ISLES 2022

Rebecca Bonato & Marianne Scoglio

November 30, 2022

0.1 Introduction to the challenge

The goal of the Ischemic Stroke Lesion Segmentation Challenge (ISLES'22) is to evaluate the best networks capable of segment stroke lesion MR images. These models are important in order to evaluate patients' disease outcome and to define optimal treatment and rehabilitation strategies. The data-set of the challenge includes acute to sub-acute infarct lesions, with high variability in size, quantity and location.

0.2 Data handling

The data-set is organized following the Brain Imaging Data Structure and contains ADC, DWI, FLAIR and ground-truth images in 3D. It was provided to us after having followed the steps available in the website of the challenge.
We chose to focus on DWI 2D images by dividing the corresponding 3D images in axial slices. Since we noticed that the first and the last slices of each 3D image were completely black, we did not use them as components of our 2D data-set. In total we had 13184 images and their related masks. ADC images were added later as input with the DWI, but the FLAIR ones do not correspond to the mask because of their different size and number of channels.
The majority of the 2D images had a size of 112x112: we decided to resize the remaining ones into this shape in order to modify the images as less as possible. We used the nearest-neighbor interpolation, which does not change the values of the pixel in intensity, aiming to resizing the masks. The Lanczos interpolation over 8×8 pixel neighborhood has a greater quality and thus we used it to resize the DWI images (and ADC later as well). Min-Max scaling normalization was used on DWI images to restrict each pixel's value between zero and one.
Once we had our data-set, we tried two different strategies for splitting data into training and validation. The classical one consists in a random split in 70% training and 30% validation: most of the tuning of parameters has been done on that division. At the end, we tried also to perform the K-fold cross validation with 3 folders. The autocontext (mentioned below) has also been done with the K-fold method.
Since we already had a high number of different images we could have neglected data augmentation. However, this method could increase the performance, thus we implemented a training data generator.

0.3 Network set up and optimization

0.3.1 Basic U-Net

To handle the segmentation problem, we chose a U-Net model. It is the mostly used for medical image segmentation tasks since its encoder-decoder structure allows to reach better performances compared to convolutional neural networks. In the model that we used, both the encoding and the decoding part are composed by 4 blocks which involve convolutional layers, maxpooling layers, optional dropout layers (20%) and batch normalization layers. Because of the binary classification task, at the end of the neural network we added a convolutional layer with one neuron and the Sigmoid activation function. Binary cross entropy, dice coefficient and weighted dice coefficient were used as loss functions.

0.3.2 Mask boundaries

Boundaries are the most difficult task in a segmentation problem. Adding a greater penalty to the boundaries pixels through the loss function, is a way to force the network to focus on that region. The kernel size with which the mask were built is important to determine the final thickness of the boundary mask.

0.3.3 Autocontext

Context aware networks allow to integrate prior information into the segmentation pipeline. In particular, we thought that the autocontext method could help us with our segmentation task. Consequently, one input channel containing the output of a precedent network was added. The model was trained in two steps, each with three folds.

0.4 Training and Tuning of parameters

We trained these three models in order to compare the performances obtained. Starting with the basic U-Net, we tuned the following parameters in order to achieve the best performance: learning rate, batch size, data augmentation parameters, number of neurons per layer and the weight assigned to the white pixels. The number of background's pixel is much greater than the number of foreground's pixel (1:250); since the network is processing more black pixels than white pixels, it is more likely to correctly classify the first ones. This unbalanced problem could be solved giving a higher weight to the white pixels of the mask.

0.5 Evaluation scheme

We chose to evaluate the performance of the network with three metrics: dice coefficient, precision and recall. The dice coefficient allows to focus on the overlapping between the predicted and the original mask, without considering the pixels belonging to the background (True Negative). This is an advantage for solving the unbalanced segmentation task. The simple accuracy could provide a misleading information. We thought it was a good idea to analyze if the network was more likely to misclassify pixels belonging to the background or to the foreground. This is the reason why we evaluated the performance of the network with 'precision', which quantifies the error of background pixel's classification, and 'recall', which quantifies the error of the foreground one. The values obtained helped setting different weights for the two classes. To select the best model, we selected at first the ones with the highest Dice Coefficient and then we chose the one with a balance between the recall and the precision's maximum values. In fact, since we did not build the network for a specific purpose, we decide to give the same weight to FN and FP in the decision process.

