Politecnico di Torino

Medical Image Processing Project

Optical Coherence Tomography Challenge

Group ID: FB5

Agresti Matteo -**295203**

Selene Maria Manzi -**287985**

Scorrano Luca -**296047**

Serracca Francesco -**291901**

1 Introduction

1.1 Description of the OCT technique

Optical Coherence Tomography (OCT) is an advanced and non-invasive medical imaging technique that allows to obtain high-resolution images, both spatially and temporally, of the internal structures of biological tissues. This technique allows to perform early diagnoses thanks to its ability to identify subtle structural changes in tissues, monitor the development of the tumor mass over time, observe the response to therapeutic treatment and evaluate its efficacy. The segmentation of tumor organoids in the acquired OCT volumes allows to distinguish the target regions and to collect information regarding the structure and morphology of the tumor, such as size, shape, density and volume, in order to evaluate its extension, its degree of development and its aggressiveness.

1.2 Objective of the challenge

The aim of this challenge is to implement an automatic segmentation algorithm for tumor organoids, grown in vitro, present within volumes acquired with the OCT technique. After discarding traditional segmentation techniques, such as Thresholding, Region Growing and Deformable Models, the choice fell on the Deep Convolutional Neural Network U-Net. In particular, the proposed solution allows obtaining a binary classification to discriminate pixels belonging to the background from those belonging to the target (tumor organoid).

2. Materials and methods

2.1 Dataset description

The dataset provided is composed of a total of 44 volumes, 256 slices each, provided in NRRD format and encoded in float32, divided as follows:

- 36 volumes for the training set 8
- volumes for the validation set

The volumes are provided with their respective 3D manual segmentations performed by an expert operator.

2.2 Preliminary operations

2.2.1 Reading, converting and slicing

The images of each volume, both from the training and validation set, were read using the function *imread*of the library*itk*(*Inside Toolkit*), which is useful from the point of view of image processing and biomedical data analysis.

To optimize memory usage, it was decided to work with images in uint8 (unsigned 8 bit integer) format and not in the original float32 format. In particular, this was done using the function called*conversion8* , based on the following operations: normalization and clipping of an array.

To convert to uint8, the pixel values of the original images were normalized to the range [0, 1]. In this specific case, the normalization was performed by subtracting from the initial array ($array_{slice}$) the smallest value assumed by a pixel (*minimum*) and then dividing by the difference between the maximum and minimum value of the pixel.

$$\text{array}_{\text{standard}} = \frac{\text{array}_{\text{slice}} - \text{minimum}}{\text{maximum} - \text{minimum}}$$

In all cases where the difference is equal to 0, that is, when the maximum coincides with the minimum, it is set equal to 1.

Subsequently, the normalized values were subjected to a clipping operation, through which all pixels having a value less than 0 were set to 0 and all pixels having a value greater than 1 were set to 1.

Finally, the array was multiplied by 255 and the data type (uint8) was defined.

In order to facilitate data processing, it was decided to break down the volumes along the z-axis into the corresponding 2D slices and then the function was applied to the latter.*cv2.resize*so as to ensure that their dimensions are 256x256.

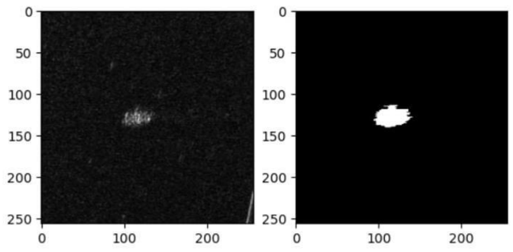

Fig. 1. Comparison between unfiltered image and manual mask.

2.2.2 Dataset reduction

It was chosen to resize the original dataset according to the following criterion: elimination of slices whose corresponding manual mask does not show any tumor organoid (composed of only black pixels). This choice allowed to obtain a substantial improvement in the performance achieved by the network, the results of which will be discussed later.

2.3 Pre-processing

2.3.1 Denoising

Several data preprocessing operations were performed in a cascade, in the following order: removal of the artifact in the lower right corner with subsequent addition of random noise in the same area and application of Wiener, median and Gaussian filters.

It was chosen as a parameter*noise*a variable named*noise*, the estimate of which was obtained by applying the function*list_mean_std_dev_calculation*, which, given a list of 2D images, calculates the standard deviation ignoring the pixels corresponding to the triangular mask. This function is given as input a list containing exclusively images without organoids, selected by comparison with the corresponding manual masks.

