A U-Net like Solution to the 2018 Data Science Bowl Challenge

François Delarbre,[1] **Nathan Greffe**,[2] and **Pierre Keutgen**[3]

[1]*fdelarbre@student.uliege.be (s143409)*
[2]*n.greffe@student.uliege.be (s141045)*
[3]*pkeutgen@student.uliege.be (s140887)*

I. ABSTRACT

The 2018 Data Science Bowl challenge was about spotting nuclei in divergent microscopy images. We framed the problem as an instance segmentation problem. We developed a solution based on a deep neural network with a U-Net like architecture. The network is trained to segment the nuclei from the backgrounds and to highlight the borders of the nuclei and their centers. A watershed transform is then applied as a post-processing step in order to split the segmentation mask "is a nuclei" into different nuclei. Several encoders pre-trained on the ImageNet dataset were compared, as well as the idea of attaching more importance into the boundaries of nuclei in the loss function. In the end, our solution was based on a mobilenetv2 U-Net like architecture and was able to reach a score of 0.226 (out of a maximum of 1) on a part on the training set (that was not used for training the network or selecting hyperparameters), see Section V.

II. INTRODUCTION

Identifying cells nuclei in microscopy images is the starting point of many treatments development for major diseases. Indeed, the identification of nuclei allows the researchers to identify the different cells of human tissue and to track their response to different stimuli, especially exposition to new drugs molecules.

Identification of nuclei is made difficult by their various shapes and the fact that the boundary between several nuclei is not obvious in some microscopy images. In addition, there are a lot of different possibilities to capture microscopy images, and so is the appearance of nuclei in the images. For those reasons, classical computer vision techniques often fail at detecting nuclei and the biologists usually have to look at thousands of images by eyes.

Building a robust computer program that would be able to spot cell's nuclei accurately in various type of microscopy images would save a tremendous amount of hours to researchers, enabling faster development of drugs and treatments for serious diseases. It is in the hope of building such a program that the kaggle challenge *2018 Data Science Bowl: Find the nuclei in divergent images to advance medical discovery* was launched last year.

Inspired by the challenge we have decided to develop a solution. That was also the occasion to learn about semantic segmentation which is a problem that has many other applications in Biomed, robotics and autonomous vehicles.

The two main deep learning methods to solve semantic segmentation are Mask-RCNN [1] and U-Net-like models [2]. We have decided to experience with U-Net-like architectures. That choice was mainly driven by the good results that U-Net based solutions had achieved on the challenge.

This paper will describe the different steps and tests we went through during the development of our solution.

III. RELATED WORK

As said before, the two most common approaches used for semantic segmentation are based on either Mask-RCNN [1] or U-Net [2]. The first approach consists in using Mask R-CNN [1] to perform the instance segmentation in one step. Mask-RCNN [1] is a modification of Faster R-CNN [3] that handles instance segmentation. Without entering in too many details, Faster R-CNN [3] has been designed to perform bounding-box detection. The architecture is composed of a Region Proposal Network that proposes primary bounding boxes and of a second network that predicts classes and finer bounding-boxes. Mask-RCNN [1] builds on top of that idea. Region Proposal Network is used in the same way, but, in the second stage, in parallel to the classes and finer boxes predictions, an up-sampling network is used to perform semantic segmentation. Different instances of the same class are separated by the Region Proposal Network. Mask R-CNN is a meta-architecture, i.e. it is built on top of existing classification neural networks like ResNet [4] or ResNext [5].

On the other hand, U-Net [2] is a convolutional architecture where the tensors width and height are first reduced in an encoder and then increased in a decoder. The particularity of U-Net w.r.t. previous works like [6] is the introduction of skip-connections between the encoder and the decoder. That allows the decoder convolutions to use both fine-grained and coarse information about the image. The famous U-Net architecture is illustrated in Figure 1. Although the original U-Net paper mentioned a completely defined architecture, it is common to use so-called U-net like architectures. In a U-Net like architecture, the U-Net idea of skip connections is reused but with a different encoder/decoder architecture. Some examples of U-Net like architectures are [7] which is based on ResNet [4] or [8] which is based on DenseNet [9].

We also looked into Deeplab. DeeplabV3 [10] is the third version of the atrous convolution based network Deeplab. It has been designed for semantic segmentation as an alternative to classic U-net like encoder-decoder networks. The main advantage of Deeplab is to improve global consistency by increasing the respective field of each pixel. This feature is not suitable for nuclei segmentation as it does not involve objects with several different parts as object recognition does.