0.6 Results and discussion

A table resuming the results obtained is available on the GitHub repository under 'appendix'. We started training our neural network using the Dice Coefficient as a loss function. One of the first observations we made was the absolute need of batch normalization layers in the neural network. With their insertion, the dice coefficient increased considerably from 0.04 to 0.77, without changing any other parameters. Batch normalization helps to solve the death node problem: it allows a better backpropagation of the error helping the network to learn properly. Generally, dropout and batch normalization are not used together. However, removing the dropout layers, an overfitting in the loss function is observed. That's the reason why we used both techniques in our neural network.

As a second step we focused on the data augmentation. Better results were achieved when small changes were applied to the images. Since the data set was already big and quite diverse, there was no need to create several different images. Big changes lowered the dice coefficient to 0.61 and most of all caused overfitting and stopped the learning process on the validation set. This was due in particular to the decrease of the recall (from 0.88 maximum to 0.68), thus a miss-classification of the white pixels.

The Learning rate is a crucial parameter since it determines the step size at each iteration in the learning algorithm: the steps have to be enough small to make the learning curve stable, but on the other side the convergence has to be reached in a reasonable number of epochs. After tuning it between 10e-5 and 10e-3, we found out that the perfect trade-off was 10e-4. Since the learning rate value is strictly connected with the optimiser, it is important to mention that we used 'Adam'.

The tuning of the batch size produced great changes in the performance. We can notice in Figure 1 that initially it increased with higher batch sizes, but finally it saturated (dice coefficient equal to 0.834) when the batch size was settled to 32 and 64. Large batch sizes allow a faster complete training (less time per epoch), but can lead to poor generalization. Small batch sizes have a faster convergence to good solutions (less epochs needed), but sometimes they cannot reach the global optima. This is why it was important to find a right size between these extremes, which depends on the model used and on the type of data set and its complexity.

The same figure allows us to evaluate the impact of the boundary masks in the network performances. We used two different kernel sizes in the building process of the boundary masks (2X2 and 3X3). Visually checking the mask realized, the first ones looked more appropriate and the minimum loss function reached during the training was lower. The results of the two networks in Figure 1 (with and without boundary masks) for a batch size equal to 16 and to 32, show that the boundary masks do not improve the performance of the U-net network. Even if we increase the value of the penalty (1 to 5) applied through the loss function, there are not changings. We do not know exactly why they are not useful in this case: the large size of the data-set, the sizes of the lesions and the visibility and clarity of their boundaries could have an influence.

With the purpose to make the data-set as balanced as possible, we started by assigning a weight of 250 to the foreground's pixels and then we tuned this parameter between 1 and 500. The value of the dice coefficient and the precision did not followed a specific trend; on the other side the the recall's value increased as the weight increased (Figure2 with batch size 8 but we obtained the same trend for batch size 16). The explanation for that could be found taking a look at the recall formula: by giving more importance to the foreground pixels during the training, we expect less errors in their classification (less False Negative). Since we want both to maximize dice coefficient and to have a balance between precision and recall, the best trade-off is reached with the weight value equal to 250.