The removal of the artifact, when present, is done via the function*remove_lines*. In order to perform this operation, it was necessary to create a triangular shaped mask to apply to a specific area of the image using the function*create_triangle_mask*In the latter, a 256x256 matrix is initialized to zero and, using a for loop, the pixels corresponding to the triangle with vertices (215, 255), (255, 255) and (255, 155) are set to 1.

Next, the pixels of the image mask corresponding to the triangle are selected. If none of the selected pixels have an intensity of 255, then all the image pixels in the area of interest are set to 0. Then, a random noise with an intensity in the range [0, 2**noise*] via the function *add_random_noise*, in order to limit serious discontinuities in the image background.

In the example in figure 2 you can see that the organoid in the lower right has been preserved despite the implementation of the function*remove_lines*(among the pixels selected by this function there is at least one with an intensity of 255).

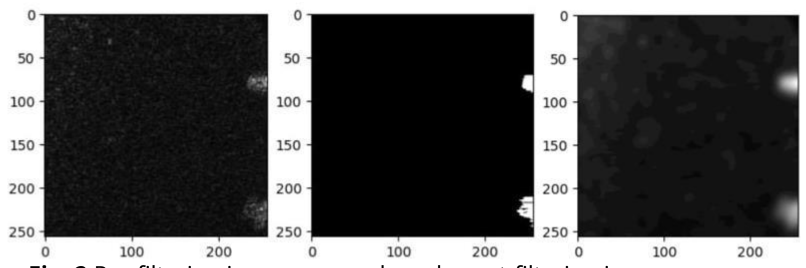

Fig. 2. Pre-filtering image, manual mask, post-filtering image

Only at this point are 3 denoising filters applied in cascade. The Wiener filter aims to improve the signal-to-noise ratio of the starting image, maintaining details and at the same time reducing noise, especially Gaussian noise. In this specific case, it was decided to use the module filter*signal* of the library*scipy*, having a kernel of size 3, and the noise level present within the image was established using the function*list_mean_std_dev_calculation*. As can be observed in the**Table 1**, the best result obtained is the one relating to the parameter*mysize*equal to 3.

WIENER FILTER	
mysize	says
1	0.468569104
2	0.47300953
3	0.473951541
4	0.464062006
5	0.463760705

Table 1. He says he obtained, with 100 epochs of training, for different values of the parameter*mysize*of the filter.

The median filter of the module was then applied to the image*ndimage*of the library*scipy*using a rectangular window of size 5x5, necessary for the calculation of the median value of the pixels. As can be observed in the **Table 2**, the best result obtained is the one relating to the parameter*size*equal to 5. This improves the sharpness of the edges of the tumor organoids and reduces salt-and-pepper noise.

MEDIAN FILTER	
size	says
1	0.454162254
2	0.457249503
3	0.438457772
4	0.418763235
5	0.463760705

Table 2. He says he obtained, with 100 epochs of training, for different values of the filter size parameter.

The last cascade filter used is the Gaussian one, also of the form*ndimage*of the library*scipy*, whose task is to further reduce the Gaussian noise. Its parameter*sigma*has been set to 5. As can be seen in the**Table 3**, the best result obtained is the one relating to the parameter *sigma*equal to 5.

GAUSSIAN FILTER	
sigma	says
1	0.469955762
2	0.466402269
3	0.473951541
4	0.45407216
5	0.478273147

Table 3. He says he obtained, with 100 epochs of training, for different values of the parameter*sigma*.

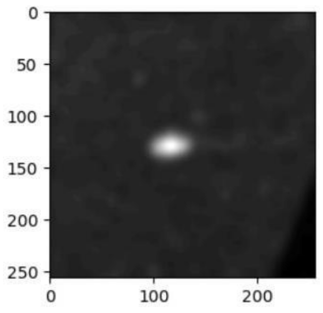

Fig. 3. Filtered slice

2.3.2 Data Augmentation and Saving Slices

In order to avoid overfitting the network, two functions were manually defined to increase the number of images to be given as input:*flip_image*And*rotate_image*. The slices of the reduced training set, equal to 5447, have been increased by about 50%. The first function flips an image along the horizontal or vertical axis, set by the input parameter*axis*; while the second one outputs an image rotated by 90° or a multiple of it using the parameter*angle_degrees*. Specifically, the images were rotated by 90° and 180°. After performing the data augmentation operations, the slices were then saved.