To train semantic segmentation networks, several improvements were made over the simple idea of applying binary or categorical cross-entropy. An example of improvement is to give more weight to the border pixels of the objects like in [2]. Another example is to use the soft-dice loss as in [11]. The later loss function (here for a single class and a batch size of 1) can be computed as

$$\begin{aligned} dice\_loss &= 1 - \frac{2 * intersection + 1}{union + 1} \\ &= 1 - \frac{2 * sum(y\_true * y\_pred) + 1}{sum(y\_true + y\_pred) + 1} \end{aligned}$$

where the sums are computed over all pixels. The "softness" comes from adding one at the numerator and the denominator. The loss intuitively forces the network to predict the class when it is present (otherwise intersection decreases) and to not predict it when it is absent (union increases).

Once the semantic segmentation is handled, post-processing methods to split several instances of the same object can be used. The watershed transform is an example of such post-processing methods. Watershed consists in marking an image with several markers; The markers are then grown simultaneously till they touch each other. At the end of the watershed, the image is therefore segmented into different regions that represent different instances.

The initial placement of the markers is of primary importance for the watershed to work. To help the good marking of the image, the semantic segmentation network can predict several masks. For example, the centers of the nuclei can be directly used as markers. The boundaries between adjacent cells also help to place good markers. In another context but needing to predict several related masks as well, [12] used several communicating decoders to predict the different masks. A simpler method would be to use different networks with one class or a single network with several classes that are either mutually exclusive (softmax activations in the last layer) or not (sigmoid activations). [13] proposed to improve the classical Watershed transform by using deep learning to learn the topology structure rather than using the intensity of the input image.

IV. OUR APPROACH

Our approach to tackle the challenge consisted of using an U-Net like network to perform semantic segmentation and then to split the different nuclei from each other using mostly the watershed transform. The U-Net encoder was pretrained on ImageNet (we also tried other approaches but they were less successful). Section IV A presents the libraries we used as well as the hardware we ran our networks on. We explain how we compared the different approaches we tested in Section IV B. The images preprocessing, segmentation and post-processing are detailed in Sections IV C 1, IV D and IV E respectively. Finally, we comment on the results we obtained in section V.

A. Libraries and hardware used

For this work, the main libraries we used were Tensorflow version 1.13.1 as well as Keras, albumentations and OpenCV. Keras usage might be surprising since it is integrated into Tensorflow 1.13.1. However, tf.keras.applicaitons does not have as many models as Keras.applications (namely ResNext50). There might be a workaround using tf.keras but we did not find any help on the internet and we, therefore, decided to install Keras instead. Albumentation is a very complete and user-friendly library allowing to perform data augmentation. OpenCV is a well-known computer vision library, we used it to perform morphological operations (erosion, dilation), distance transforms (to get the centers of each cell) and the watershed transform.

For the hardware part, we used two computers to train our networks. The first one is a desktop computer equipped with a Nvidia gtx 970 (4 GB of VRAM) and the second one is a server equipped with a Nvidia rtx 2070 (8 GB of VRAM). We initially only disposed of the 970 desktop. The computational resources were a problem during the whole project. That influenced some of our design choices.

B. Performance assessment methodology

In order to assess the performance of a model, we used a validation set composed of 10% of the images (after a random shuffle with the same seed for each training), that is about 65 images. The validation images were created by applying 5-crops of the initial image (middle, top left/right, bottom left/right) without scaling. When using other data during training, we did not include any of them in the validation set. We only performed measurements on the semantic segmentation of the whole nuclei for simplicity. We assumed that the best network at identifying nuclei would be the best network at identifying

