Imperial College
London

Final Report for MSc Individual Project

Segmentation of Cortical Images Using Deep Neural Networks Pre-trained with Surrogate Images

Author name: Benjamin Billot

C.I.D: 01273185

Supervisor: Dr. Anil Bharath

Submitted in partial fulfilment of the requirements for the award of MSc in Biomedical Engineering from Imperial College London

September 2017 Word count: 5372

Feedback box for project markers:

WHAT I liked about the report:
WHAT could/should be improved:

***Abstract*— This project attempts to perform axon segmentation of cortical images by using a Convolutional Neural Network. These structures have to be trained with several thousands of images. However in this case such a dataset would be very difficult to obtain. That is why, in order to limit overfitting due to the lack of data, we propose a generative model of surrogate images that would be used as a complementary tool to data augmentation. The potential benefits of this model are assessed by evaluating the results obtained after having pre-trained an Convolutional Neural Network with generated images.**

I. Introduction

Segmentation is one of the key problems in automation of medical image processing. Its goal is to extract one type of object from an image by providing a binary map, in which white pixels indicate the object's presence (Fig. 1). Building standard semi-automated tools would have major consequences in patient diagnosis, which could become faster and more accurate [1]. Furthermore, applying this technique to cortical images could lead to breakthroughs in quantitative research. Indeed providing a tool able to detect changes occurring in cortical areas could result in a better understanding of neuronal plasticity [2] and also in new diagnosis methods of neurodegenerative pathologies such as Parkinsons and Alzheimers diseases (Fig. 2) [3].

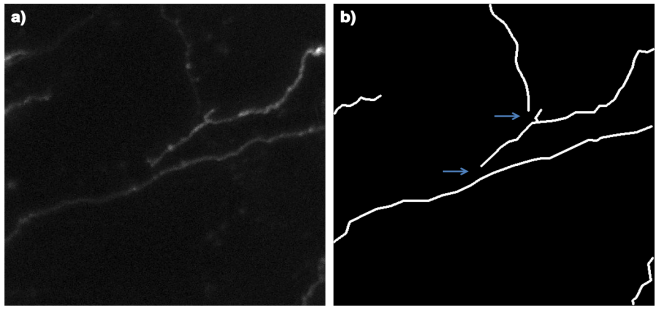

Fig. 1: Segmentation of a cortical image

a) Two Photon Microscopy of first somatosensory cortex layer 2-3 (Cher Bachar, BICV Group, 2016) b) Corresponding segmentation map. This map presents local disappearances (blue arrows) inherent to existing segmentation techniques.

Nevertheless the accuracy of existing classical segmentation techniques is debatable. These algorithms all use some kind of thresholding criteria or discriminative functions in order to differentiate structures of interest from background. Since no selection process allows a perfect separation, these algorithms are bound to result in misclassifications (Fig. 3) [4], [5]. Moreover these techniques are optimized only for one class of images, making them weakly transferable to other datasets where other threshold values should be applied. The problem is that such algorithms are based on local features; they dont have a global understanding of visual scenes. Here we propose to address this issue by using biologically inspired Deep Neural Networks (DNN) that are designed to mimic the architecture of the human visual system. As these Artificial Intelligence systems learn

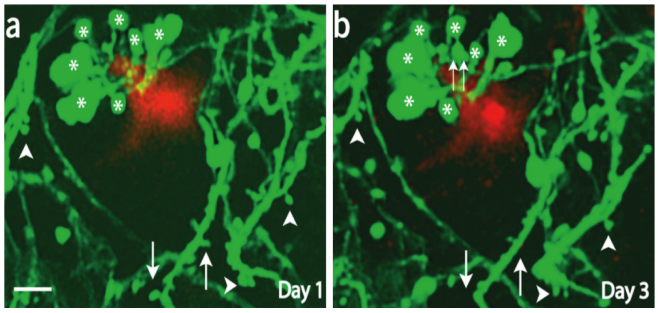

Fig. 2: Potential segmentation contributions to scientific research

a), b) In vivo time-lapse imaging of cortical axons (green) of PSAPP/YFP layer 1 [3]. an Aβ deposit (red), substance thought to be responsible of Alzeihmer's disease, provokes dendritic and axonal abnormalities (arrows and double arrow). These changes could be tracked by segmentation, which could eventually lead to a faster and more accurate diagnosis method.

the general characteristics of a dataset, they are able to perform image processing tasks such as image classification [6] and segmentation [7] with very good results.

The purpose of this project is to use a DNN to segment cortical images. Yet the efficiency of such networks is directly linked to the number of examples they are trained on. A small training dataset is likely to cause either ineffective training or poor generalisation results (overfitting) [8]. In our case obtaining a large dataset of such images would be too difficult and expansive. One solution to that problem could be to augment the original dataset by applying geometrical transformations like rotations, cropping or elastic transformations. Nevertheless, the obtained images are sometimes too redundant and irrelevant (e.g. black images). The solution

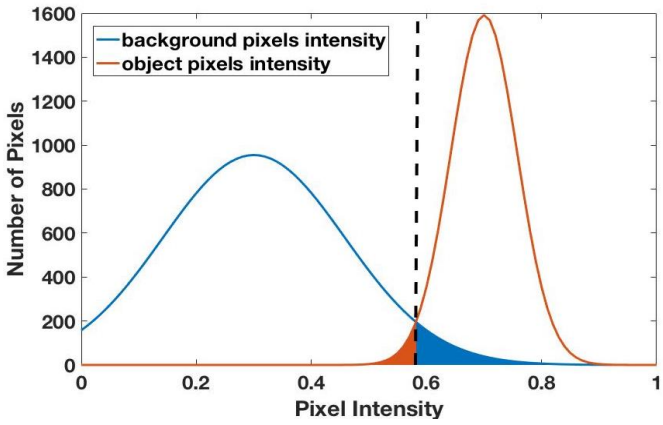

Fig. 3: Thresholding Misclassifications

This graph exemplifies the problem that emerges with thresholding processes. Assuming that an image contains one kind of object on a relatively uniform background, one can plot the intensity distribution of object pixels (orange curve) and background pixels (blue curve). If this image is segmented by using a threshold (black dashed line) the orange region depicts object pixels falsely allocated to the background whereas the blue region represents background pixels wrongly identified as being part of the object.