2.3.3 Network input data preparation

First, two dictionaries were created, called respectively*training_data*And*validation_data*, containing within them the paths relating to the images and the respective segmentation masks. Subsequently, the class was used*monai.transforms.Compose*, provided by the MONAI (Medical Open Network for AI) library, in order to obtain a suitable dataset to be fed as input to the network.

The set of transformations used to process the data was defined as follows:

- *LoadImaged*: load data using reader*NrrdReader()*.
- *AddChanneld*: adds a channel to the image and the mask. Since the images being analyzed are grayscale and two-dimensional, after applying this transformation they become three-dimensional with an explicit empty channel (batch_size, height, width).
- *Resized*: ensures that the image and mask are 256x256 pixels in size without spatial distortion.
- *ScaleIntensityRanged*: normalizes the intensity of pixel values in the range [0, 1].
- *ToTensord*: converts the image and mask into tensors.

2.4 Network architecture

After testing different convolutional neural networks suitable for segmentation of tomographic images, in particular UNet and ResNet with and without backbone, the choice fell on UNet implemented through the MONAI library.

The UNet, with its characteristic U-shaped architecture, is mainly composed of three sections: encoder (also called downsampling), bottleneck and decoder (also called upsampling). Downsampling captures low-level information by progressively reducing the size of the input images and increasing the number of filter channels (*channels*), which allows for the achievement of a higher level of detail in the images; on the other hand, upsampling captures high-level information by gradually returning the images to their original dimensions through a reduction in the number of filter channels. In particular, the progression with which the spatial dimensions increase or decrease is defined by the parameter *strides*.

The two main sections, encoder and decoder, communicate through the so-called bottleneck, in order to capture more complex information, a task performed by the parameter*num_res_unit*.

The network receives 2D images as input (*spatial_dims = 2*) in grayscale (*in_channels = 1*) and outputs a 2D binary segmentation mask (*out_channels = 1*), which provides information about the probability of each pixel belonging to the target or the background.

The input data was organized into smaller groups, called batches and subsequently normalized (*norm.BATCH*), in order to improve the computational efficiency of the network and optimize their parallel processing.

The choice of loss function fell on BCELoss (Binary Cross-Entropy Loss), as it is suitable for binary classification problems.

From a mathematical point of view, the binary cross entropy function minimizes the error between the class predicted by the model and the real class provided by the expert operator.,through the following formula:

$$(\, , \,) = - \frac{1}{\;} \sum_{=1} (\; \ast \log(\;) + (1 - \;) \ast \log(1 - \;))$$

Where:

- N is the number of samples in the batch.
- is the model's prediction for the i-th sample
- is the binary value (0 or 1) of the ground truth label for the i-th sample

The two addends of the formula previously reported are explained in detail below:

- $\ast \log(\;)$it is related to the samples to which the expert operator has associated a value equal to 1. Therefore when is equal to 1, this addend measures how much the forecasts of the model tend to 1. It follows that, the closer this value is to 1, the higher the probability that the model correctly classifies the pixels belonging to the positive class.
- $(1 - \;) \ast \log(1 - \;)$It is related to the samples to which the expert operator has associated a value equal to 0. Therefore when is equal to 0, this addend measures how much the forecasts of the model tend to 0. It follows that, the closer this value is to 0, the higher the probability that the model correctly classifies the pixels belonging to the negative class.

The two summands are averaged over the number of samples in the batch.

2.4.1 Hyperparameters

Below are all the chosen hyperparameters, set before training the network empirically with the “trial and error” technique. In order to achieve certain performances, their choice takes into account the fact that, although they are independent, they can influence each other.

Batch size

The following considerations were made in choosing the batch value:

- having a data set of sufficiently large size allows you to use larger batch sizes;
- to increase the network training speed, you can set a higher parameter value;
- In order to reduce the risk of overfitting, it is necessary to set a smaller batch;
- larger batches facilitate the convergence of the model training process.

The considerations made together with the results obtained from the tests reported in the**Table 4**led to the choice of a value equal to 32.

BATCH SIZE	
batch size	**says**
8	0.464490237
16	0.465262861
32	0.481770755
64	0.45687585

Table 4. He says he obtained, with 50 eras of training, for different batch values.

Learning rate

The learning rate, also called learning rate, indicates the extent to which the weights of a deep neural network are modified and updated during training. In other words, it determines how quickly the model adapts to the training data provided. High learning rates make the model converge faster (fast weight updating) but if too high they could exceed the optimization point, which could cause the model's performance to oscillate. On the other hand, small learning rates lead to long training times (slow weight updating) and if too low, the probability of reaching convergence is reduced to the point that the algorithm ends up in a local minimum point. Taking into account what has been said and analyzing the test results reported in**Table 5**, it was decided to set a fixed value of 10-3.

