E4 project : Maximin Affinity Learning of Image Segmentation

Quentin Garrido, Tiphanie Lamy Verdin, Josselin Lefèvre, Annie Lim

January 2020

Contents

1 Introduction

2 Theoretical background

3 Maximin Affinity learning

3.1 Method's presentation

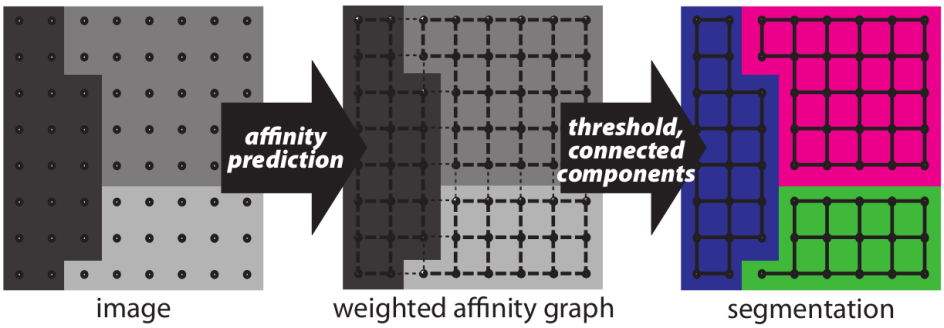


Figure 1: Description of the method, from [1]

The method, as illustrated in figure 1 works in two steps.
The first step is to compute an affinity graph G, with nodes V representing each pixel in the original image and edges E weighted by the affinity between neighbouring pixels. The graph G will be 4-connected and for every pair of pixels $i, j \in V \times V$ there exist and edge $(i, j) \in E$ iff i and j are neighbours. We will note the affinity between neigbouring pixels A_{ij}.
An affinity of 1 between i and j means that they belong to the same object, and an affinity of 0 means that they belong to different objetcs.

Once we have our affinity graph, with affinities between 0 and 1, we will threshold it and remove edges under a certain affinity to obtain connected components that will be our objects, and this thresholded affinity graph will be our final segmentation.

3.2 Optimizing the Rand Index

There exist various methods of evaluating an image segmentation, one of them being the Rand Index which is defiend as follow :

$$1 - RI(\hat{S}, S) = \binom{N}{2}^{-1} \sum_{i<j} |\delta(s_i, s_j) - \delta(\hat{s}_i, \hat{s}_j)|$$

With S our groundtruth segmentatiin $\hat{S}$ our predicted segmentation and $\delta(s_i, s_j)$ the indicator function taking value 1 if $s_i = s_j$ (if pixels i and j are in the same segment/object) and 0 otherwise.
Intuitively, this can be seen as the fraction of image pixels where both segmentations agree.
This gives us a way to evaluate an image segmentation that penalizes when two objects are merged or split in our final segmentation as we can see in figure 2. On the left, only one edge is misclassified but this merges two objects and will be heavily penalized by the Rand Index (we would see the same results if an object was split). On the contrary if we have misclassified edges inside of objects that do not create any splits, this will not be penalized at all by the loss since our objects are still correctly delimited.

As such the Rand Index seems a good measure of segmentation quality for our problem (edge-based segmentation). We will have to see how exactly can we optimise this metric in our case.

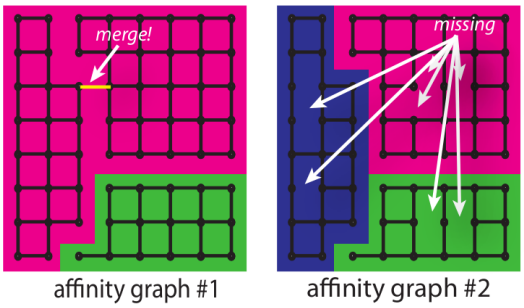


Figure 2: Exemple image segmentations and their affinity graph. On the left we have a merging of two objects due to a misclassified edge and on the right we have edges that should be of value 1 that have been removed. From [1]

3.3 Maximin affinity and the maximin edge

In order to optmize the Rand Index, we must find a way to compute $\delta(\hat{s}_i, \hat{s}_j)$. To to this we will use the concept of maximin affinity, as described in [1].
Let $\mathcal{P}_{ij}$ the set of all paths between i and j in our image. For every path $P \in \mathcal{P}_{ij}$ there is an edge(s) (k, l) with minimal affinity.

This allows us to define the maximin path, which is th path which maximizes the mininal affinity. It is defined as follow :

$$P_{ij}^* = \arg \max_{P \in \mathcal{P}_{ij}} \min_{(k,l) \in P} A_{kl}$$

The edge of minimal affinity in the maximin path will be called the maximin edge and will be writter $mm(i, j)$.
It allows to define the maximin affinity which is simply the affinity of the maximin edge.

$$A_{ij}^* = \max_{P \in \mathcal{P}_{ij}} \min_{(k,l) \in P} A_{kl}$$

The most important consequence of this is that a pair of pixels i, j in connected in the thresholded affinity graph iff A_{ij}^* is greater than the threshold value, as shown in [1].

This means that if we can find the maximin affinity efficiently, we will be able to compute the Rand Index efficiently.
In practice the maximin affinities can bee computed efficiently using a maximum spanning tree (MST), since any path in the MST is a maximin path.