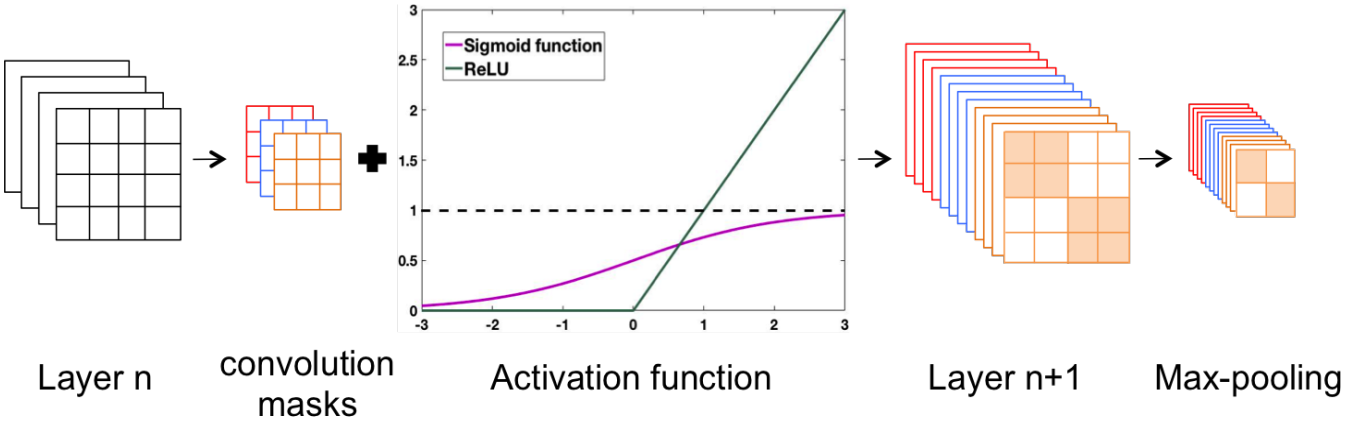

Fig. 4: Description of convolution and max-pooling operations

What is called a convolution is in fact the combination of two operations. Information is processed from layer n to layer n+1 by applying several convolutional masks (or kernels) to the feature maps of layer n. Immediately after, each resulting pixel value is mapped in a specific space by an activation function. There are numerous choices, and here we plotted two of the most commonly used functions (sigmoid function and Rectified Linear Unit). The convolution operation is often followed by a max-pooling layer, which takes the maximum value out of four contiguous pixels.

proposed in this project is to create a generative model of surrogate images mimicking real datasets. It would provide an alternative to data augmentation by producing new datasets instead of transforming already existing ones. However it is unlikely that the model's complexity would allow training only on generated data, or even on a mixed dataset of generated and real images. This model has been thought as a complementary tool that could be used to pre-train a network. By expanding the available data it would prevent overfitting and possibly accelerate training. The same way one would teach a task by starting with simple examples and finishing with more complicated exceptions, the network would first learn the general shape of an axon on simple images and then real images of increased difficulty would be used to perform fine-tuning. Therefore, it is of extreme importance that we are able to control the model's complexity. Moreover, making this generative model highly tunable would allow its extension to a lot of different datasets, notably those containing long and thin branching structure such as neurons or vascular networks.

II. Background

A. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are Artificial Intelligence structures inspired by the human visual system. Their specific architecture makes them extremely efficient to perform image processing tasks such as classification, segmentation or features detection. A network is constituted by cascaded layers. Each one of them is made of many feature maps that are tuned to detect specific characteristics in the input image. Information propagates through the network by convolving the feature maps with different masks and by applying an activation function (sigmoid, ReLU) to the result. These manipulations are often followed by a max-pooling operation that consists in selecting the maximum pixel value out of a square window (Fig. 4). As it reduces by half the size of a map, max-pooling enables to double the receptive field of the next layer. Therefore the network starts with local operations and then builds progressively its understanding of the entire image.

B. Autoencoders

An Autoencoder is a Convolutional Neural Network with an output layer of the same size as the input [9]. This type of network is often used to reconstruct an output image similar to its corresponding input, consequently making it very suitable for segmentation [10].

The architecture of this type of network contains two distinct parts. First a contractive path (encoder) tries to capture main images features and stores them in the middle layer (latent space). It can be seen as a compressed version of the input. The expansive path (decoder) retrieves the information contained in the latent space to build the output image (Fig. 5). To obtain the same image size at the output, several up-convolutions must be applied to the intermediate feature maps. Here this operation is simply used to double the size of images by inserting zeros between each pixel.

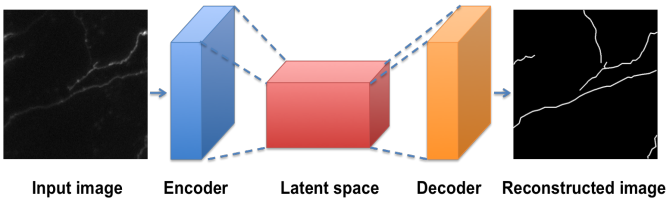

Fig. 5: Autoencoder architecture

An autoencoder is constituted of two sub-CNNs: first an encoder seizes images' features and a stores them in the latent space, then a decoder reconstructs the output image.

C. Training a network

The purpose of training is to find the set of convolution masks that yields the best prediction on test data. To train a network, one needs to dispose of a dataset containing original images paired with their expected output. A fixed number of images is randomly picked from the dataset (batch) and run through the network. The error between the obtained outputs and their corresponding expected version is measured by computing a cost function. Then the backpropagation algorithm estimates the gradient of this cost function with respect to every parameter by averaging it over the batch [11]. Every coefficient of every convolution map is then modified with a gradient descent approach:

$$\omega_{l,m,i,j}^{(k+1)} = \omega_{l,m,i,j}^{(k)} - \frac{\eta}{N} \sum_{n=1}^{N} \frac{\partial C_n}{\omega_{l,m,i,j}} \tag{1}$$

$\omega_{m,l,i,j}$ = coefficient (i,j) of m^{th} mask of convolutional layer l
$\omega_{m,l,i,j}^{(k)}$ = value of $\omega_{m,l,i,j}$ after batch k
η = learning rate
N = batch size
C_n = cost function of batch n^{th} image

When the entire dataset has been used (epoch), the training carries on by iterating the process on the training dataset several hundreds of times. Before being able to efficiently train a network, one must first find suitable values for all the hyper parameters (learning rate, batch size, kernel size, ...).