LEARNING RATE	
learning rate	**says**
0.01	0.369358848
0.005	0.367100069
0.001	0.45617061
0.0005	0.458367619
0.0001	0.464690341

Table 5. He says he obtained, with 50 eras of training, for different values of learning rate.

Weight loss

Weight decay is a regularization technique commonly used during neural network training, useful to ensure a sufficient generalization capacity of the network. This technique adds a penalty term to the loss function, which leads to a reduction in the value of the weights during the Back-Propagation phase and a consequent decrease in the complexity of the model. Resuming the concept of mutual influence between parameters, it is known that, for the use of a high learning rate, it is advisable to also increase the value of weight decay, as it allows to mitigate the fluctuations of the model performance. Specifically, after analyzing the results of the tests reported in**Table 6**, the weight decay has been set to a constant value of 10-4.

WEIGHT DECAY	
weight loss	**says**
0.01	0.422258866
0.001	0.459714841
0.0001	0.481770755
0.00001	0.467693373

Table 6. He says he obtained, with 50 eras of training, for different values of weight decay.

Number of epochs

The choice of the number of epochs is dictated by the need to avoid running into two problems:

- if an excessive number of epochs is set, it may happen that, after the model has reached a performance plateau, in order for it to continue improving, it specializes too much in the training data, consequently resulting in overfitting;

- If too few epochs are selected, the model does not have enough time to optimize the weight distribution, and thus learn how to segment the organoids.

Figure 4 shows the performance curve obtained by the model in terms of the evaluation metric on which the network training is based. Several tests were carried out, varying the number of epochs, in order to understand after how long the model reaches a plateau. In particular, from the graph it is possible to notice that from the 104th epoch onwards the Dice no longer increases significantly. Therefore, in order to avoid problems such as overfitting or wasting resources, a value of 200 was set as the number of epochs for the entire training.

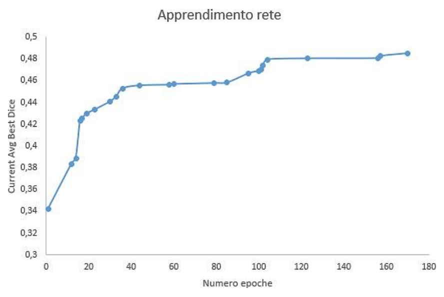

Fig. 4. Performance achieved in terms of Dice
(complete training of 200 epochs)

Each epoch is composed of a number of iterations equal to eval$_{number}$, calculated as follows:

$$= \frac{\#}{h}$$

This value has been rounded to the nearest integer.

Considering the behavior of the model performance in relation to the maximum number of epochs, i.e. the number of times the entire dataset is given as input to the network during the training process, a maximum number of epochs equal to 300 was chosen.

The total number of iterations is therefore equal to:

$$= h *$$

Optimizer

The chosen inference method is Adam (ADAptive Moment estimation), implemented by the PyTorch library, which is a widely used optimization algorithm in training deep neural networks. The main goal of ADAM is to update the model weights efficiently to reduce the loss function during neural network training. The learning rate used is the default one, which is 10^{-3}.

2.5 Post-processing

2.5.1 Post-processing transformations

Royal class

- *EnsureType*: ensures that input data are PyTorch tensors.
- *AsDiscrete*: converts input tensors to binary values (0 or 1) based on a threshold, specifically set to 0.6. If the input is below the threshold, the result of the conversion is 0; while if the input is above the threshold, the result is 1.

Predicted class

The transformations*EnsureType*And*AsDiscrete*have been applied, as for the real class, also for the aforementioned one. In addition, two further transformations have been introduced, listed below:

- *Activations*: applies a sigmoid to the model's output tensors*S*, whose mathematical expression is the following:

$$( ) = \frac{1}{1 + \ -}$$

  This activation function, by compressing the input data (x) into the interval [0, 1], allows the output to be interpreted in a probabilistic manner.
- *RemoveSmallObjects*: removes small objects, which must consist of only adjacent pixels (*connectivity=1*) present in the model's output images. In detail, all those composed of less than 25 pixels are removed (*min_size=25*). The transformation in question is applied to each channel of an image (*independent_channels=True*).