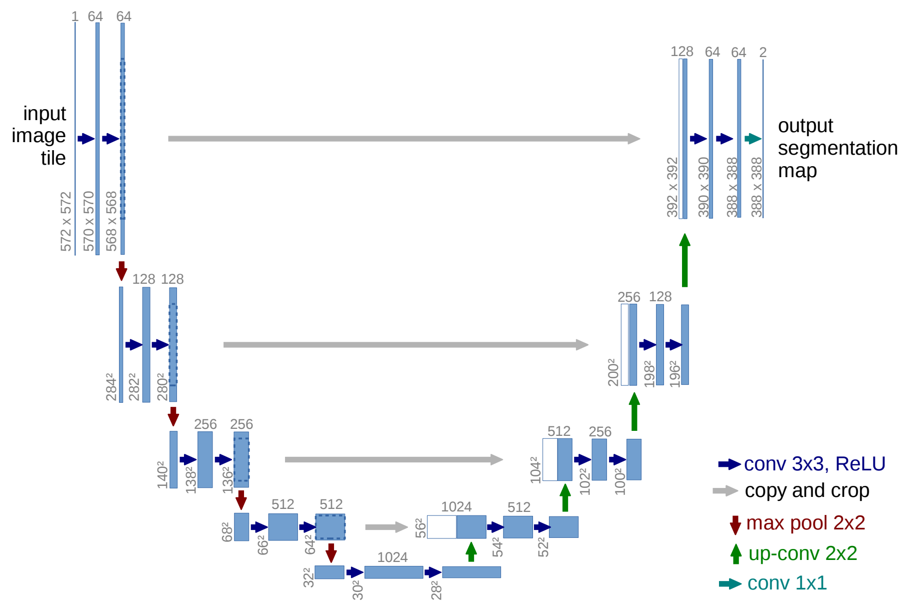


FIG. 1: U-Net architecture, image and description taken from [2]
U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

nuclei centers and nuclei border. The metric used for validation was the Sorensen-Dice coefficient with smoothing (adding 1 to the numerator and denominator); including smoothing to the metric makes no sense and was done by mistake.

$$softDC = \frac{2 * sum(y\_true * y\_pred) + 1}{sum(y\_true + y\_pred) + 1}$$

However, in order to use the same metric throughout the project and since the metric is a monotonic transformation of the classical Dice coefficient (since num. and den. are positive), we kept it. The metric is called soft DC in the rest of this document.

Due to the huge training time needed and the many parameters (architecture, data augmentation, additional datasets to use, loss function, optimizer, depth, ...) to optimize, we did not have the resources to compute rigorous comparison plots on several runs with confidence intervals for every parameter combinations. Instead, we incrementally built our model by choosing the parameters that we observed to work best.

Additionally, we implemented data visualization pipelines at several steps of the semantic segmentation learning. First, during training, we integrated to Tensorboard some images of the validation set as well as their predictions every ten epochs (*TensorBoardPredictedImages.py*). This was mostly used in the early stages of the project to get an idea of how well the model trained. We later developed a script to show the predicted and ground truth masks of the fully trained models for all of the validation set, one image at a time (*show_net_predictions.py*). To have a direct feeling of the performances of the model, we chose to put the ground truth mask into the blue channel of an RGB image and the predicted mask into the red channel of the same image. A resulting RGB image can be seen in Figure 2.

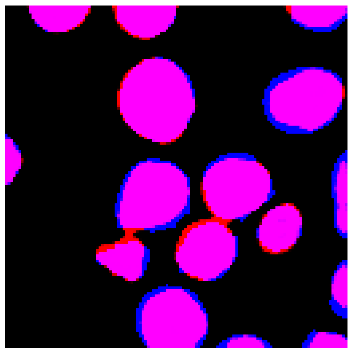


FIG. 2: prediction of the nuclei of the cells without differentiating different cells (sem. segm. performed by weighting border pixels more (see Section IV D 2)
A red pixel is a false positive, a blue pixel a false negative, a purple pixel is a true positive and a black pixel a true negative.

We also visualized instance segmented images of the test set of Kaggle to ensure our algorithm worked on the test set too. Figure 3 shows how the algorithm performed on some images of the test set of kaggle. We can notice that these images are qui different from the ones given in the labelled data.