III. Methods

The final purpose of this project is to perform segmentation on a real dataset consisting of 152 Two Photon Microscopy images of layer 2,3 in the first somatosensory cortex (S1), provided along with their corresponding maps. Twenty images have been randomly selected from this dataset to form the test data. The training and validation datasets have been obtained by data augmentation of the remaining images. The images were first cropped, rotated and underwent elastic deformations. The resulting training and validation datasets were eventually constituted of respectively 26,200 and 200 images of size 128x128.

Instead of directly using the augmented real dataset, the network was first trained with different surrogate datasets of increasing complexity obtained with the generative model. We used this first phase to conjointly validate the development of the model and the tuning of the networks parameters.

A. Generative model of surrogate images

The model for surrogate images have been created on Matlab. Although it has been designed to mimic the provided real dataset, we directed our efforts to make this code highly tunable in order to be adaptable to as many datasets as possible.

The algorithm starts with an empty matrix that is progressively filled with Control Points generated by a random walk

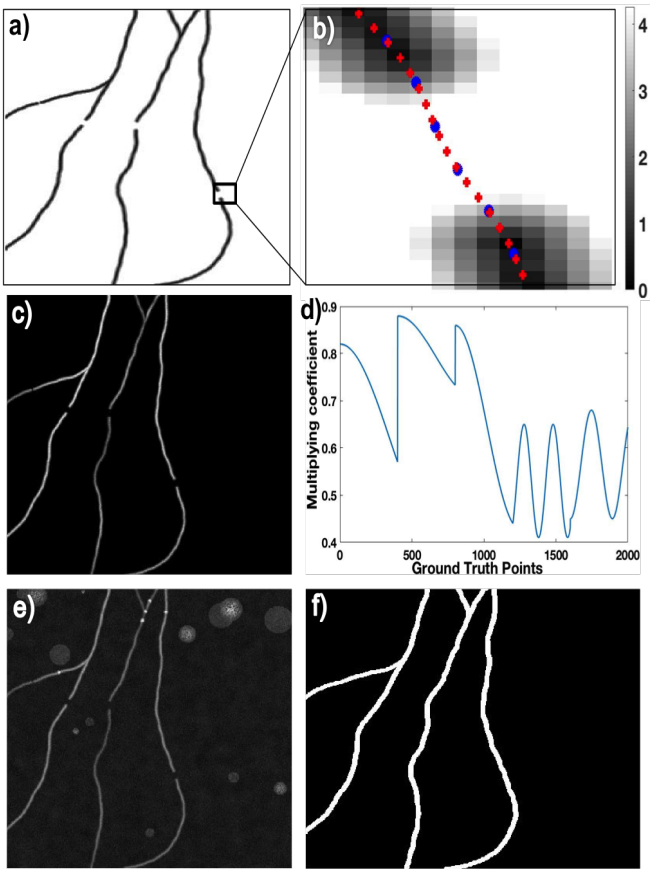

Fig. 6: Generation of an image

a) distance matrix b) zoom on a gap region. Blue circles illustrate Control Points and Red crosses denote Ground Truth Points. c) Intensity matrix d) Ground Truth Points' associated variation coefficients. e) Example of a generated image. It contains axons, synaptic boutons (bright spots on the axons), cells (large circles), and different sources of noise (white and colored noise). f) Corresponding segmentation map

through the matrix. The angle between three consecutive points is controlled so that we obtain relatively straight axons. Control points generation stops when one of the matrix's edges have been reached. Ground Truth Points are then obtained by fitting a spline through the Control Points. GT Points have a much higher density than Control Points therefore allowing to attain a sub-pixel resolution. For each pixel we calculate the distance to every GT Point and we save the distance to the closest one as well as its corresponding index. If this distance is below a given threshold (i.e. axon's thickness) the pixel is said to be part of the axon. All the other pixels are reset to zero. Gaps can also be introduced by omitting several contiguous GT Points in the distance calculation (Fig. 6,b). Several daughter branches can further be attached by randomly picking one GT Point and starting the same process over. Other axons can also be added to the image by repeating this mechanism (Fig. 6,a).

However the simultaneous presence of several axons in an image raises the question of crossings. In fact crossings frequently occur in real images but depending on the use

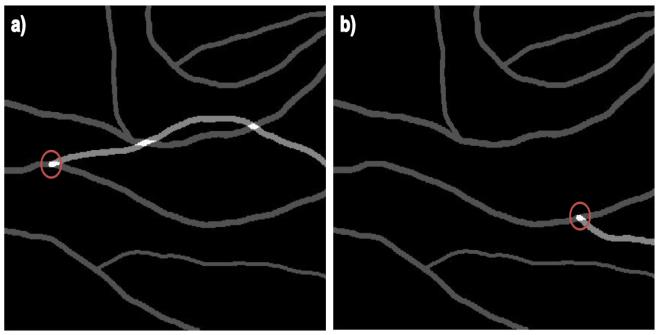

Fig. 7: Creation of a new branch

a) Intersections (white regions) between the previously created branches (dark gray curves) and the new branch (light gray curve). As this curve crosses other existing structures outside the tolerance region (red circle) a new branch will be created. b) No crossings occur outside the tolerance region, the branch will be added to the image.

of generated images they can sometimes be unsuitable. Therefore a parameter can activate or deactivate a specific mechanism dedicated to detect potential intersections. Each time a new branch is created, the algorithm uses binary masks to check if it crosses previously created axons. The problem is that mother and daughter branches necessarily cross each other at the branching point. That is why a circle centered on the branching point defines a tolerance region, in which all intersections are ignored (Fig. 7). Nevertheless one must be careful when adjusting the circle's radius. Indeed a small tolerance region would forbid mother and daughter branches to form low angles and a too large region could lead to intern crossings. This radius is directly controllable by a parameter, which was set to a middle range value in order to match real axons' geometry [12]. New branches are generated until one does not cross other axons outside the branching region. Infinite loops are avoided by restarting the entire image when 50 iterations have failed.