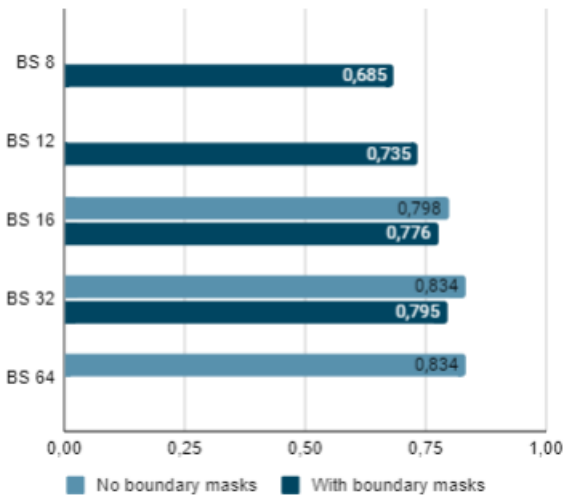

Figure 1: Batch size impact on the dice coefficient with and without boundary masks

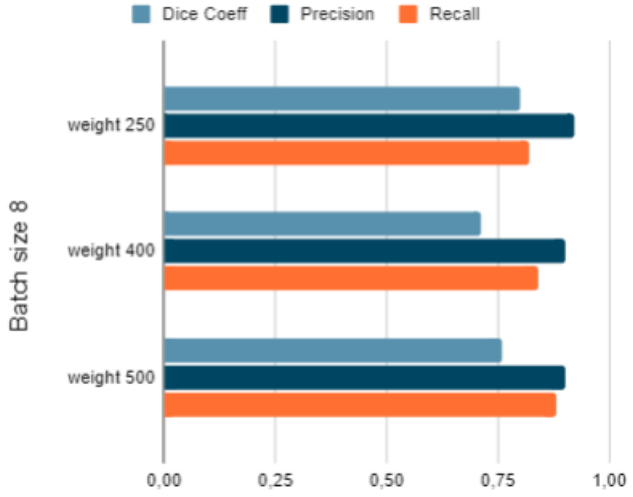

Figure 2: Foreground pixels weight impact on evaluation metrics

After that, we used the K-fold cross validation and compared it with the performance obtained with the classical split in training and validation set without changing other parameters. From all these attempts, we obtained worst results compared to the classical method. Differences in the two approaches can be related to slightly changes in the percentage of the training (66% 3-fold and 70% normal approach) and/or to a different way of splitting data into TensorFlow's function used to perform K-fold.

The training of the network including autocontext information was done in two steps and it exploited the the K-fold division of data. Since it was based on K-Fold splitting of data, as we expected we obtained worst performances compared with the classic model. Furthermore, in the second step the performance decreased. We do not have an explanation for this (except of a possible mistake in the coding part): the second step should have theoretically reached a equal or better performance, since the information obtained from the first step should have improved the network.

As the next step, we changed the loss function into binary cross-entropy. The rationale behind this is that so far we evaluated the performances with the Dice Coefficient that is also used as a loss function: our results can be biased. The performances obtained are comparable with the ones that use Dice coefficient so we assumed that the model is learning properly.

Finally, we added the ADC images, concatenating them with DWI images and we run the model which allowed us to reach the best performance. These images did not improved the network, which converged to the same metrics' values as before.

0.7 Conclusion

The best model we reached, according with our evaluation scheme, is obtained with use of the standard U-net with dice coefficient as a loss function. The hyperparameters we set are: number base = 8, learning rate = 1e-4, batch size = 32, number of epochs =150, dropout = 20%, foreground's weight = 250. Boundary mask, autocontext and K-fold have not been used. The minimum value of the loss was 0.003 while the maximum values of dice coefficient = 0.834, max precision = 0.910 and max recall = 0.860.

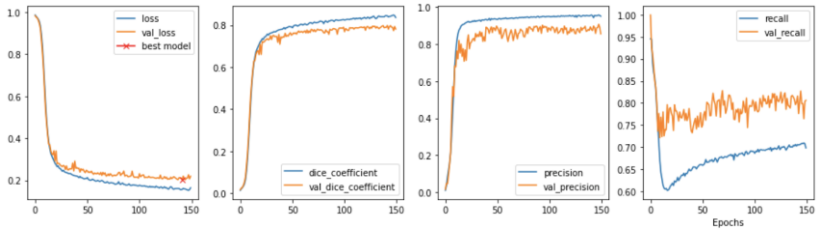

Figure 3: Best Model training process