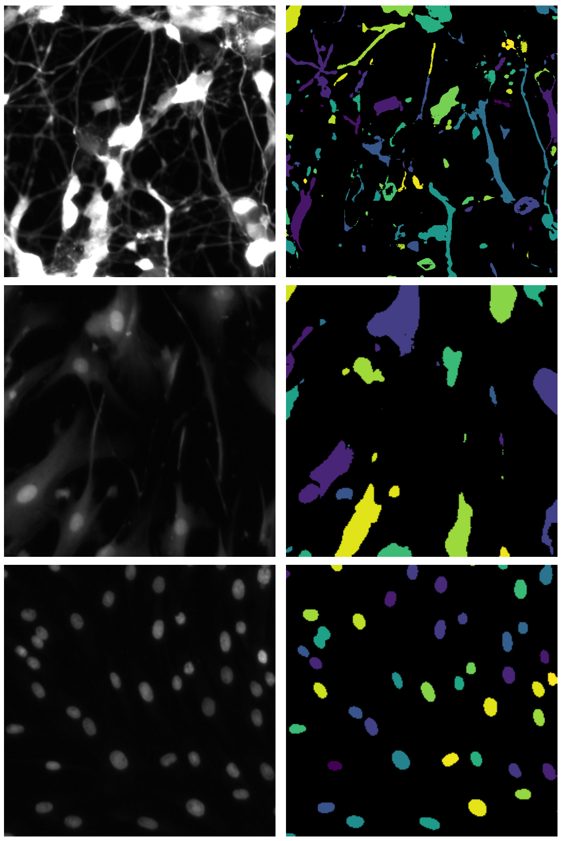


FIG. 3: prediction of the nuclei of the cells from image of the test set with instance differentiation

On the right is the original image and on the left is the corresponding instance segmented image

C. Prepossessing

We built a custom class to handle data loading and data augmentation (*data_generator.py*). Due to the small size of the base dataset (670 images with sizes of 256*256 and bigger) and its high heterogeneity, we chose to use heavy data augmentation and an additional dataset to use alongside the base one. We did not saw any noticeable improvements on the validation set (even when varying the strength of the data augmentation or the importance of the second dataset). However, since the testing data are very different from the training data in this challenge, we decided to stick with the data augmentation and the second dataset, assuming that the generalization benefits would be more important on the testing set.

1. *creation of the training labels*

In the challenge training dataset, the training label of an image was given in the form of a binary mask per nucleus. That is, each nucleus in the image appears alone into a binary mask. The first task, therefore, was to use OpenCV to group the different nucleus masks into a single one where all nuclei of the same image appear together. Then, because we wanted our models to predict not only whether pixels belong to a nucleus or not but also the center of the nuclei and the borders between nuclei, we had to create "centers masks" and "borders masks". The border masks were created thanks to OpenCV morphological operations such as dilation. The center masks were created thanks to the distance transform of OpenCV. The distance transform calculates the distance of each pixel of a nucleus to the closest border of the nucleus. The pixels that have the maximal distance represent a kind of center of mass that we chose to be the center of the nucleus.

2. *Data augmentation*

We used heavily data augmentation at training time. As said before, we were able to do so thanks to the albumentations library. To summarize the library API, the data-processing pipeline is specified as a sequence of operations to apply. We can also specify several operations that are mutually exclusive, i.e. we apply A with probability p, B with probability q and nothing with probability $1-p-q$. Both A and B are never applied at the same time. Each operation is specified alongside the probability to apply it for a given input image. Our pipeline is as follow (the meaning of each operation can be found on albumentations):

- with $p = 1$: we resize a random crop of the image to the network input size. The random crop can be bigger or smaller than the input of the network (between 2/3 and 3/2 of the input size) and will thus be rescaled to the network input size.
- with $p = 0.2$: apply optical distortion; with $p = 0.2$: apply Elastic transform, with $p = 0.1$: apply grid distortion; with $p = 0.5$: do nothing
- with $p = 0.5$: apply a random rotation of 0, 90, 180 or 270 degrees
- with $p = 0.5$: apply a random vertical flip
- with $p = 0.2$: apply a small color augmentation [14]; with $p = 0.2$: apply a medium color augmentation [15]; with $p = 0.4$: apply a huge color augmentation [16]; with $p = 0.2$: do nothing
- with $p = 1/6$: apply blur; with $p = 1/6$: add gaussian noise; with $p = 1/6$: add motion blur; with $p = 0.5$: do nothing

- with $p = 1/3$: randomly shift values for each channel of the input image; with $p = 1/3$: convert image to grayscale; with $p = 1/3$: do nothing;

3. *Using an additional dataset*

As we mentioned in the project proposal, external data usage was permitted by the challenge under the condition that it was mentioned in a specific thread. We used the data set provided by Costas Voglis. It basically consists of the dataset of the challenge MICCAI 2018 transformed to fit the database format of the kaggle challenge.

The dataset is composed of 30 histological images and about 22,000 nuclei annotations. The images of this dataset do not look like the data from the kaggle competition. We thought that it would add diversity into the training data and therefore help the model to generalize.