After having created all the axons, the distance matrix containing all the branches is converted into an intensity matrix by applying a Gaussian profile to the distances. In order to obtain intensity variations along axons, the peak values of the Gaussian profiles are multiplied by a variation coefficient defined for each GT Points (Fig. 6,d). Eventually, white Gaussian noise is added to the axons (Fig. 6,c). These transformations, underwent only by axon pixels, are summarized in Equation (2).

$$I(m,n) = \frac{V(m,n)}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{D(m,n)^2}{2\sigma^2}\right) + N(m,n) \qquad (2)$$

$I(m,n)$ = Intensity matrix
$V(m,n)$ = Variation coefficients of the corresponding GT Point
$D(m,n)$ = Distance matrix
$N(m,n)$ = Gaussian White Noise
σ = Standard deviation of the Gaussian intensity profile

In order to obtain more realistic images of axons, structures mimicking synaptic boutons are inserted along branches. Those are represented by little bright circles of varying intensity and with a radius equal to the axon's thickness. The black background is then filled with other noisy circles that reproduce cells, often present in this kind of dataset. Again the intensity of both types of circles follows a Gaussian profile.

Eventually, the image is completed by imitating different types of perturbations occurring during image acquisition. The noise model takes into account random photon reception by CDD cameras, optical blurring and perturbations introduced by the electronics [13]. Equation (3) (where $\circledast$ denotes the two-dimensional convolution and $\lfloor x \rfloor = max(0,x)$) provides an accurate definition of how this noise was mathematically expressed. As a result the image is blurred with white and colored (correlated) noises (Fig. 6,f).

$$F = \lfloor f_{real} \circledast O1 + f_{Poisson} \circledast O2 + f_{GWN} \rfloor \qquad (3)$$

F = final image
f_{real} = real image as perceived without noise
$f_{Poisson}$ = Poisson Process mimicking random photon reception at the CDD camera
$O1, O2$ = Point Spread Functions modeling Optical blurrings caused by the lens
F_{GWN} = Gaussian White Noise introduced by the electronics

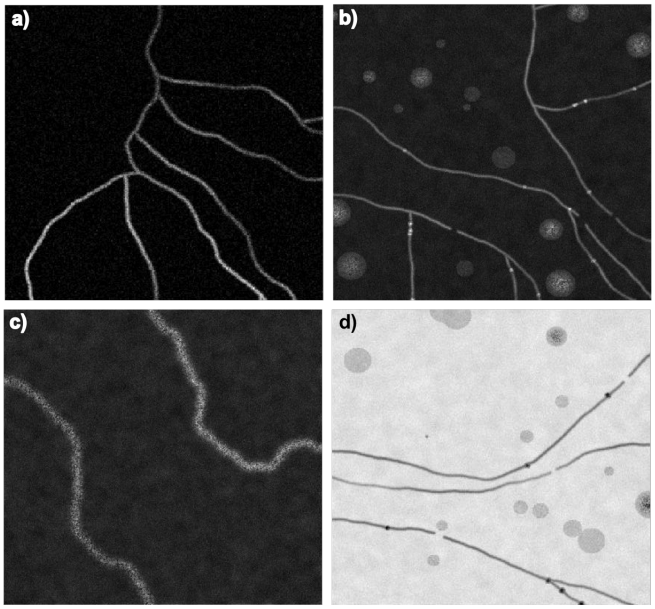

Fig. 8: Examples of images generated with different settings

a) one axon with 6 branches, no background noise. b) Thin axons with synaptic boutons and gaps, cells of different size and brightness, moderate noise level. c) Two broad, noisy and curvy axons on a very noisy background. d) Dark structures on light background.

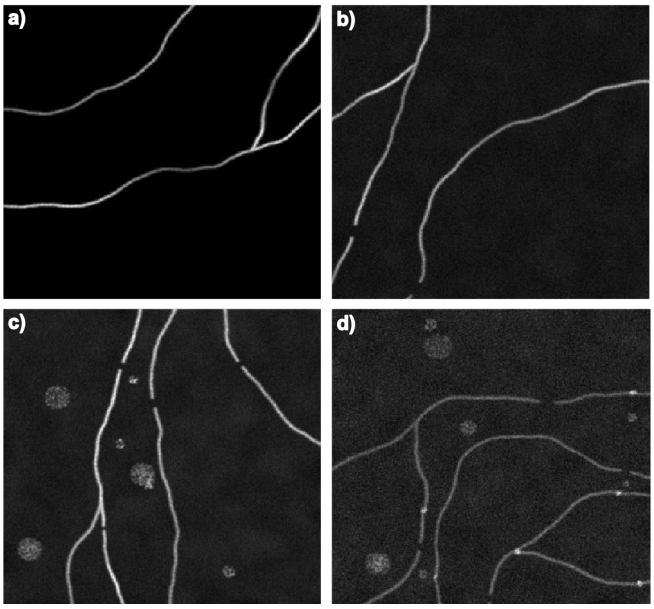

Fig. 9: Examples of 256x256 images generated with different complexity levels

a) very easy b) easy c) medium d) hard

Complexity Level	Corresponding Difficulties
very easy	axons
easy	axons + noise
medium	axons + noise + gaps + cells
hard	axons + noise + gaps + cells + boutons

TABLE I: Definition of the Complexity levels

This generative model can produce several thousands of images along with their corresponding maps in few hours (depending on the image complexity). As the purpose of this model is to be highly adaptable to numerous datasets, we made it tunable with a lot of parameters (some of them have already been introduced). For each image, every parameters are randomly picked in a predefined range in order to obtain varied datasets. The settings control the number, shape and intensity of each type of objects, as well as the background noise and intensity (Fig. 8). A complete list of the parameters is provided in Annex I.

B. Experiments

1) Development of the model and tuning of the network: The purpose of this first part was to progressively tune all the networks hyper parameters (learning rate, batch size, number of epochs, kernel size,) and to validate the development of the model. In order to see if the network was able to separately deal with the different structures present in the model (cells, boutons) twelve categories of datasets of progressive difficulty were created. Each category is defined by an image size (64x64, 256x256, 512x512) and by a complexity level (very easy, easy, medium, hard) defined in table I and illustrated in Figure 9. For every category, three datasets were created: a training dataset of 26,200 images, a dataset of 200 images to validate training after each epoch, and a test dataset of 200 images.

