Project 2 - Detecting the Degree of Cavitation In Situ in Young Trees

Théophane Mayaud, Davide Rosso, Alon Tchelet
With the EPFL PERL lab, supervised by Charlotte Weil, Switzerland

***Abstract*—As part of Project 2 (option A: ML4Science) of CS-433: Machine Learning EPFL, we worked on recognizing "air bubbles" in images of sections of trunk of plants obtained by an experiment of the Plant Ecology Research Laboratory (PERL) of EPFL using machine learning. We trained a U-net (a convolutional NN architecture) with different parameters and using two different types of labels. We compared the models obtained and analyzed their predictions on some images as validation set. The project code, Jupyter Notebooks and model parameters are available at https://github.com/CS-433/cs-433-project-2-ml_fools.**

I. Introduction

Every plant has inside its trunk a certain number of vessels where water, essential for the life of a plant, passes going from the roots to the leaves. Number and dimensions of vessels are different from plant to plant. During drought, the air present in a vessel with the water can expand (**cavitation process**) making the vessel unable to transport water. Considering a section of the trunk, in vessels where this phenomenon happens, we can see the presence of 'air bubbles'.
The Plant Ecology Research Laboratory (PERL) at EPFL did an experiment about cavitation using three species of three years-old plants; Quercus pubescens, Quercus ilex, and Fagus sylvatica. For their research, each species was subjected to various drought conditions. To analyze the cavitation phenomenon, each plant was scanned using X-ray microtomography techniques. To assess the effect of drought, each plant was scanned twice - once when still alive, and the second time after it was cut off and all of its vessels have been flushed.
In this way, it is possible to approximate the effect of cavitation by calculating the ratio between the area of vessels full of air (embolism vessels) in living plants and the total area of vessels. The total vessel area is given by the total area of embolism in the flushed images, where all vessels are visible.
The aim of the project was to generate a machine learning algorithm which receives as input an image of the section of the trunk of a plant and automatically recognize where the embolism vessels are. In other words, the algorithm should be able to label all pixels containing an embolism vessel from the provided image. In this way, by inputting a slice image from a living and a flushed scan of the same plant, the ratio could be calculated as described before. Therefore, this project could create a useful tool that can assist the lab in their research to analyse the effect of cavitation in a particular plant.

II. Models and Methods

In this section we describe the methods used to realize the project. In particular, we analyze how we generate the labels for our data (images) and the machine learning algorithm used.

A. Generating labels for images

During two sessions of PERL's experiment, a total of 404 plants were fully analyzed - 97 living and 101 flushed in Session 1, and 101 living and 105 flushed in Session 2. However, as 47 scans were of bad quality, only 357 plants were used as our dataset. From each plant scan, a single scan slice image was selected for further analysis. The lab processed the images to calculate the area of the xylem, that is referred to as the Region Of Interest (ROI), and the area and number of embolism areas. As the shape of these objects are complex but generally circular, the data was saved as if each of the previously mentioned objects was a circle with estimated centre and diameters to produce the surface.
As the software did not save the data in image form, the team had to generate the label images from this data. The label images created are all binary matrices in the shape of the original images. Embolism pixels were set to 1 while the rest of the pixels were set to 0. The team used two sets of labels, as described below:

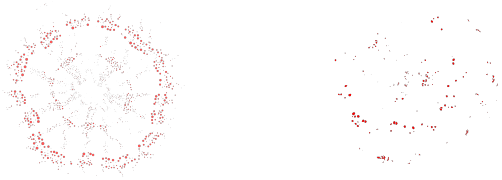
1) Generating masks using data analyzed by the lab from .CSV files: In this method, the labels of the embolism areas are realised using *.CSV* files of the 357 good analyzed plants with the data of the ROI and embolism areas. In this situation, every embolism vessels is labeled using a circle (even if, clearly, in term of pixels this is not a circle) with the centres and diameter reported in the *.CSV* files. The resulting label images are masks containing all embolism areas of a plant slice image in circles with the same surface area of the embolism regions above their centers.
This method's labels had a good representation of the area and location of the embolism vessels, but lacked in terms of the shape of the vessels.

2) Generating masks using signal processing and the data analyzed by the lab: In this method, the team tried to replicate the signal processing originally performed by the lab and take advantage of the analyzed data from the *.CSV* files. To analyze the data mentioned before, the lab used a software called *Avizo* that used four signal processing steps to extract the embolism areas. Each image was independently analyzed as the images differ from one another in resolution and contrast. The first stage was to reduce the noise in the image using

the tool's non-local means denoising function. The algorithm reduces the speckle noise generated by the scan and allows the image to be more efficiently thresholded. Second, by using a manual selection tool, the ROI was marked to exclude the background and bark, and then the image was thresholded. An adequate threshold value was selected for each image such that just the embolism areas are left without over estimating nor underestimating the embolism area. Furthermore, air clusters and air bubbles inside the heartwood were removed. Third, the embolism areas were isolated and finally saved into *.CSV* files.