3.4 Computing an affinity graph

As we can see, once we have our affinty graph, obtaining the segmentation is straightforward, but the issue is : How to obtain an affinity graph?
As stated before, an affinity graph is equivalent to the contours in the image, and as such any method to find the contours in an image can be used to find the affinity graph (with relative success).
The first idea that we could have would be to use a contour detector, such as the Canny filter for example, however this would lead to an obvious over segmentation, which is not desirable here.

An other idea (the idea used in [1]) would be to use a neural network to obtain our affinity graph. Neural networks, and particularly convolutionnal neural networks (CNNs) have proven to be an extremely powerful tool in image processing and especially in image classification and segmentation, which makes

them a good candidate for MALIS.

The architecture used originally is a fully convolutionnal neural network (FCNN) which is a CNN with only convolutionnal layers. As such they can work with various input sizes which is always a nice feature.

Now that we have seen how we can compute an affinity graph, let's see how we can train this classifier, and if it even possible to train it with the Rand Index as a loss function.

3.5 Training a classifier

As stated before, the goal of this method is to optimise the Rand Index, defined as:

$$1 - RI(\hat{S}, S) = \binom{N}{2}^{-1} \sum_{i<j} |\delta(s_i, s_j) - \delta(\hat{s}_i, \hat{s}_j)|$$

with S our groudtruth and $\hat{S}$ our predicted segmentation.
We can define $A^*_{ij}(I, \theta)$ the maximin affinity of pixels i and j in the predicted affinty graph of our classifier on I with parameters θ. We can rewrite the previous equation as :

$$1 - RI(I, \theta, S) = \binom{N}{2}^{-1} \sum_{i<j} |\delta(s_i, s_j) - A^*_{ij}(I, \theta)|$$

However this function is not differentiable everywhere so we cannot use it directly for our training. We will replace the absolute value with a smooth loss function l such as the mean squared error. This relaxation will allow us to use gradient descent to train our classifier(refer to [1] for more details on this).

Thus our final loss function will be :

$$L(I, \theta, S) = \binom{N}{2}^{-1} \sum_{i<j} l(\delta(s_i, s_j), A^*_{ij}(I, \theta)) = \binom{N}{2}^{-1} \sum_{i<j} l(\delta(s_i, s_j), A_{mm(i,j)}(I, \theta))$$

We can now look at what the training loop will look like.

Algorithm 1 MALIS training loop, from [1]

1: **repeat**
2: Predict the affinity graph for an image I
3: Pick two random pixels i, j from I
4: Find the maximin edge $mm(i, j)$
5: Update our parameters θ with gradient

$$\nabla l(\delta(s_i, s_j), A_{mm(i,j)}(I, \theta)$$

6: **until** convergence is reached

In practice we won't train on the whole image since it would be too long to compute the maximin edge in the whole image. We will instead train on 21x21 patches of the image. We have to choose these patches carefully as most of them will mostly be in an object and not around a border.
Since borders between objects are far less common than the "inside" of an object, we may most of the time train only for pixels inside a same object.
This would result in class imbalance between our "borders" (affinity of value 0) and or "inside" (affinity of value 1). Training with such imbalance would lead to worsened results since borders would not be present most of the time in the chosen image.(one example of such pathological behaviour would be the prediction of affinty 1 everywhere, far from what we desire)

As such when training we need to be careful about what training image we select.

4 Implementation

4.1 Computing the affinity graph

4.2 Training the neural network

After reading the paper we realized that there was some missing informations. One of the most important points, the input and the output of the neural network, was not clear. After some research we were lucky to find the Turaga's Phd about MALIS. Thus, we found important information we were missing.
The network predicts the affinity following each axis. As we can see on the Figure 3, in case there the network is feed by a 3D image, it output three affinity images following the axis X, Y and Z. Then we have to merge these three images in a way to obtain the complete affinity image.

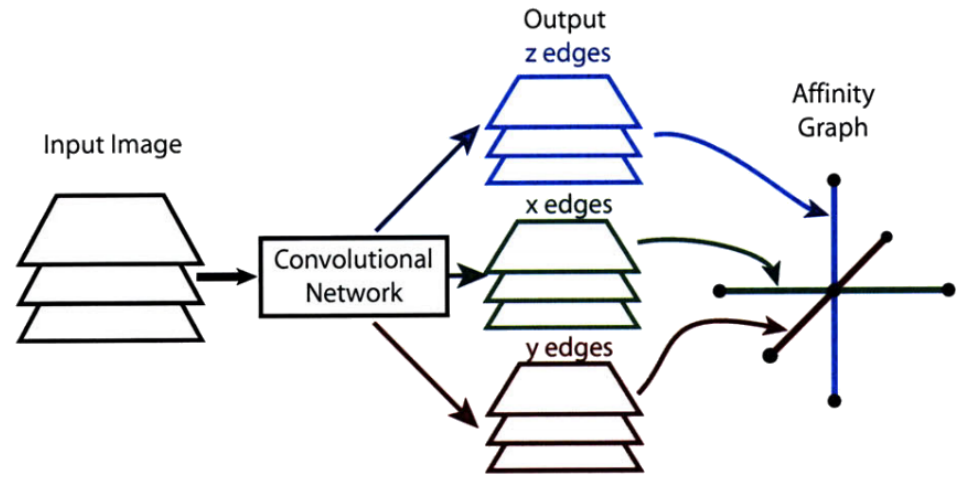


Figure 3: Creating the affinity graph using a convolutional network. The input to the network is the 3d EM image and the desired output is a set of 3d images: one each for the x, y and z directions representing the affinity graph, from [2]

Another problem was to understand the input shape. In the paper, they use a patch size of 21*21*21. But it is also said that it led to an affinity classifier that use a patch with a shape of 17*17*17 to classify an affinity edge. In a first time it was kind of blur but we figured out that 17 correspond to the volume taking in account after four convolution layers, reminiscent to the proposed architecture. The patch size is also arbitrary as we are using a FCN that, by definition, don't care about the input shape.