2) Segmentation of the real dataset: The goal of this second part was to compare different training methods in order to perform segmentation of real images. In particular we want to analyse the effects of pre-training the network with surrogate images, and see if the model could efficiently be used as a tool to increase the amount of images for a given dataset.

In order to make measurements comparable, we generated a new surrogate dataset with the same number of images as the real dataset (26,200 128x128 images). The ranges of parameters were increased with respect to the hard_512 dataset to obtain images of higher variability. Moreover the average amount of noise was also increased. The goal was to make the network more flexible and robust to feature changes when testing it with real data.

After having generated the new dataset, six different experiments were conducted (Table II). The first one was the control experiment because the network was trained and testes only with real images. Experiments 2,3 and 4 tested the generative model and its ability to replace real data. In order to efficiently compare the results, the network was always trained with the same number of examples (10,480,000). In experiment 3 and 4, the ratio between surrogate and real data was increased to act as if the number of real images was limited. Therefore the surrogate dataset can be seen as a way to multiply the amount of available data (x2 and x3 for experiments 3 and 4). All other things being equal (especially the number of pre-training examples), we preferred to maximise the number of surrogate images rather than augmenting the number of epochs. For instance, in the third experiment 26,200 surrogate images were used on 200 epochs rather than 13,100 images on 400 epochs. That allowed to get better generalisation results and to avoid overfitting. Finally the two last experiments evaluated the contribution of real images in the training of the network. A big difference in the test accuracies (between experiments 3 and 6 on one side, 4 and 7 on the other) would underline the efficiency of the pre-training phase.

exp. number	surrogate data		real data	
	images	epochs	images	epochs
1	-	-	26,200	400
2	26,200	400	-	-
3	26,200	200	13,100	400
4	26,200	300	6,550	400
5	26,200	400	1,000	400
6	-	-	13,100	400
7	-	-	6,550	400
8	-	-	500	400

TABLE II: Experiments for Segmentation of real images

For each experiment the table shows the number of images and epochs on which the network was trained. Surrogate data were always applied first when both types of datasets were used. For experiments 1 to 4 the network was trained with the same number of examples.

Two more tests were completed in order to assess the performances of the generated images in case of a very small dataset. The network was first trained with the full surrogate dataset during 400 epochs, and then with only 500 real images for another 400 epochs. This was compared to the results of the network trained only with the 500 real images (experiments 5 and 8).

C. Network Architecture

The network used in this project is a Unet, which has been proved to perform especially well in segmentation of biomedical images [14]. A Unet is a type of Autoencoder which is typified by concatenating the feature maps of the contractive path to the symmetrical ones in the expansive path (Fig. 10). This allows to provide contextual information enabling a more accurate reconstruction of the input.

The contractive path counts three convolutions combined with rectified linear units (ReLU). At each downsampling steps we apply a 2x2 max-pooling that reduces the image size by half and we double the number of feature maps. The expansive path is built in a symmetrical way. At each step, we up-convolve the feature maps and the result is concatenated with the corresponding ones of the contractive path. Convolving masks are then applied with ReLU reducing the number of feature maps by half. Eventually the output is obtained by convolving the last layer and by applying a sigmoid function.

D. Training and hyper-parameters calibration

Training has been achieved by following the standard procedure described in the Background section (cf.§II,C). We detail, here, some considerations related to training parameters.

1) Learning rate and Optimizer: Different values of the learning rate have been tested (every powers of ten from 4 to 6). Every time 10^{-5} yielded the best results after 50 and 400 epochs. The chosen optimizer is Adam because it is one of the most efficient algorithm for gradient descent [15].

2) Batch size: Concerning the batch size, there is a trade off between the accuracy of gradient estimation (a large batch allows a better gradient estimation, cf. §II,C) and the memory space of the GPU. Hence this parameter has been adapted to the size of the images on which the network was trained.

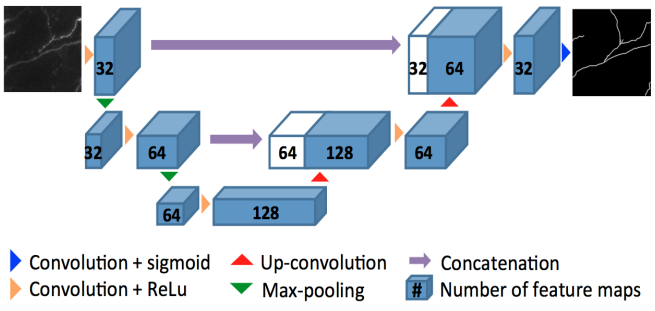

Fig. 10: Architecture of the Unet

3) Kernel size: Segmentation's purpose is to indicate the presence of axons in their entirety, even if there are local disappearances due to intensity variations. Therefore we want the maps to detect these little gaps and to fill them. This can only be achieved if the receptive field of the latent space's feature masks are able to visualize an entire gap (otherwise the latent space would perceive two axonal endings). As the receptive field is determined by the size of the convolutional masks, three values were tested: 3,5 and 7. Knowing that each max-pooling operation doubles previous layer's visual space (cf. §III,A), it led to receptive fields of sizes 12, 20 and 28. Because of the concatenations between the contractive and expansive paths, gaps can still be present in the second part of the network, that is why this change was applied to the masks of both paths. Broader masks werent tested because they would have implied an excessive training time (an epoch with 26,200 images already took 30 minutes for a kernel size of 7).

4) Cost function: The metric used to compute the error after each batch is extremely important because the entire backpropagation algorithm is based on the evaluation of its gradient (cf. §II,C). Several choices are possible, but in segmentation the most commonly used functions are the well-known binary cross-entropy [10] and the dice coefficient metrics [16]. If $\odot$ denotes the element wise product and A and B are two images, the dice coefficient is defined as:

$$Dice\ coefficient = \frac{\sum_{m,n} A(m,n) \odot B(m,n)}{\sum_{m,n} A(m,n) + B(m,n)} \tag{4}$$

Both of dice coefficient and binary cross-entropy metrics were tested on surrogate datasets.