Since the images in this dataset are more or less 16 times bigger ($1000 * 1000$ instead of $256 * 256$) than those from the kaggle challenge but there are only 30 of them (instead of 670), we added them 4 times to the dataset. That means that, in one epoch, the network was given 4 different random crops of the images of the auxiliary dataset.

The other datasets proposed in the special thread of the challenge wasn't appealing to us. Lot of them didn't differentiate the nuclei instances or were encoded in a format that would require specific processing that we didn't have the time to manage.

D. U-Net based segmentation

The core of the semantic segmentation training procedure (*sem_segmentation.py*) is to train a U-net based architecture.

The training is performed by using Adam optimizer ($lr = 8e - 5$). Since it is what was used in the paper [8], RMSprop was also used for FC-DenseNet but with no success.

The loss was a weighted combination of categorical-cross-entropy and soft dice loss.

Tensorboard was used to visualize network progression.

The two main variables in our work were the network architecture we used and whether we attached more importance to the boundaries of nuclei or not. The files corresponding to the different architectures are in the *model* folder and the different losses are implemented in *loss.py*.

1. *Architectures*

Please, note that we built the decoder of a U-Net-like network as a mirror image of its encoder. Therefore, when we change the encoder (MobileNetV2/ResNet50/...), the decoder is modified as well.

a. MobileNetV2 [17] is a SOTA lightweight CNN architecture. This architecture is based on a "classical depthwise-separable block" ($1*1$ conv $\rightarrow 3*3$ depthwise-separable-conv $\rightarrow 1 * 1$ conv). Other particularities of this block are the fact that there are more channels inside of the block (input and output tensors of the 3*3 depthwise-separable-conv) than outside. The last $1 * 1$ layer has no activation.

Initially, we were reduced to use a gtx 970 with 4GB of ram while semantic segmentation is a computation-hungry task. Using MobileNetV2 was therefore very interesting since it allowed the code to run on the 970. Indeed, since the MobileNetV2 architecture uses input resolution multipliers, we could obtain weights pretrained on ImageNet with smaller resolutions ($128 * 128$ instead of $224 * 224$). This also allowed us to train with bigger batch-sizes (20 for $128 * 128$ MobileNetV2 and 10 for $224 * 224$ MobileNetV2 instead of at most 5 on the rtx 2070 for the other networks)

Most of the networks hereafter used weights pretrained on ImageNet. Out of curiosity, we tested if it has really an impact. Training the network from scratch (on MobilenetV2 with an input size of $128 * 128$) got us nearly the same results on validation (validation soft DC 2% worse, training soft DC 4% worse).

Changing the input feature map size from 128 to 224 gave us better results (approximately 2 %). Surprisingly, the other networks are not performing better than $224 * 224$ MobileNetV2 and are several times slower to train. We thus used $224 * 224$ MobileNetV2 in our final submissions.

b. ResNet50 [4] is a well-know architecture that introduced skip_connections. We used it as an encoder with pretrained weights for ImageNet.

c. ResNeXt50 [5] is a modification of ResNet that uses depthwise separable convolutions. We used it as an encoder with pretrained weights for ImageNet.

d. FC-DenseNet [8] is a U-Net variant based on DenseNet (without pre-training on Image-Net). We re-implemented the whole network based on the paper. We tried both 50 and 103 layers variants. We failed to do more than a validation soft DC of 45% (against 92% for $224 * 224$ MobileNetV2) by using RMS-prop with the same options as the paper. By using Adam, we managed to achieve 72 and 88% for 50 and 103 layers respectively. The 50 layers variant has not completely converged but the bad results and the huge training costs made us drop that architecture.

See figure 4 to have a comparison between ResNet50, ResNeXt50 and MobileNetV2. We can also see on that figure that the validation soft DC is quite unstable. It is especially true for ResNet and ResNeXt (although we do