As said earlier, we have to find the maximin edge in order to compute the loss. As we have to do this operation for each iteration, it is very important to guarantee a very low computational time. Our first approach was to use the Breadth First Search algorithm on the Maximum Spanning Tree efficiently created with Higra. Finally, this method was not good because our implementation was suffering from the slowness of Python. In our last version we are computing a Binary Partition Tree, a binary tree by altitude ordering, on the MST. This data structure is pretty pertinent as the maximin edge between to pixel i and j is the lowest common ancestor of these two pixels in the BPT. Higra also allow us to compute the loss with a larger number of pairs without an explosion of computing time because picking the lowest common ancestor is achieved in constant time.

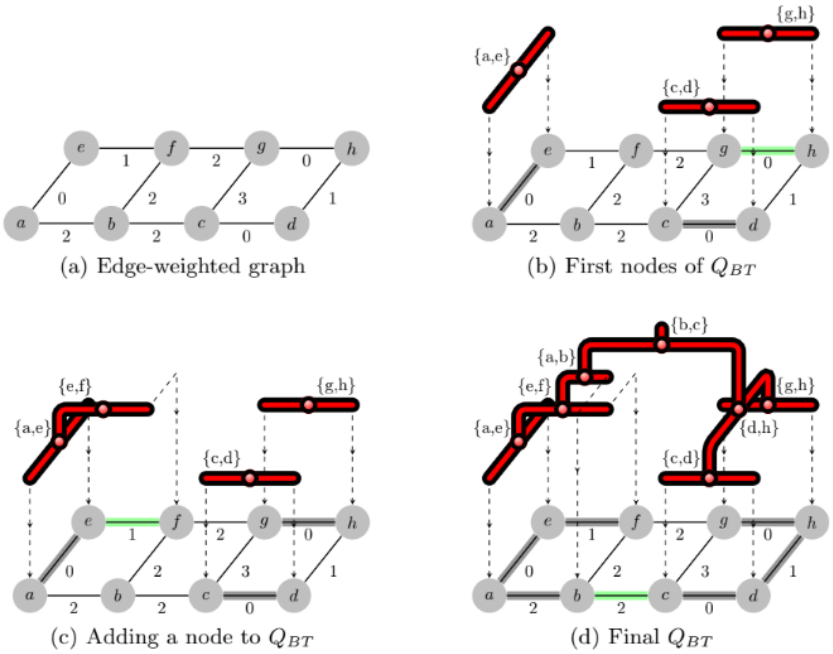


Figure 4: A simple process for obtaining a binary tree providing a strict total order relation on the edges of the MST [3]

We had some trouble with the interaction between Higra and Pytorch. In order to compute the loss, we have to compute a MST on the affinity image. Then we have to compute a BPT using the MST to find the maximin edge used in the loss computation. Our problem was the following: with a view to use Higra, the affinity image is turned into a graph, but by making this the gradient history is lost. Without this history it is impossible to train our NN. So, how to keep tracking the gradient history using Higra? We found the solution in a code proposed by XXXXXXXX. Higra has a function that allow us to make the correspondence between an edge in the graph and the output affinity image. Consequently, we are able to localise the maximin edge in the output image. Due to the fact that picking an edge does not cause gradient history loss we are done.

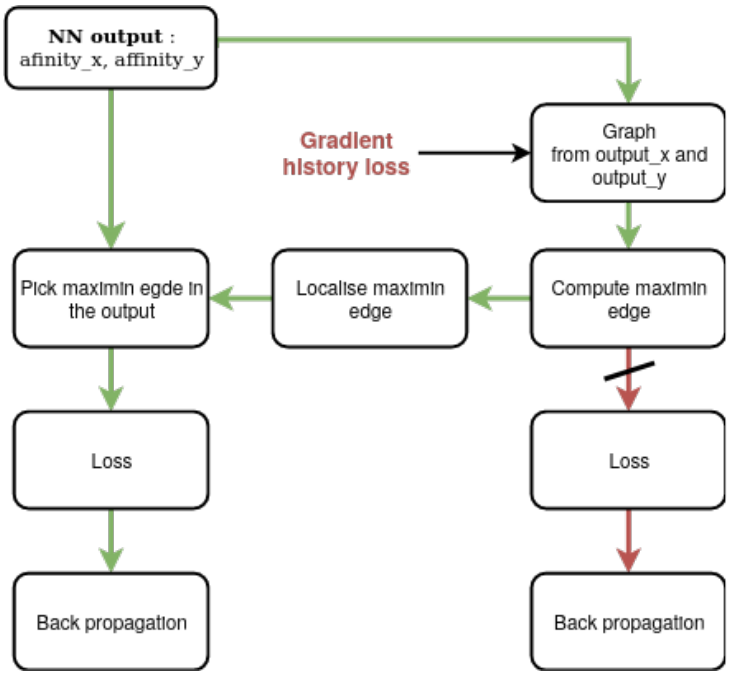


Figure 5: Green arrows represent out path.