5) Dropout probability: Dropout was first presented in 2012 [17] as an alternative solution to the overfitting problem introduced in §I. It simply sets to zero the coefficients of randomly picked masks during training. Therefore the convolution masks can't rely on the presence of other masks and they are forced to learn more robust features. This technique used conjointly with data augmentation allows to significantly increase the performance of Deep Neural Networks [17]. The dropout probability was another parameter to adjust. Again several values were tested: 0, 0.1, 0.2, 0.3, 0.4 and 0.5 (after 0.5 training is much slower and likely to be ineffective).

E. Post-processing and testing

To measure training's efficiency we compare predicted and expected maps. This process starts by running the test data through the network. Nevertheless Unet's last activation function is a sigmoid, therefore the pixel values of the outputs are real numbers between 0 and 1. Instead of being purely binary as they should be, these values reflect the probability of the presence of an axon. That is why 200 thresholds (evenly spaced between 0 and 1) were tested in order to get the best logical maps possible. For each threshold, we computed the average prediction accuracy. The threshold

that yielded the best score was then saved as well as the corresponding maps.

IV. Results and Discussion

A. Kernel size and Cost function effects on training

1) Study of kernel size effects:

Several kernel sizes (3, 5, 7) were tested in order to measure their effect on the map prediction (cf. §III,D,3). They respectively achieved 95.2%, 96.4%, and 96.5% of segmentation accuracy after 400 training epochs. The comparison between Figures 11,b and 11,c demonstrates that the first type of kernel did not provide a receptive field large enough to apprehend and fill most of the gaps. The stagnation of dice coefficient between convolution masks of size 5 and 7 is depicted in Figures 11,c and 11,d. This result suggests that there is a width limit above which gaps cant be filled because the axons on each sides are perceived as two different entities. This limitation is probably due to the fact that we trained the network to separate axons and especially those that are very close to one another in order to prevent irrelevant connections between unrelated branches. As a result, the kernels of size 5 were chosen because training was much faster and they got very similar results compared to those of size 7.

2) Study of the cost functions:

Segmentation of the hard_512 dataset was performed using the dice coefficient and binary cross-entropy cost functions. After 400 epochs we obtained 96.4% and 93.9% respectively with the dice coefficient and the binary cross-entropy metrics. Although these measurements are not comparable, we can say that the maps were globally well reconstructed. Nevertheless the two types of function can be discriminated based on visual assessment. Contrary to binary cross-entropy, the use of dice coefficient allowed to fill most of the gaps and to get relatively clear branch separations (Fig. 12).

These results are explained by considering the very sparse structure of the datasets used in this project (white thin structures on a black background). The binary cross-entropy is computed by taken all the pixels into account, whereas the dice coefficient only measures the overlay of the bright pixels between two maps (by definition if a pixel is black

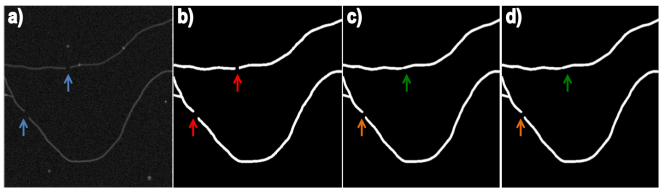

Fig. 11: Consequences of changing the kernel size

a) Example of an image from hard_512 dataset. Gaps are indicated by blue arrows. b) Corresponding predicted maps for kernel of size 3. Neither small or large gaps are filled (red arrows). c,d) Kernel size is increase to 5 and 7. Small gaps are filled (green arrows), whereas large gaps are only partially filled (orange arrows).

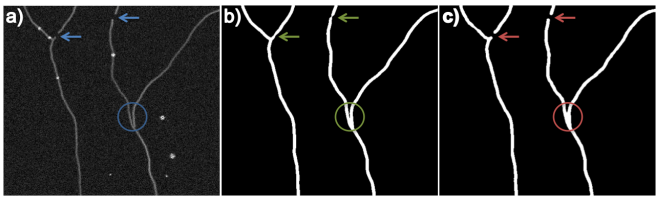

Fig. 12: Effects of changing the cost function

a) Example of an image from hard_512 test dataset. Gaps are indicated by blue arrows. The blue circle denotes a tricky branching zone b,c) Corresponding maps respectively predicted with the dice coefficient and binary cross-entropy cost functions (kernel size 5). In the first case the network is able to fill the gaps (green arrows) and to distinguish branches in the branch region (green circle). In the second case, the network can't fill the gaps (red arrows) and struggles to separate two branches in the division region (red circle).

in both images it doesnt contribute to the calculation, cf. §III,D,4). This characteristic allowed focusing the learning on the regions of high pixel intensity where the network had to learn the difference between axons and other objects (cells, noise artifacts) [18].

B. Segmentation of surrogate datasets

Table III summarizes the results obtained after training the network during 400 epochs with the previously defined surrogate datasets (cf. §III,B,1). One can observe that test accuracy generally decreases with complexity level. For instance the dice coefficients values declined from 99.9% for the veasy_64 dataset to 96.4% for the hard_64 dataset. Indeed the network had to learn more complex features as image difficulty is increased, therefore leading to slightly poorer results.

The same accuracy evolution can be discerned when the size of images was increased. For example the dice coefficient dropped from 97.5% for the veasy_256 dataset to 90.2% for the hard_256 dataset. Indeed, when calculating the surface ratio of bright regions over background, one would find that it rapidly falls with image height (around 35% for 64x64 images down to only 4% for 512x512 images). This rarefaction of regions of interest means that learning requires much more time, regardless of the epoch duration time that inherently soars with image size. Another problem arising with the use of large images is the reduction of the batch size (cf. §III,D,2). For 64x64 training images we were able to use 64 images for a batch, whereas the batch size was reduced to only 4 samples when using datasets of 512x512 images.

complexity \ image size	64	256	512
very easy	99.9%	97.5%	94.2%
easy	99.2%	93.8%	90.1%
medium	97.1%	92.6%	97.4%
hard	96.4%	90.2%	96.5%

TABLE III: Dice coefficient for surrogate datasets segmentation

The sudden increase of dice coefficient for medium_512 and hard_512 is due to a change in the training method. Because of the image size and complexity, the network was not able to learn anything at all, even after a long training time (2 days). One solution could have been to crop these huge images in order to reduce the duration of one epoch and to increase the batch size. However cropping 512x512 images, which have a very sparse structure, would have given a lot of irrelevant black images. However the generative model offered the possibility to produce small images containing all the needed difficulties. That is why we generated new datasets of 128x128 images using the same parameters as we did for the corresponding 512x512 datasets. This new method significantly accelerated the training and obtained very satisfying segmentation maps: 97.4% and 96.5% for medium_512 and hard_512.