To replicate this process in an automated manner, *Scikit-image* library was used to smooth the image with the method *denoise_nl_means*. To improve the performance of the method, the standard deviation σ of the image was estimated using the background of each image. Then, to avoid a manual identification of an optimal threshold value for each image, Otsu's method was implemented to calculate the optimal value to separate the background and embolism regions from the bark, heartwood, and xylem. The calculated value was then used as the thresholding value for the *OpenCV* method *threshold*. At that point, the mask contained all dark areas. Each independent region was labeled using the *SciPy* method *label*. Finally, to select only the embolism regions, the centers from the *.CSV* files were used to keep only the regions that matched a center. Regions that contained more that 1000 pixels were assumed to be too large to be vessels and were removed to avoid accidentally labeling the background as embolism.

This method generates far more representative labels, as their shapes better match the shapes of the original vessels, but it lacks as some vessels are missed and some unwanted regions were sometimes wrongly labeled ass seen in Fig 1.



(a) Label from CSV

(b) Label from Image processing

Fig. 1: Comparison of label generation, example of a bad situation

B. Data augmentation

To have more data for the training of a model, we implemented the possibility to generate more data based on the data obtained through the experiment of PERL. Two techniques were implemented; mirroring and rotating and adding noise to the image. With the flip-rotate augmentation 7 additional images could be generated from every existing image. While 3 noise adding methods were implemented in 2 Gaussian noise levels and a single speckle noise. Altogether, it allowed us to have the ability to achieve 32 times the original number of images.

C. U-net

Having output for each element of our sample of image, we focused on a supervised algorithm for the machine learning part. The fact of dealing with an image segmentation task and the type of input and output (images with binary output for every pixel) lead us to use U-net, a Convolutional Neural Network (CNN) developed for image segmentation of biomedical images (see [2] for the original paper). The implementation of the U-net was realised using PyTorch, with inspiration from Milesial's public code of U-Net [1].

1) U-net architecture: The U-net is composed by a lot of parts, in particular in the first half the U-net does a procedure of downsampling. Some operations are repeated several times, in particular, every time a convolutional layer is applied the kernel size is always $3X3$ followed by a $ReLU$ as activaction function where $ReLU(x) = \max(0, x)$. Moreover, when a max-pooling layer is applied the kernel size used is always 2 with a stride equal to 2. This last operation means to divide the input (considering each channel) in 2x2 pixels squares and take the maximum value between these 4 pixel values. At the beginning, a convolutional layer (as described above) is applied to the input image generating output with 64 channels, then another convolutional layer is applied starting from the previous 64 to arrive to other 64 channels. After this operation, the max-pooling layer is applied. After this, the U-net repeats for 3 times this sequence of operations: a convolutional layer which has as output a number of channels which is two times the number of input channel, a convolutional layer where the number of input and output channels are the same and a max-pooling layer. The sequence is repeated for a fourth time but without max-pooling, in this way at this point we have 1024 channels. Now it begins a part of upsampling. At first, it is performed $2x2$ upconvolution, realized using "ConvTranspose2d" of pytorch with kernel size $2x2$. This is equivalent to a normal convolution but with the input and output reversed, so in this case the output is larger than the input. The number of channels, generated with this step, is equal to an half of the previous one so 512 channels. To keep the same number of channels, a "copy and crop" operation is done using the 512 output channels obtained in a step of the downsampling part (as shown in Figure 1). After this step, again the U-net repeats some operations, in particular four times the following sequence of operations is repeated: a $3x3$ convolution with as number of output channels half of the number of the input ones, a $3x3$ convolution with the same number of output channels as the input and an upconvolution with half of output channels respect to the input. In particular at the end of the upconvolution the same number of channels is added coping and cropping the block (of the same size in term of channels) output of a convolutional layer in the downsampling process. In the end 64 channels of shape $388x388$ are obtained and a 1x1 convolutional layer