5 Results

5.1 Evaluation method

We evaluated our method with two different datasets : CREMI and ISBI datasets.
Both are composed of Drosophila melanogaster adult brain images.

A very simple architecture is used for the training. It's composed of 6 layers of convolution.
In [1], the original architecture had 4 convolutionnal layers with 5 features in each. We decided to add more layers and features as it gave us better results. Batch normalisation was also added as it has also given us better results.

the architecture is described in detail in table 1

Layer	Kernel	Strides	Features	BN	Activation	Output shape
Input						(21, 21, 1)
Convolution	(5, 5)	(1, 1)	8	Y	ReLU	(21, 21, 8)
Convolution	(5, 5)	(1, 1)	32	Y	ReLU	(21, 21, 32)
Convolution	(5, 5)	(1, 1)	32	Y	ReLU	(21, 21, 32)
Convolution	(5, 5)	(1, 1)	32	Y	ReLU	(21, 21, 32)
Convolution	(5, 5)	(1, 1)	8	Y	ReLU	(21, 21, 8)
Convolution	(5, 5)	(1, 1)	2	N	sigmoid	(21, 21, 2)

Table 1: Architecture used in all of our experiments

5.2 CREMI

The CREMI (Circuit Reconstruction from Electron Microscopy Images) dataset has three volumes but we decided to use only one (volume A) for our training.
This 3D image has a size of 1250x1250x125. Its corresponding grountruth was also provided in the dataset. The segmentation has labeled connected components with really thin edges. For the evaluation, we used the CREMI library that was given with the dataset in Python 2. We, then, adapted it in Python 3.

(a) Original image (b) Groundtruth

Figure 6: CREMI dataset (volume A)

Our 3D image of size X x Y x Z was easier to predict in two-dimension. That's why we considered Z images of size X x Y which were stacked together to get back a 3D image.

A question arises : how to get an image segmentation from the affinity graph ?

We used two different methods to answer this problem.
First of all, a BPT (Binary Partition Tree) and a graph cut could get a good image segmentation.

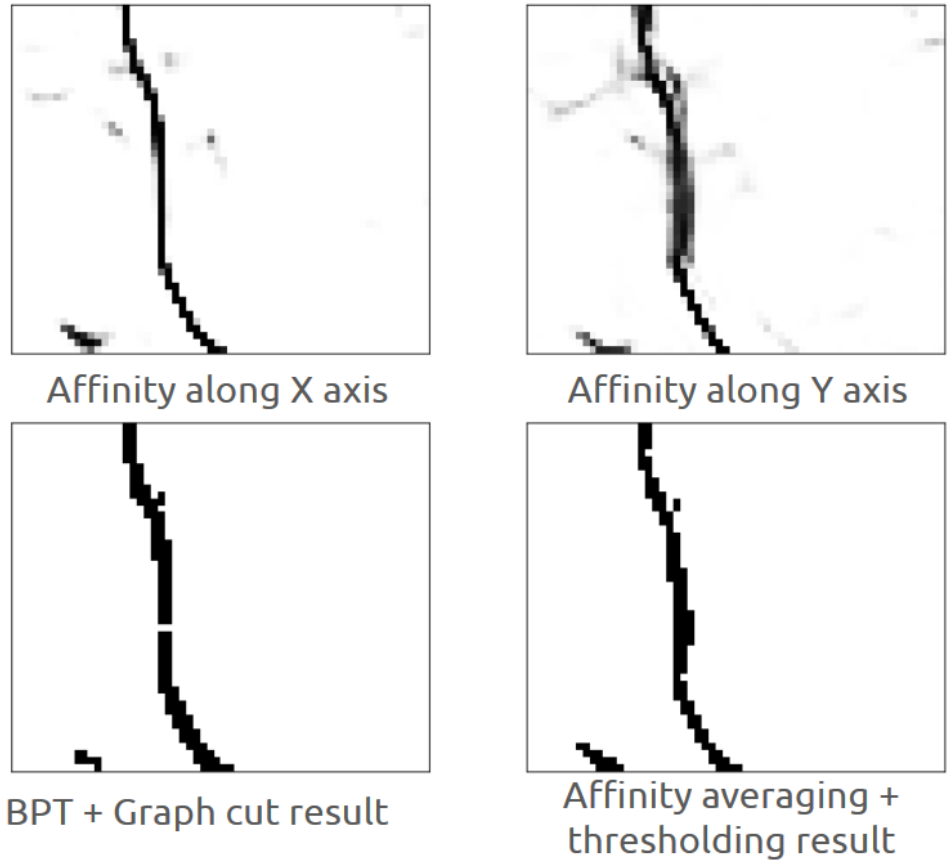
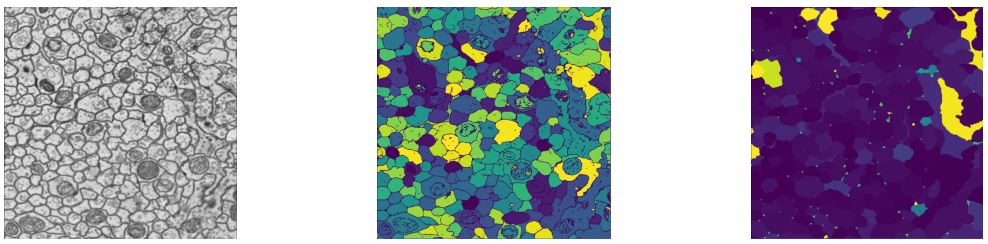