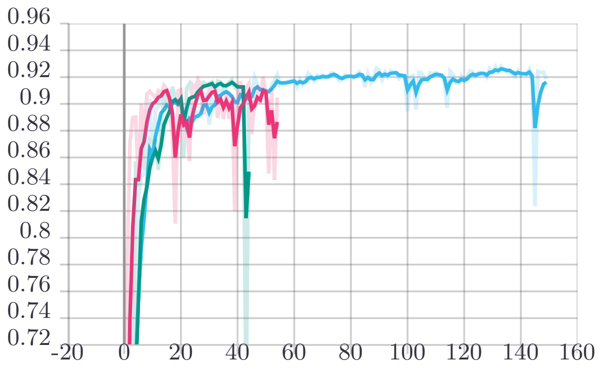


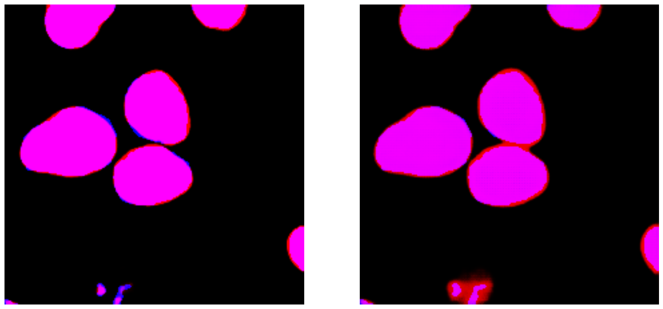
FIG. 4: smoothed Soft Intersection over Union on the validation set (downloaded from tensorboard)
MobileNetV2: blue; ResNet50: pink; ResNeXt50: green

not have enough data to reliably get to that conclusion).

e. Deeplab V3 [10] is a atrous convolution based neural network. An implementation is available in the Tensorflow framework. However, this implementation is hard to retrain as it does not use the Keras interface. We also tried to use a Keras implementation of Deeplab but it was very buggy with either version 1.13 or 2 of the Tensorflow API. We did not push the research further as our focus was on U-net like networks.

2. *Pixel weighting*

An idea to improve semantic segmentation is to weight the border of the nuclei more importantly. That means that miss-classifying pixels at the border of a nucleus would be worse than miss-classifying pixels at the center of a nucleus or background pixels. A similar idea was used in [2] for other purposes. As we can see in Figure 5, weighting the borders more heavily partially solved the fact that we miss-classified them as background. However, it also decreased the soft DC by 1 to 2 %. We can see in the figure that we trade-off less false negatives for more false positives. We finally decided to use the weighted version because it performed better on the final (including postprocessing to split different cells) metric, see Section IV E.



(a) without weight masks (b) with weight masks

FIG. 5: prediction of the nuclei of the cells without differentiating different cells
A red pixel is a false positive, a blue pixel a false negative, a purple pixel is a true positive and a black pixel a true negative.

E. Postprocessing

In addition to predicting the pixels that belong to nuclei, we use different U-nets (with the exact same architecture) to predict the centers of the nuclei and the borders between nuclei. That information can be used to perform instance segmentation. We have made three simple and (we think) fare assumptions in regards to the different images:

- the nuclei in a given image have pretty much the same size;
- nuclei are quite round shaped;
- the area of nuclei that overlap is usually small.

Based on those assumptions, We have experimented with three different post-processing strategies.

The first one is solely based on the ”is a nuclei” mask. The idea is to iterate through all blobs predicted by the network and see if the blob is made up of several nuclei. To do so, we compute the distance transform of the blob. That gives the distance between each pixel of the blob and the closest border of the blob. Because of our assumption, a local maxima within the blob should correspond to a nucleus center. The result of the distance transform on an image can be seen in Figure 6. The local maxima are used as a marker for the watershed. The watershed is then performed and the different nuclei are separated.

The second method uses the border between nuclei predicted by the network in addition to the ”is nucleus” mask. The border mask is subtracted from the ”is nucleus” mask in order to obtain a mask where more nuclei are already separated. Then, the first technique is applied to the resulting mask.

The third technique consists of using the centers predicted by the network directly as the markers for the watershed.

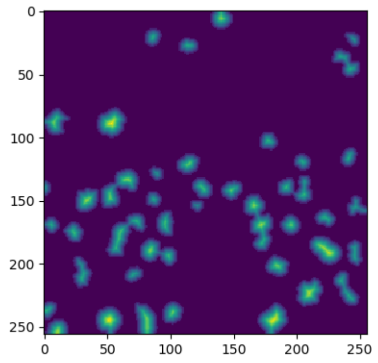


FIG. 6: result of the distance transform

We have tested each of those techniques on our validation set to find which one is the best. We tested the masks produced by both the "classical" network and the one trained with an emphasis on the nuclei's border. The metric we used is the same as in the kaggle competition. First, an intersection over union matrix is computed. The matrix gives the intersection over union of each predicted nucleus with each ground truth nucleus. Then, the values of the matrix under a certain threshold t are set to zero and the other are set to 1. A score is computed thanks to the formula