C. Real dataset segmentation

In this section, we present the results of the experiments defined in §III,B,2.

1) Training with real and surrogate datasets:

The two first experiments consisted of training the network exclusively with real or surrogate images (Exp. 1 and 2). The reconstructed maps respectively yielded dice coefficient values of 65.1% and 58.3% (Fig. 13,a). The difference between the validation dice coefficients and the testing accuracies came from the post processing phase (Fig. 13,b). Although the structure of the maps predicted after training on surrogate data presented large gaps, they clearly identified the axons. These good results underline the quality of the generative model and are probably linked to the high variability of the surrogate images used to train the network (cf. §III,B,2). Training with real data enabled finner and more accurate predictions, despite gaps still being present in the segmentation maps.

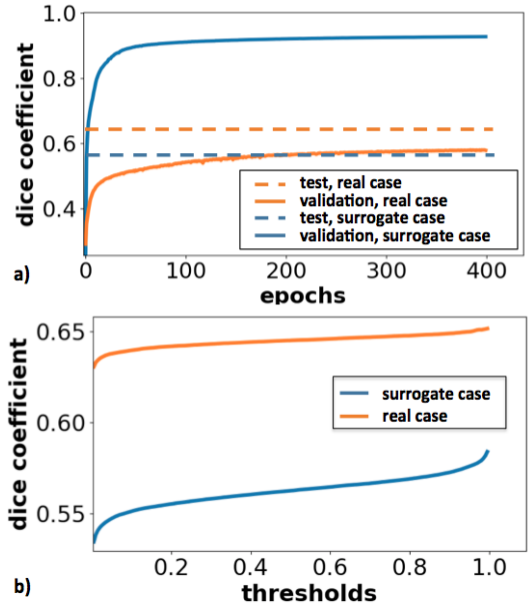

Fig. 13: Comparison of training and surrogate datasets

a) Validations dice coefficients during training. After each epoch the network is evaluated by a validation dataset similar to the training one (therefore explaining the difference between the two curves). In the real case the accuracy converges much slower because of the increased image complexity. The dashed lines illustrate the results obtained after testing the network. b) Dice coefficients obtained with different threshold levels. As the threshold raises, the segmentation gets finer because it discards the noisy pixel.

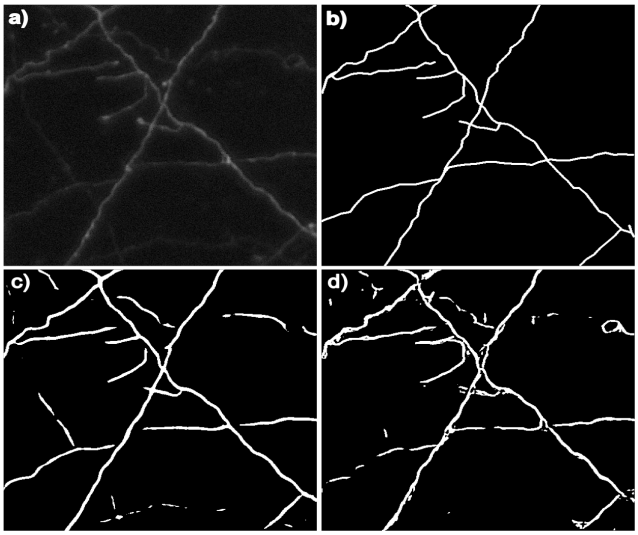

Fig. 14: Comparison of predictions based on real and surrogate datasets

a) Example of a real image. b) Corresponding map. c) Map predicted after training with real data (exp. 1). d) Maps predicted after training with generated data (exp. 2).

2) Pre-training of the network and fine-tuning:

The table IV summarizes the dice coefficients yielded by experiments 1 to 6. As expected the accuracies of experiments 3 to 6 are between those obtained when training the network exclusively with real or surrogate data (experiments 1 and 2).

The dice coefficient obtained after experiment 3 (where the network was trained equally with real and surrogate images) are very close to those of experiment 1 (1.1% of difference). When comparing experiments 3 and 6, one can find that results are similar, dice coefficients only differing by 1.2%. This suggests that the pre-training images did not have much influence because even if it was divided by half, the amount of real images used was still sufficient to attain satisfying segmentation level.

Nevertheless, when the proportion of real images used for training was decreased from half to quarter the accuracy of prediction only declined from 64.0% to 63.5 % (experiment 3 and 4) whereas the corresponding control experiments dropped from 62.8% to 59.4% (experiments 6 and 7). The large gap between test accuracies of experiments 4 and 7 (4.1%) proved that this time the pre-training has significantly contributed to the reconstruction of test maps.

exp. number	proportion of real data used	accuracy
1	100%	65.1%
2	0%	58.3%
3	50%	64.0%
4	25%	63.5%
5	3.9%	59.3%
6	100%	62.8%
7	100%	59.4%
8	100%	33.1%

TABLE IV: Dice coefficient after pre-training and fine-tuning experiments

This general trend is confirmed by the experiments 5 and 8 where only a small part of the real dataset was used to train the network. The two-phased training allowed to reach an equivalent prediction accuracy (59.4%), whereas the real images subset only led to a dice coefficient value of 33.1%. Therefore the generative model can be efficiently used as an complementary method to data augmentation, especially in the case of a small dataset.

V. Conclusion

In this project we developed a generative model of surrogate images. This model has been designed so it can be adaptable to numerous datasets containing long and thin branching structures such as axons and vascular networks. Here, we used this model to mimic a dataset of cortical images (first somatosensory cortex, layer 2-3).

The model has been tested by segmenting these cortical images with a Convolutional Neural Network yielding a Unet-like architecture. The comparison between the results obtained when the network was trained exclusively with real or surrogate images (respectively 65.1% and 58.3% dice coefficients) highlighted the quality of the model.