Figure 7: Illustration of the isthmus issue in our segmentation

However, even with a strong threshold (around 0.99), isthmus appeared and fusion two different objects together. It's a big issue as it affects our scores.
Secondly, to get rid of isthmus, we did an average affinity and it could segmented the objects nicely.
Isthmus issues can be solved by improving our post-processing, and it should disappear with a better architecture.



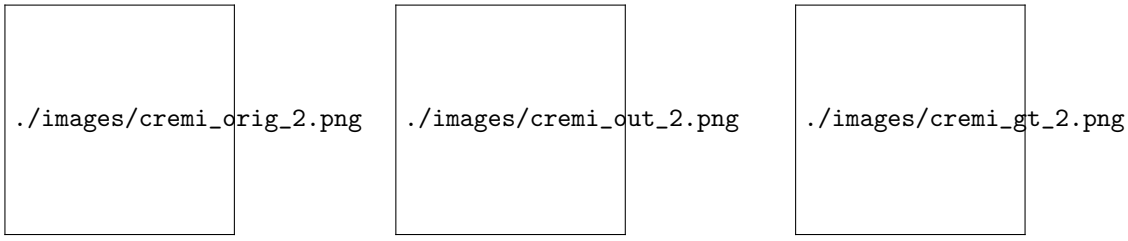
(a) Original image (b) Our segmentation (c) Groundtruth

Figure 8: Results of our training on the CREMI volume A

Our results are promising as the different objects are well segmented. Yet, there are a few oversegmen-

tations when regions are darker.
Nucleus are not detected as a same object as the cell.

add volume B test



(a) Original image (b) Our segmentation (c) Groundtruth

Figure 9: Results of our test on the CREMI volume B

The second volume (volume B) was used as the test set. The image was similar to the one in the first volume but the objects are more stretched out.
Here the objects are well segmented but it was harder on darker regions.

With more details, we can evaluate with numerical results, according to the Rand index and the VOI (variation of information) merge and split. "The Rand index is a measure of the similarity between two data clustering." *explain VOI*
The Rand index should be the highest as possible, closer to 1. The metric VOI should be lower to be better.

	Rand index	VOI merge (lower is better)	VOI split (lower is better)
MALIS : Training set (original architecture)	0.53	2.08	1.32
MALIS : Training set	0.61	1.25	1.03
MALIS : Test set	0.53	1.57	1.38

Table 2: Results on the CREMI dataset

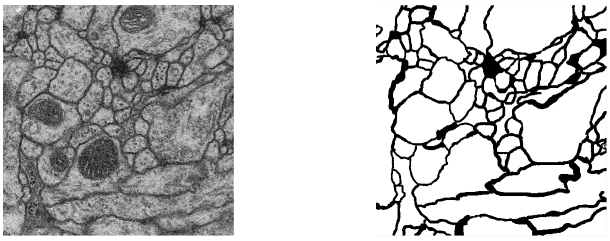
In the original architecture of 4 layers, the Rand index is 0.53 while we got 0.61 with our training set and 0.53 with our test set. Our Rand index is higher to the one of the original architecture. Moreover our VOI is low so it's corresponding to our desired outcome.

Thus, our results are hopeful knowing our architecture used was really simple.
It's still far from the state of the art but it could get even better with a more complex network.

5.3 ISBI 2012

In a second part, we evaluated our method on the ISBI 2012 Challenge dataset which is a set of 30 sections from a serial section Transmission Electron Microscopy (ssTEM) data set of Drosophila first instar larva ventral nerve cord(VNC).

The dataset also provides a corresponding groundtruth with labeled connected components, and thick contours.

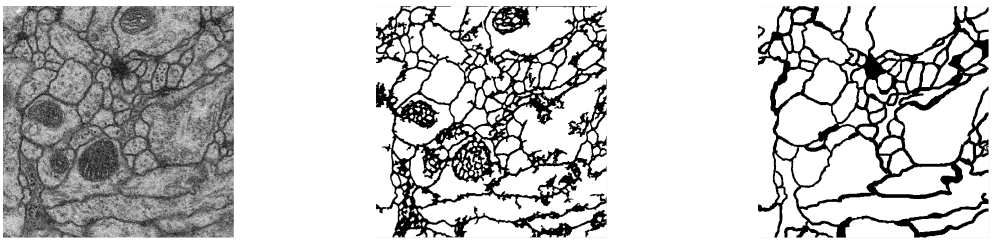


(a) Original image (b) Groundtruth

Figure 10: ISBI dataset example

There is no test set available because we should submit our method to the challenge's leaderboard to get our scores. The image has a size of 512x512x30 which is smaller than the CREMI's.
We evaluated using FIJI (Fiji Is Just Imagej) as an evaluation script was given for it in the challenge.