2.5.2 Volumetric reconstruction

Since the goal is to obtain a 3D segmentation mask, it is necessary to reconstruct the starting volumes: therefore the 256 slices composing each single volume have been reassembled through a for loop, both for the predicted class and for the real one.

An example of the final 3D result is shown in figure 5, where it is possible to observe the oversegmentation of the organoid present in the volume under analysis. For ease of visualization, the pixels belonging to the target were printed in red on the screen, and the background pixels were printed in transparency.

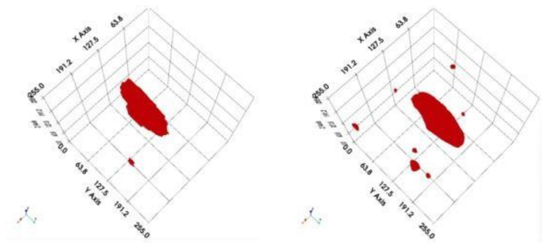

Fig. 5. Example of 3D manual mask vs 3D segmented mask

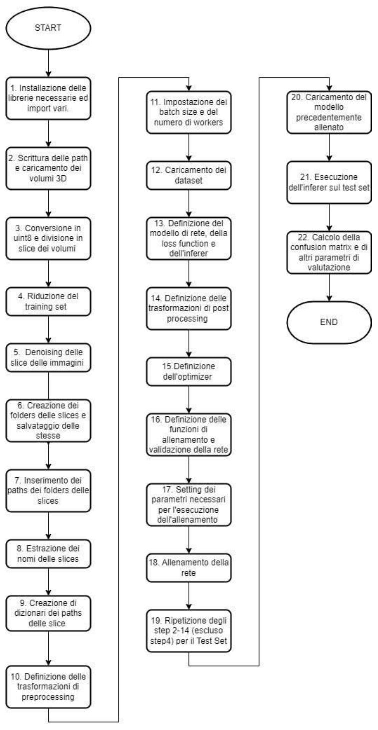
START
1. Installazione delle librerie necessarie ed import vari.
2. Scrittura delle path e caricamento dei volumi 3D
3. Conversione in uint8 e divisione in slice dei volumi
4. Riduzione del training set
5. Denoising delle slice delle immagini
6. Creazione dei folders delle slices e salvataggio delle stesse
7. Inserimento dei paths dei folders delle slices
8. Estrazione dei nomi delle slices
9. Creazione di dizionari dei paths delle slice
10. Definizione delle trasformazioni di preprocessing
11. Impostazione dei batch size e del numero di workers
12. Caricamento dei dataset
13. Definizione del modello di rete, della loss function e dell'inferer
14. Definizione delle trasformazioni di post processing
15.Definizione dell'optimizer
16. Definizione delle funzioni di allenamento e validazione della rete
17. Setting dei parametri necessari per l'esecuzione dell'allenamento
18. Allenamento della rete
19. Ripetizione degli step 2-14 (escluso step4) per il Test Set
20. Caricamento del modello precedentemente allenato
21. Esecuzione dell'inferer sul test set
22. Calcolo della confusion matrix e di altri parametri di valutazione
END

Fig. 6. Flowchart of the strategy used

3. Performance achieved

Taking into account that the classes of interest, target and background, are significantly unbalanced, the metrics chosen to evaluate in a sufficiently complete manner the performances obtained on the training and validation set are those reported in**Table 7**. Due to limited resources in terms of available RAM, the values obtained for the first 18 volumes and for the last 18 were calculated separately. All reported results are expressed as mean ± standard deviation.

- The***DSC***(*Dice Similarity Coefficient*) measures the degree of similarity between the segmentation result of the model and that of the expert operator. The higher the value of this metric, the more the two results coincide. From a mathematical point of view, this coefficient is defined as the ratio between twice the number of voxels belonging to both the manual and segmented mask (i.e. the number of true positives) and the sum of the number of voxels present in the manual mask (i.e. the sum of true positives and false negatives) and those identified in the segmented mask (sum of true positives and false positives).

- There***sensitivity***measures the number of true positives correctly identified by the mask segmented by the model compared to the total number of true positives present in the manual mask. If this metric is equal to 1, the true positives identified by the model coincide with those detected by the expert operator. It is particularly important in medical applications where correctly identifying positive cases is critical.

- There***precision***represents the ability of the model to correctly label as positive the elements that actually are positive compared to the total number of positive elements, whether true or false, identified by the model. The more the value of this metric tends to 1, the more accurate the model's prediction is.