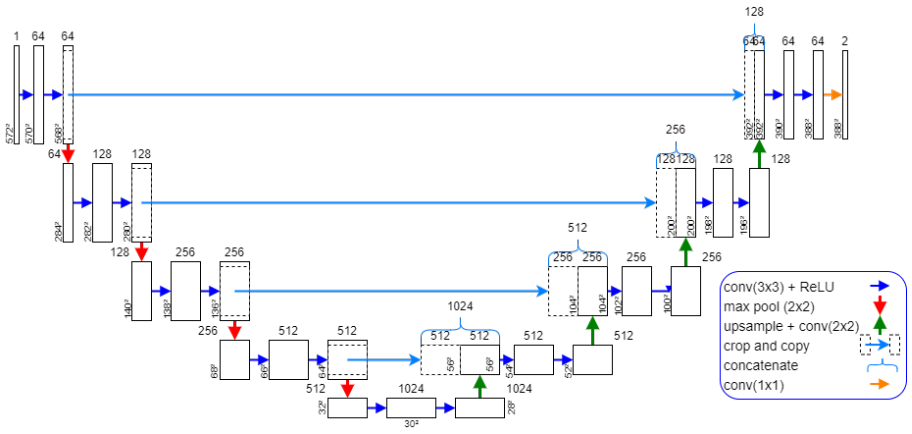


Fig. 2: U-net architecture

(without $ReLU$) reduces the number of channels to two. Each one is representative of class of the classification problem (Embolized/not embolized pixel).

2) Mirroring: From the architecture of the neural network it is possible to notice that input and output have different dimensions due to the presence of convolutional layers. If the original image is used as input, the output obtained is smaller than the labeled image. To solve this problem images are extended. The area added is created mirroring the external border of the original image. This choice was done following the work of the original use of U-net [2].

3) Segmentation of the image: The images inside the dataset are not square in general and, due to different resolutions, shapes are different from image to image. Some images can have a resolution of about $500X500$ pixels, while others of about $2000x2000$. The neural networks needs to work on input and output of a certain shapes, so it is impossible to train it using the images as they are inside the dataset. This problem was solved dividing the image and the correspondent labeled image in parts of square shape and of constant dimension. This operation was done on the image after the mirroring. In this way every segment of dimension $388x388$ of original image could be used as input with a border, formed by other parts of the image and by mirrored parts, in such a way that its final dimension was $572x572$. This is important in fact the output of U-net, given this input, is $388x388$ and can be used inside the loss function with the correspondent label ($388x388$) of the original segmented area. Segments of an image can have overlapping regions but this is not a problem for the training of the neural network.

4) Loss function: During the training to make prediction we used cross entropy loss. To choose the correct type of pixels for each one of them we pick the $argmax$ between the two channels for its relative position in the $388x388$ square.

III. Results

Here we will describe how we put together the models and methods to implement the different approaches we tried, and what we obtained from them. In particular we train the U-net with different parameters (type of optimizer, learning rate, number of epochs) and we compare the results on some test images. You can observe in Fig 3 some of the outputs, that we saved.

A. Results with different training of U-net (without parameters initialization)

We saved the results of different training. At the beginning, we tried to use data augmentation with few epochs. In this case, the method, due to the too few epochs, does not converge. We did not try with more epochs because the training would become too long in terms of computational time for the huge amount of images. Table 1 shows some results obtained using different hyperparameters, the labels are the ones generated from CSV files information. In the table is specified the number of images used, but, due to image segmentation, the NN is trained using all the segments generated from each image considered. In particular here the batch size is equal to 1, so the U-net is trained using segments one by one.

To evaluate the performance of each model obtained with different training parameters we at first looked at accuracy (number of pixels predicted correctly over the total number of pixels) but this showed to be a not meaningful evaluation parameter as the background area (pixels not embolized so black area) is very large. So accuracy can be large also if the model predicted a image more or less of all zero pixels. So we used a confusion matrix focusing the attention on true positive and false positive values. True positive is important because show how much of the embolized area (which is smaller respect to the background) is predicted correctly. More precisely true positive rate represents the fraction of embolized pixels predicted over all the pixels embolized in the labeled image; so the higher this value is the better our model perform. False positive rate measure the number of pixels predicted as embolized area, while these pixels in the label are not embolized area, over the not embolized area. So the lower this parameter is the better the model performs. Using the 197 images of session 2 (this is refered to a session of the

TABLE I: **Parameters and results for different trainings of the U-net**

training images	learning rate	optimizer	epochs	TPR	FPR
30	10^{-4}	Adam	500	0.75	0.0026
30	10^{-3}	Adam	30	0.57	0.0018
2	10^{-5}	SGD	500	0.35	0.005
30	10^{-6}	SGD (momentum=0.9)	70	0.53	0.009
160 (all session1)	10^{-4}	Adam	500	0.645	0.0016