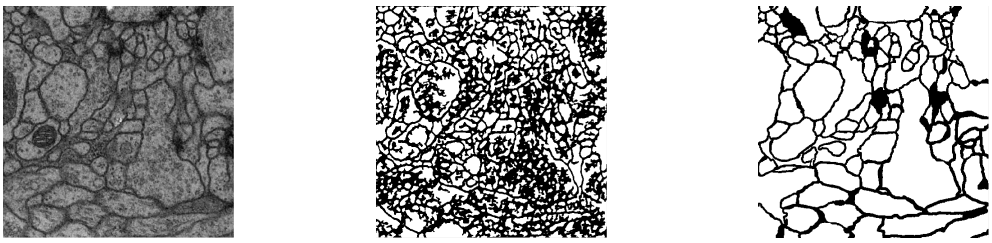
The same architecture that we used for the CREMI dataset was used for the ISBI dataset.



(a) Original image (b) Our segmentation (c) Groundtruth

Figure 11: Results of our training on the ISBI dataset

We still get good results even with a simple architecture, because the different objects are well seperated. However, there are oversegmentations for darker regions as well.



(a) Original image (b) Our segmentation (c) Groundtruth

Figure 12: Results of our test on the ISBI dataset

For example, in the figure above, there a lot of oversegmentations. This is one of worst image segmentation among the whole dataset.
This image is also the darkest, that's why we made the hypothesis that the oversegmentation issue come from the lack of contrast of the image.

To get numerical results we had to submit our method to the ISBI challenge.
We evaluate through the same metrics as for the CREMI dataset, that is to say the Rand index and the

VOI. This time the VOI should be higher to be better.

	Rand index	VOI
MALIS : Training set	0.76	0.89
MALIS : Test set	0.73	0.87
Thresholding : Test set	0.752	0.82

Table 3: Results on the ISBI dataset

The Rand index of our training set was 0.76 and 0.73 for the test test, which is similar but still higher than the thresholding with a Rand index of 0.72. Our VOI is also a bit higher than the threshold.
Those are better results than CREMI's probably due to the small size of the image and the thick borders of the segmentation.
It means we could have very promising results with our actual method. We could get even closer to the state of the art if we adopt an improved method.

6 Going forward

6.1 Improvement on MALIS

As we have seen before, MALIS performs really well, but can still be improved.
It was most notably improved in [4] where they were able to improve the affinity prediction using more recent architectures, by improving the training and taking full advantage of the MST and by applying a post-processing on the affinity graph instead of a simple thresholding.

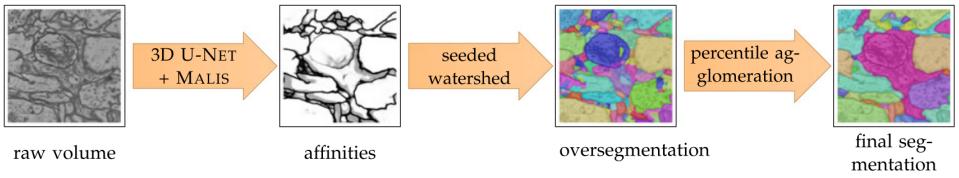


Figure 13: Improves MALIS as described in [4]

As we can see in figure 13 the CNN was replaced by a U-Net, which we will describe afterwards. Then the segmentation is obtained using a seeded-watershed, which is then improved using a percentile agglomeration of small objects.

6.1.1 Using a more potent architecture

Indeed, one of the limits of the previous method was the use of a relatively simple neural network to predict the affinity. This is mostly due to the fact that neural networks have greatly improved since the original paper [1] in 2009.

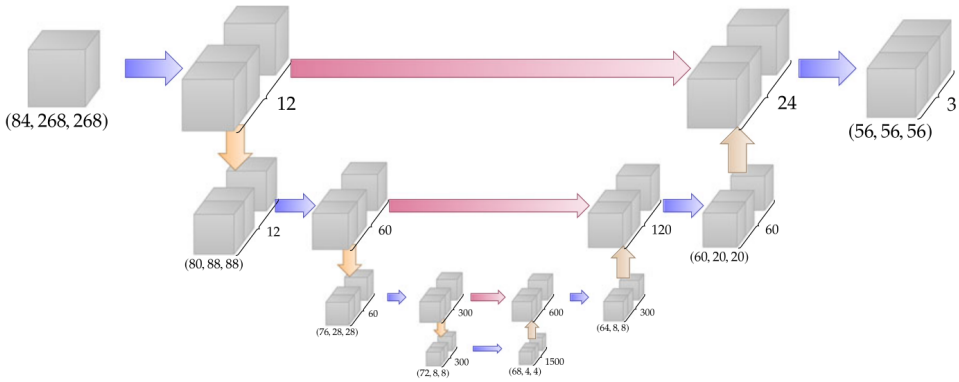


Figure 14: U-Net architecture used on the CREMI dataset from [4]

U-nets especially have been widely used for image segmentation and are thus a natural choice in our case. The architecture used in [4] is shown in figure 14. As we can see, the training will obviously be much longer than previously, but the results should be significantly better.
A question that can arise is why not simply use just a U-Net without the MALIS loss, and as we will see later, the MALIS loss gives us better results on differents measures of segmentation quality.