$$\frac{TP(t)}{TP(t)+FP(t)+FN(t)}$$

Where TP is the number of true positive at the threshold value t, FP the number of false positive at t and FN is the number of false negative at t. A true positive is counted when a single predicted nucleus matches a ground truth in the IOU matrix. A false positive indicates that a predicted nucleus has no associated ground truth nuclei. A false negative indicates that a ground truth nucleus has no associated predicted nuclei. The score is calculated for t in $[0.5, 0.55, 0.6, ..., 0.95]$ and the different values are averaged. That gives the final score of the image. The score of all images are then averaged and that gives the metric that we used. The results of the validation set are shown in table I. What we see is that the network trained with an emphasis on the border in association with the watershed that uses the center mask gives the best result. We, therefore, kept that configuration for our final solution.

technique	1	2	3
classical	0.236	0.240	0.265
border emphasis	0.248	0.253	0.279

TABLE I: score on the validation set using the three post processing techniques on the predictions of the classical model and those of the model that have an emphasis on the nuclei's borders

V. RESULTS

We saw that the U-Net-like network based on MobileNetV2, trained with a loss that penalizes more the border of the nuclei and combined with post-processing that uses the center of the nuclei gives the best results on the validation set. Unfortunately, we were not able to submit our results to get a score on the test set via kaggle. All of the archives we submitted ended up giving back a score of 0. We are confident into the correctness of our submission procedure because we managed to get the masks back from the encoding submitted to kaggle. Such masks can be seen in Figure 3 and altough not looking perfect they definitively do not deserve a score of 0. We were therefore left without testing set available late in the project progress. If we had known that fact at the beginning of the project, we would have divided the labeled data into a training set, a validation set and a test set. Unfortunately, at the end of the project, we didn't have the time to do the fine tuning of the hyperparameters again and what we did instead is illustrated in Figure 7. Doing so avoided the methodological error to compute the "final performance metric" on data that was used to train the network or to fine-tune the hyperparameters. This gave us the score on the test set of 0.226 presented in the introduction.

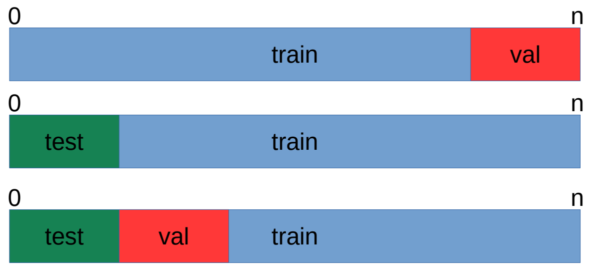
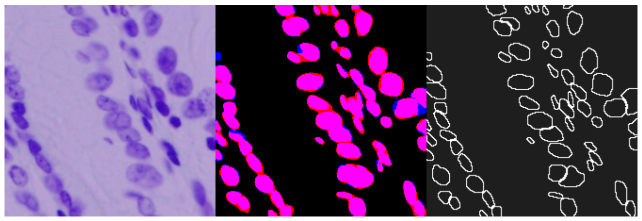
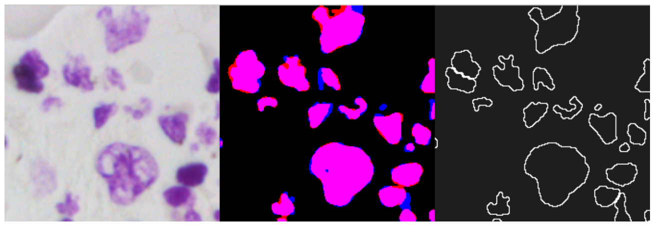


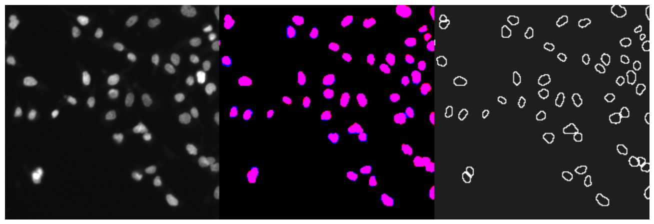
FIG. 7: Illustration of the testing/validation procedure
top: our dataset at validation time; middle: our dataset at test time; bottom: the usual approach

VI. ANNEX

The following are masks generated by our network trained with the border of the cells weighed more heavily. Figure 8 shows the semantic segmentation of images predicted by the network along with the masks used to increase the importance of the borders. Figure 9 shows the predicted centers of the nuclei for some images. Both figures show data from the test set as described in Section V.