- THE'***IU***(*Intersection over Union*) calculates the overlap between the area segmented by the model and that of the manual mask. The value of the metric is equal to 1, when there is a total overlap between the two areas. The choice of this parameter in the context under consideration allows to take into account two fundamental aspects: the ability of the model to correctly detect the regions of interest (sensitivity) and that of avoiding the overextension of the regions (precision).
- There***RVD***(*Relative Volume Difference*) is a metric calculated by comparing two volumes, the manually segmented one and the predicted one. A positive value of it corresponds to an over-segmentation, while a negative value to an under-segmentation.

Metrics	**Volumes 1-18 training**	**Volumes 19-36 training**	**Volumes validation**
DSC	0.692150303 ± 0.074659064	0.806823784 ± 0.152536457	0.749285095 ± 0.083147947
Sensitivity	0.686431132 ± 0.069936931	0.807252437 ± 0.103536287	0.751312 ± 0.086726916
Precision	0.692150304 ± 0.074659064	0.836619524 ± 0.175387422	0.756448501 ± 0.116033104
IU	0.534248832 ± 0.088022626	0.697209254 ± 0.166743541	0.605924493 ± 0.103030422
RVD	- 0.005150253 + 0.137343005	0.156907668 ± 0.837289217	0.009284565 ± 0.155702583

Table 7. Selected evaluation metrics expressed as*mean value ± standard deviation*

Please note that the results reported in the**Table 7**were obtained by applying the model to the images with which it was trained (except for the reduction of the training set); it follows that, by giving new unknown data as input to the network, the values obtained from the metrics will most likely be worse.

4. Critical analysis of limitations

During the writing of the code and its execution, several problems and aspects emerged that require further investigation and possible improvements:

- Since some functions may cause a distortion of the image size, in order to avoid any errors, it was decided to implement a possible redundancy, by performing a check twice.*resize*of the slices. In particular, the function was used*resize*of the form *cv2*of the OpenCV library and the transformation*resized*of the form*transforms*of the MONAI library. In writing the function
- *remove_lines*It was deliberately decided to give greater priority to the presence or absence of an organoid than to the removal of the artifact. Because of this, even a single pixel of an organoid needs to fall within the triangular region we described, for the removal of the artifact to not occur.
  Also, note that it is possible for the artifact to appear outside the manually set triangular region, and therefore not be entirely eliminated.
- After observing multiple images before and after implementing the Gaussian filter, and then completing the entire denoising process, it was possible to notice that the images, in which the intensity of the organoid pixels is close to that of the noise or in which only noise is present, tend to undergo a worsening transformation following the Gaussian filtering operation. Despite this limitation, it was decided to implement it anyway, because the introduced error could cause an increase in false positives, while, due to the high noise level of the image being analyzed, its absence could lead to ignoring a possible organoid (**Fig. 6**).
- A further limitation of the proposed solution concerns the decision to use a constant learning rate for the entire duration of the training. This means not taking into account the way in which the weights, within the UNet, are modified epoch after epoch.
  During the various attempts carried out, a second training session was performed with a different learning rate value, equal to 10^{-4}, which however did not lead to significant improvements in the model's performance. Therefore, in the final analysis, it was chosen to keep the value of the parameter in question fixed.
  Furthermore, an alternative solution could be the implementation of a learning rate scheduler, whose task is to automatically change the value of the hyperparameter under consideration during the model training.
- With regard to the training algorithm, it would be appropriate to introduce a stop condition that allows the training to be terminated early to limit the risk of overfitting.

In figure 7 the following images can be observed: on the top left there is the original slice, on the top right the manual mask, on the bottom left the filtered image before the implementation of the Gaussian filter and on the bottom right the image obtained after all the filtering operations.

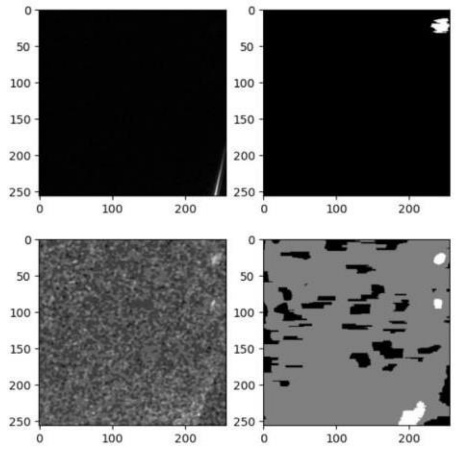

Fig. 7. Example on Gaussian filter limitation