6.1.2 Constrained MALIS loss

Previously, we only computed the maximin edge for a pair of pixels (or a finite amount of pairs), which means that some information from the MST was not used.
However, we would like to compute the maximin edge for all pairs of pixels in the image. This was not

done in the previous paper for time efficiency reasons.
However they describ a way to compute the loss with in quasilinear time instead of in polynomial time.

The main idea is that all maximin edges are in the MST, and there are $n-1$ edges if we have n pixels in our image. We could then simply compute the loss over all those edges but this would mean that they are all as important as the others. However since we have n^2 pairs of points and $n-1$ edges in our MST, they will be the maximin edge for a different number of pixel-pairs. So we must find a way to see how often an edge from the MST is a maximin edge.

We won't go into much details here on how exactly this is done, but they are able to find for how many pairs of pixels an edge is a maximin edge when constructing the MST.
When we add an edge using Kruskal's algorithm, we can look at the "size" of the trees it merges and deduce the number of pairs for which the current edge is the minimax edge.

From this, we can define the positive weight of an edge e as the number of pairs from the same object/segment merged by adding e to the MST. More formally we have :

$$w_p(e) = |\{(u,v) \in F^2 \mid \delta(u,v) = 1, e = mm(u,v)\}|$$

Similarly we can define the negative weight of an edge as :

$$w_n(e) = |\{(u,v) \in F^2 \mid \delta(u,v) = 0, e = mm(u,v)\}|$$

These formulations allow us to rewrite our loss function as :

$$L(I,\theta,S) = \sum_{e \in MST(G)} w_p(e) l(1, A_e(I,\theta)) + w_n l(0, A_e(I,\theta))$$

With A_e the affinity of an edge e.

With this loss function being able to be computed in quasilinear time, this will allow us to use the whole image for the loss computation instead of a few pairs of pixels, which should improve the results. This loss function is the same as before, it still is related to the Rand Index, but it should now approximate it much better.

However this doesn't really solve the issue of class imbalance that we noted before. They suggest a way to resolve this by first computing the positive part of the loss first (with w_p) and then the negative part (with w_n). In both case, we set the other edges (the edges between objects for the positive pass) to their groundtruth value so that we only take into account a category of edges.
This means that we will computethe loss for edges between objects and edges inside objects separately, which allows use to combine them afterwards with the appropriate weights so which should resolve the class imbalance issue.

6.1.3 Seeded watershed as post processing

Remember that before, we computed the segmentation by thresholding our affinity graph. However this doesn't give optimal results, as for example locally another threshold would perform better.
An issue that was also encountered was small objects that were inside bigger ones. These objects lead to an oversegmentation and we would like a way to automatically remove those inaccuracies, or at least part of them.
This is where the framework described in [4] comes in play. As we can see in figure 15, the first step of the process is to average the affinities, as we did before. Afterwards the averaged affinities are thresholded at 0.5 (here this threshold is not a parameter). Then a distance transform (or a distance map) is computed from the objects to the borders. In this distance transform, the furthest points from the borders will

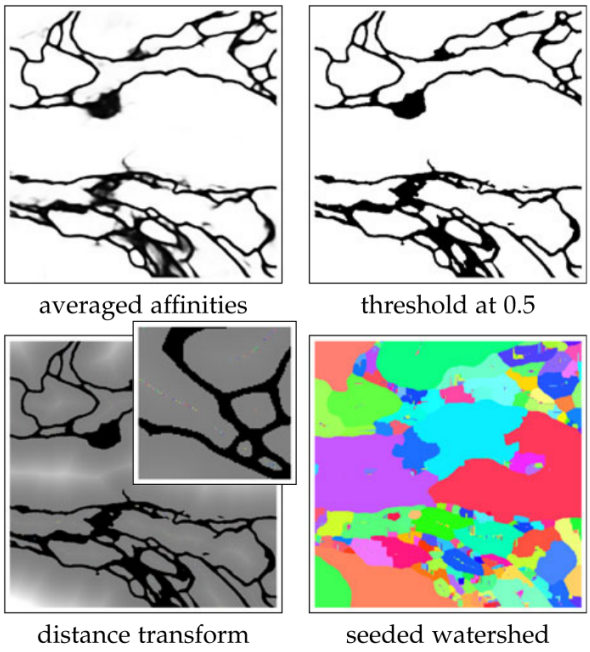


Figure 15: Seeded watershed on an affinity graph, as described in [4]

have the higher values. Then, all the local maxima are taken as seeds for a seeded watershed, which gives a a first segmentation.

This is still an oversegmentation, but a fragment aggloremeration algorithm is then used to fuse regions together. multiple criteria can be used to determine which regions to merge together and they are described in [4].
When all of those steps are done, we obtain the final segmentation.

As we can see the process is greatly improved from the first version of MALIS, with the loss function, the neural network architecture and the post processign used being much more powerful than their previous counterparts.