In order to investigate if the generative model could be used as a complementary method to data augmentation, the network was pre-trained with surrogate data and then fine-tuned with real images. It has been demonstrated that the model particularly ameliorates the results obtained for small datasets.

Future work would involve to develop the model to three dimensions in order to segment entire regions of the brain because 3D datasets are even more difficult to obtain and likely to be very small. In shorter-term we could also integrate different types of synaptic boutons to the model, as for now only those appearing directly on the axons are taken into account by the model. Generating images with a more precise representation of these structures would allow to segment bouton images, on which it is much more difficult to perform data augmentation (elastic deformations are not really possible in this case).

The model could also be modified to be able to generate time-series images that would vary very little in time. A network could then be trained to detect changes across time, allowing to study more precisely synaptic changes and abnormalities caused by neurodegenerative diseases.

Another way of improving the results could be to investigate how the model's parameters could be optimized to match as precisely as possible the features of the real dataset. This study would be very interesting because the generative model is highly non-linear (thresholds, max and min-like operations,etc) and no reliable solution exist for solving this kind of problems.

Material

Generative model and Unet implementations available at https://github.com/BBillot/generative_model_and_unet.git

References

[1] Neeraj Sharma and Lalit M. Aggarwal. Automated medical image segmentation techniques. *Journal of Medical Physics / Association of Medical Physicists of India*, 35(1):3–14, 2010.

[2] S. Basu, A. Aksel, B. Condron, and S. T. Acton. Tree2tree: Neuron segmentation for generation of neuronal morphology. In *2010 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 548–551, April 2010.

[3] Jaime Grutzendler and Wen-Biao Gan. Two-photon imaging of synaptic plasticity and pathology in the living mouse brain. *NeuroRx: The Journal of the American Society for Experimental NeuroTherapeutics*, 3(4):489–496, October 2006.

[4] Dzung L. Pham, Chenyang Xu, and Jerry L. Prince. Current Methods in Medical Image Segmentation1, November 2003. DOI: 10.1146/annurev.bioeng.2.1.315.

[5] Siti Noraini Sulaiman, Noreliani Awang Non, Iza Sazanita Isa, and Norhazimi Hamzah. Segmentation of Brain MRI Image Based on Clustering Algorithm. *ENERGY, ENVIRONMENT, BIOLOGY and BIOMEDICINE*, page 54, 2014.

[6] Alex Krizhev, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *NIPS*, 2012.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *arXiv:1412.7062 [cs]*, December 2014. arXiv: 1412.7062.

[8] Michael A. Nielsen. Why are deep neural networks hard to train ? In *Neural Networks and Deep Learning*. 2015.

[9] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[10] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *arXiv:1511.00561 [cs]*, November 2015. arXiv: 1511.00561.

[11] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning representations by back-propagating errors. *Nature*, 323:533, October 1986.

[12] Quan Wen and Dmitri B. Chklovskii. A CostBenefit Analysis of Neuronal Morphology. *Journal of Neurophysiology*, 99(5):2320–2328, May 2008.

[13] Pierre Gravel, Gilles Beaudoin, and Jacques A. De Guise. A method for modeling noise in medical images. *IEEE transactions on medical imaging*, 23(10):1221–1232, October 2004.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[16] Kelly H. Zou, Simon K. Warfield, Aditya Bharatha, Clare M.C. Tempany, Michael R. Kaus, Steven J. Haker, William M. Wells, Ferenc A. Jolesz, and Ron Kikinis. Statistical Validation of Image Segmentation Quality Based on a Spatial Overlap Index. *Academic radiology*, 11(2):178–189, February 2004.

[17] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*, July 2012. arXiv: 1207.0580.

[18] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, 15, August 2015.

APPENDIX

Parameter	Utility	Type
General characteristics		
height	height of the image	int
width	width of the image	int
negative_image	allows to change background's color from black to white if set to 1	logical
MinAxons	minimum number of axons	int
MaxAxons	maximum number of axons	int
MinBran	minimum number of branch per axon	int
MaxBran	maximum number of branch per axon	int
Background noise		
sigma_noise_min	minimum white Gaussian noise level	int
sigma_noise_max	maximum white Gaussian noise level	int
lambdaMin	minimum white Poisson noise level	int
lambdaMax	maximum white Poisson noise level	int
Shape of axons		
conformity	set the straightness of axons	float $\in [0, 1]$
MinThickness	minimum axon thickness	float
MaxThickness	maximum axon thickness	float
MinGapSize	minimum gap size	int
MaxGapSize	maximum gap size	int
Construction of axons		
StepSize	distance between each Control Points	float
NSplinePoints	number of GT Points per branch	int
crossingOK	$0 =$ no crossings between axons, $1 =$ crossings allowed	logical
StraigthBranching	maximum angle between two branches of the same axon	float
sigma_spread	standard deviation of the axonal intensity Gaussian profile	float
Intensity of axons		
MinAxonIntensity	minimum intensity for an axon	int $\in [0, 100]$
MaxAxonIntensity	maximum intensity for an axon	int $\in [0, 100]$
AxonProfile	intensity function along branches	linear or cosine
MinPeriod	minimum period of the cosine intensity function	float $\in [0, 100]$
MaxPeriod	minimum period of the cosine intensity function	float $\in [0, 100]$
sigma_noise_axon	level of white Gaussian noise added to axons	float
Shape and intensity of boutons		
MinNbBouton	minimum number of synaptic boutons	int
MaxNbBouton	maximum number of synaptic boutons	int
MinBouRadius	minimum bouton radius	int
MaxBouRadius	maximum bouton radius	int
MinBrightnessBouton	minimum bouton intensity	int $\in [0, 100]$
MaxBrightnessBouton	maximum bouton intensity	int $\in [0, 100]$
sigma_noise_bouton	level of white Gaussian noise added to boutons	float
Shape and intensity of cells		
MinNbCircles	minimum number of circles	int
MaxNbCircles	maximum number of circles	int
MinRadius	minimum circle radius	int
MaxRadius	maximum circle radius	int
MinBrightnessCircles	minimum circle brightness	int $\in [0, 100]$
MaxBrightnessCircles	maximum circle brightness	int $\in [0, 100]$
sigma_noise_circle	level of white Gaussian noise added to circles	float

TABLE I: Dice coefficient for surrogate datasets segmentation