256×256. As the tile outputs have shape $L \times 256 \times 256$, we one-hot encoded the 36 mask tiles of each image in order to compare them to the outputs. A mask label therefore consists of 36 $(L \times 256 \times 256)$ tiles.
- Hyperparameters : We chose a batchsize of 1, a learning rate of 0.001, Adam optimizer and the mean dice loss criterion.
- Tricks : We overrode the function $collate\_fn$ implemented by pytorch in order to give our model batches of size 36 while setting a batch size equal to 1. More precisely, we concatenate the 36 tiles in order to produce a batch of size 36, even though the batchsize is set to 1 in the dataloader. This enabled us to considerably reduce the GPU Ram required by our model as each tile is treated indenpendently of others by the weights of our model.



(a) Mean dice loss per epoch



(b) Segmentation map on two Radboud test tiles

As we can see, the mean dice loss does not seem to improve over the epochs. On the other hand, the segmented masks created seem to capture information from the tiles. Indeed after verification, the label 5 (red) pixels correspond to cancerous epithelium tissue.

3.3 Classification and segmentation with EfficientUnet

Finally, we combined these two models in order to classify the images using the information contained in the mask labels. We used the same architectures and parameters than the ones described previously for the independent models. The classifier part only differs in the input it takes. Instead of taking as input 36 tiles of shape $256 \times 256 \times 3$, it takes 36 tiles of shape $256 \times 256 \times 4$ with the first 3 channels corresponding to the images and last channel to the segmentation mask. As UNet outputs $6 \times 256 \times 256$ one-hot encoded maps, we reconstructed them to give our classifier a single 256×256 segmentation map. As each provider used different notations, we trained a segmentation network on each of the provider's dataset. We also multiplied the pixels of each mask by $\frac{255}{5}$ for the radboud dataset and by $\frac{255}{2}$ for the karolinska dataset. This enabled us to tackle the issue with the different labeling of both datasets while ensuiring masks and images had same order of magnitude. Below is a summary of the architecture of our final model.

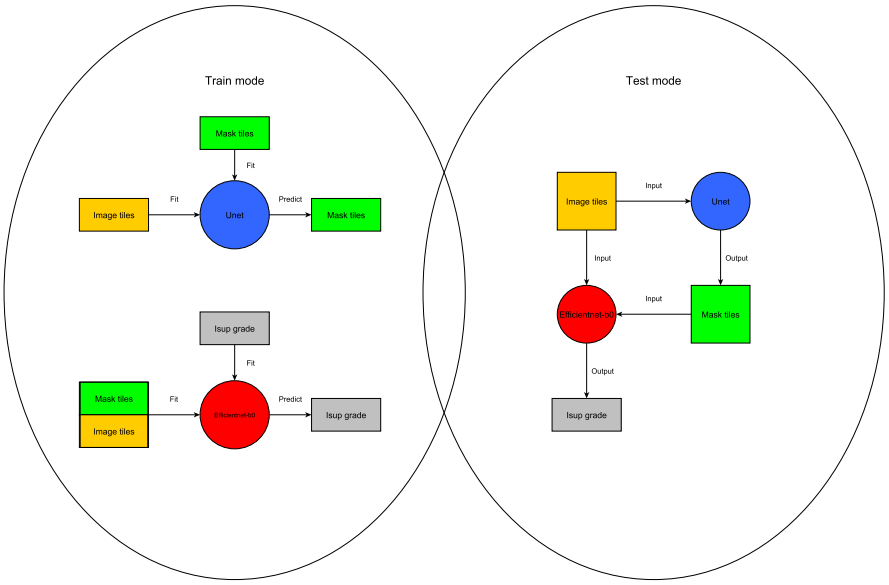


Figure 7: Architecture of our final model

4 Model tuning and comparison

Results : We faced different issues with this model for several reasons. First, all classical pretrained models only take as input 4 channels. We therefore had to replace the first layer of our pretrained model with our own. This significantly decreased the performance of our model. Furthermore, we face some major GPU Ram memory issues due to the newly added fourth channel of the tiles. We therefore had no choice but to take a batch size equal to 1, as well as only 24 tiles. The maximum ACC score that we obtained on the valid dataset was 51%.

4.1 Validation procedure

The validation procedure was the same for all of our models. we seperated the train.csv dataframe into a trainset and a validset using the function *train_test_split* from sklearn. We stratified over the isup grades in order to have a balanced valid set.

4.2 Hyper-parameter tuning

For the model tuning part, we analysed the influence of the larning rate as well as the batch size over the performance of our classifier without masks. As we explained before, we encountered GPU Ram limitations using masks and decided to conduct our studies without. For batch sizes bigger than 2 we ran out of memory.

Learning rate	0.1	0.01	0.001	0.0003
Valid ACC	0%	32%	71%	79%

Batch size	1	2	4	8
Valid ACC	75%	79%	Expl	Expl

We used several tricks that we found on other notebooks on the kaggle competition available at https://www.kaggle.com/c/prostate-cancer-grade-assessment. We used a learning rate scheduler to reduce the learning rate every 10 steps. In order to reduce GPU Ram explosion, we emptied the cuda cache at each training iteration and made sure to compute scores on the valid test using *torch.no_grad*

5 Conclusion

For this challenge, we implemented different classifiers using various methods. We concluded from our analysis that Efficientnet-b0 trained without masks was the best model to predict Isup grades. We successfully segmented the image tiles in order to get masks for the test data. We faced GPU Ram constraints that impacted the performance but still managed to get a decent 0.88 Acc score on the public dataset of the challenge competition.

References

[1] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 5