6.2 Goals for the second semester

The main goal for this next semester will be to implement this improved MALIS. The mains components to implement are as follow:

- Implement the U-Net
- Implement the constrained MALIS loss, with it's optimizations
- Implement the seeded watershed and fragment agglomeration
- Try other methods of post processing

Although this is a very important step and a large amount of work, the good thing with theses improvements is that we can still use parts of the older method while implementing new ones. For example we can try the U-Net with the first MALIS loss and no post processing, and the same is true for every new component that we will implement.

Once we have a working implementation the goal will be to try the method on various datasets. The CREMI dataset will provide us a baseline to compare our results with those obtained in [4]. We will also try the method on other dataset such as the BSDS500 dataset, which is composed of more varied types of images and not just connectomes.

These are our main goals for the second semester, but if we are able to finish all of this early, there are a lot of interesting experiments where we could further test the limits of this method.

7 Teamwork

7.1 Team organization

In order to have an overview of the project, we are using the software Trello. With this, we can write all tasks that we need to do, who are responsible of the task and the progression of the project: we can see what is already done and what the other members are currently working on. We communicate with each other using Slack in which we put information and additional contents related to the project.

Every week, we have a meeting with our supervisor in which we discuss about what we did during the week, our issues and solutions if we had some and what we will do afterwards. During each meeting, a different person will lead the discussion. We also write a report every week in which we write what we did in the week to prepare for the meeting. We write all the codes in documented Notebooks that we will clean and put on GitHub.

Until now, we managed to implement the first paper and train on different datasets. We have some good results as well but it can be better. We will now implement the U-net in 2D and 3D and try to implement a new loss, which will be computed from the computation of the MST. We will also train on BSDS and apply different post processing.

After knowing what we wanted to do, we split the work based on preferences of everyone. Quentin was chosen team leader, he worked on a lot of things in the project and supervised the other team members. Annie worked on the dataset, Josselin on the computation of the loss and the maximum spanning tree and Tiphanie worked on the evaluation and image generation.

7.2 Task distribution

Annie	Josselin	Quentin	Tiphanie
Find dataset	Computation of MST	Finalize the saliency notebook	Affinity graph thresholding
Exploration of dataset	Computation and optimisation of the path between i and j	Add features in the saliency notebook	Computation of connected components
Analyze what is the intput and output of the neural network and their size	Computation of the maximum path in the MST	Graph generation from output of neural network	Image generation
Understand how to use hdf files	Computation of the loss	Preparation of the dataset	Finishing inference by creating segmentation
Create architecture of convolutional neural network	Find interesting patches in dataset	Get vertices pairs in same and different object	
Load ISBI-2012 data	Train and evaluate on ISBI in 2D	Train the network on maximin affinty	
		Image reconstruction with inference	
		Fiji for evaluation on ISBI	
		Load ISBI-2012 data	
		Train and evaluate on ISBI in 3D	

7.3 Obstacles and overcoming them

The subject of this project is quite difficult at first because we did not have any basis on image segmentation. Moreover, it was also hard to read and understand the scientific paper because it was written in English and the content was not explained well. To overcome this difficulty, we planned several hours to go through the important things that we needed to retain. After knowing what we must do, the next step was how to implement these ideas. We took some time to know exactly what information we needed and how to use them for the implementation. Once we knew, it was not hard really hard to code with the library Higra, we just needed to find the right functions. Afterwards, we choose to use the PyTorch library, which is easier for us to compute our code. Having no experience with this library, we needed to understand how to use it. There is a tutorial online allowing us to see the basis of PyTorch, which is quite easy to understand.

During this project, we received a lot of help from Quentin. This project would have been really hard to do if he was not there, having no great knowledge in image segmentation and machine learning. Thanks to him, we were able to have a better understanding of the papers by explaining to us. It would have taken us a lot longer to understand the papers and how to implement them.

8 Conclusion

References

[1] S. C. Turaga, K. L. Briggman, M. Helmstaedter, W. Denk, and H. S. Seung, "Maximin affinity learning of image segmentation," *arXiv:0911.5372 [cs]*, Nov. 2009. arXiv: 0911.5372.

[2] S. Turaga, "Learning image segmentation and hierarchies by learning ultrametric distances," Apr. 2010.

[3] L. Najman, J. Cousty, and B. Perret, "Playing with Kruskal: Algorithms for Morphological Trees in Edge-Weighted Graphs," in *Mathematical Morphology and Its Applications to Signal and Image Processing*, vol. 7883, pp. 135–146, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[4] J. Funke, F. Tschopp, W. Grisaitis, A. Sheridan, C. Singh, S. Saalfeld, and S. C. Turaga, "Large Scale Image Segmentation with Structured Loss Based Deep Learning for Connectome Reconstruction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 1669–1680, July 2019.

[5] G. Chierchia and B. Perret, "Ultrametric Fitting by Gradient Descent," *arXiv:1905.10566 [cs, stat]*, Oct. 2019. arXiv: 1905.10566.

[6] A. Challa, S. Danda, B. S. D. Sagar, and L. Najman, "Watersheds for Semi-Supervised Classification," *IEEE Signal Processing Letters*, vol. 26, pp. 720–724, May 2019.

[7] S. Wolf, L. Schott, U. Köthe, and F. Hamprecht, "Learned Watershed: End-to-End Learning of Seeded Segmentation," *arXiv:1704.02249 [cs]*, Sept. 2017. arXiv: 1704.02249.