(a) image 1


(b) image 2


(c) image 3

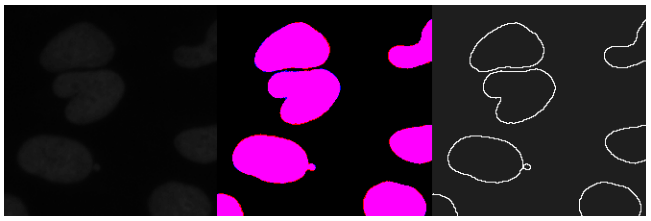
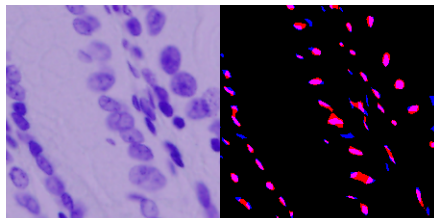
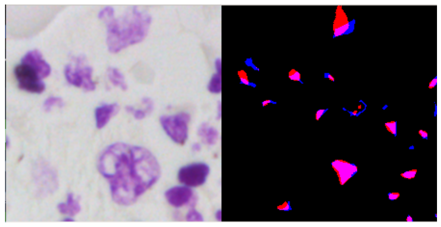
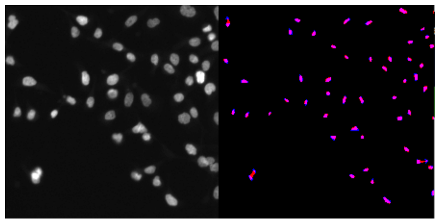

(d) image 4

FIG. 8: prediction of the semantic segmentation mask of the network with different weights for the borders of the nuclei
left: input image, middle: predicted and given semantic segmentation masks, right: nuclei borders


(a) image 1


(b) image 2


(c) image 3

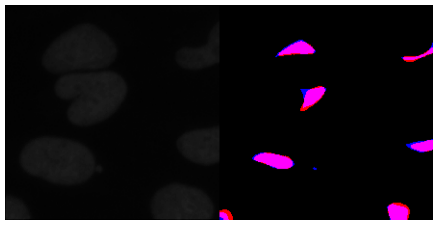

(d) image 4

FIG. 9: prediction of the center of the nuclei
left: input image, middle: predicted and given center masks

[1] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv:1703.06870 [cs]*, March 2017. arXiv: 1703.06870.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. arXiv: 1505.04597.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, June 2015. arXiv: 1506.01497.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[5] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv:1611.05431 [cs]*, November 2016. arXiv: 1611.05431.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *arXiv:1411.4038 [cs]*, November 2014. arXiv: 1411.4038.

[7] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The Importance of Skip Connections in Biomedical Image Segmentation. *arXiv:1608.04117 [cs]*, August 2016. arXiv: 1608.04117.

[8] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. *arXiv:1611.09326 [cs]*, November 2016. arXiv: 1611.09326.

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, August 2016. arXiv: 1608.06993.

[10] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[11] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. *arXiv:1606.04797 [cs]*, June 2016. arXiv: 1606.04797.

[12] Hao Chen, Xiaojuan Qi, Lequan Yu, and Pheng-Ann Heng. DCAN: Deep Contour-Aware Networks for Accurate Gland Segmentation. *arXiv:1604.02677 [cs]*, April 2016. arXiv: 1604.02677.

[13] Min Bai and Raquel Urtasun. Deep Watershed Transform for Instance Segmentation. *arXiv:1611.08303 [cs]*, November 2016. arXiv: 1611.08303.

[14] randomly change brightness and contrast with $p = 1$, followed by random change in gamma with $p = 1$, followed by contrast limited adaptive histogram equalization with $p = 1$.

[15] contrast limited adaptive histogram equalization with $p = 1$, followed by a random change of hue, saturation and value with $p = 1$.

[16] randomly shuffle the channels with $p = 1$.

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]*, January 2018. arXiv: 1801.04381.