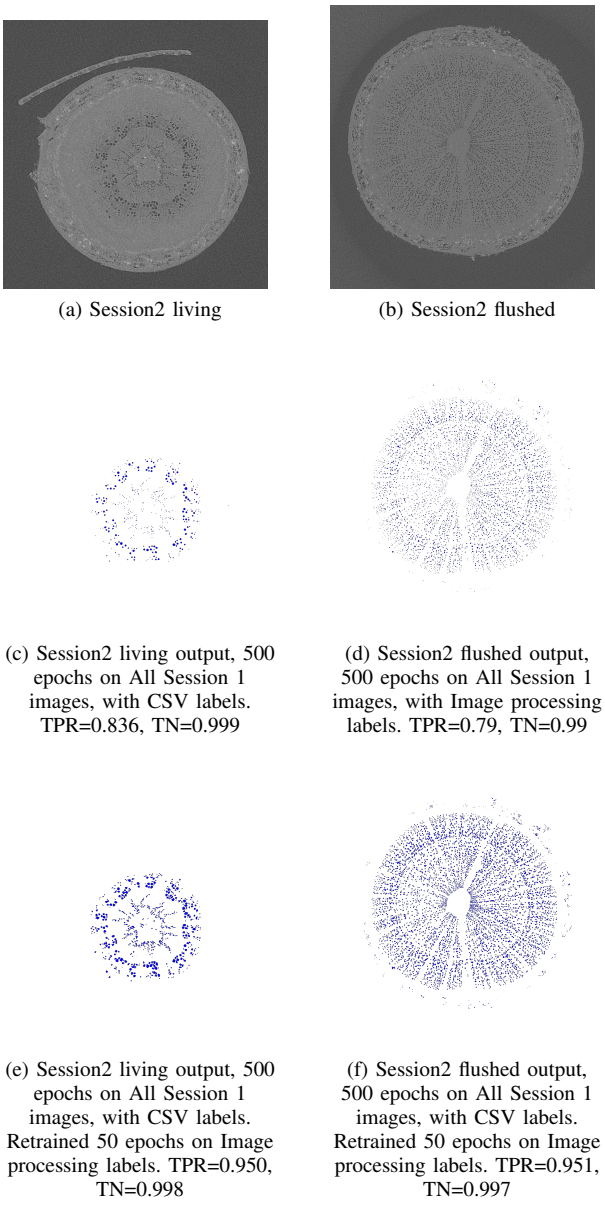
experiment) as test set in Table1, the mean True positive and False positive rates are shown to evaluate performance.

B. Results of a training using signal processing masks

The NN was trained also using the labels generated with the signal processing method and all images of session 1 (160 images) as training set. The optimizer used is Adam with learning rate of 10^{-4} and 500 epochs. The True-Positive Ratio (TPR) and False-Positive Ratio (FPR) were calculated from the ratios of correct embolism predictions over overall labeled embolism pixels and false embolism predictions over overall labeled background pixels. These ratios are complementary to 1 for the False-Negative Ratio (FNR) and True-Negative Ratio (TNR), respectively, from which one can calculate the confusion matrix. The mean TPR and FPR that were found using the same test set as in Table 1 are 0.884 and 0.0038.

IV. Discussion and Summary

From Table 1 it is possible to see that many training (the ones from row 2 to row 4) does not give a good model. Focusing the attention on the first and last training in the table it is possible to notice that they have the same learning rate, optimizer and number of epochs. The training with 160 images gives worse results in terms of True positive rate even if it uses more images to train. This is probably due to the fact the the first training converge faster while the other one, for the presence of more samples, requires more iterations to arrive to lower values of loss function, even if potentially it could be more general and better than the training with few images. This comparison shows also that the creation of labels is important for the training of the model. Looking at the models obtained, also when the model has good parameters, for some images the embolized areas are predicted very well while for others not so well. The training shown in the last row of Table1 has the same parameters of the training done using signal processing masks. The only difference is that the first one uses masks generated from $.CSV$ files. It is possible to see that the training with signal processing mask gives better results. In fact these masks embolized area are more similar to the reality while the other mask using circles as labeled mark some pixels equal to 1 also if they are not part of an embolized vessels (in fact these regions are not circular in general). A possible improvement of the results can be done training the U-net starting from parameters of some pre-trained model, used for similar purpose.



(a) Session2 living

(b) Session2 flushed

(c) Session2 living output, 500 epochs on All Session 1 images, with CSV labels. TPR=0.836, TN=0.999

(d) Session2 flushed output, 500 epochs on All Session 1 images, with Image processing labels. TPR=0.79, TN=0.99

(e) Session2 living output, 500 epochs on All Session 1 images, with CSV labels. Retrained 50 epochs on Image processing labels. TPR=0.950, TN=0.998

(f) Session2 flushed output, 500 epochs on All Session 1 images, with CSV labels. Retrained 50 epochs on Image processing labels. TPR=0.951, TN=0.997

Fig. 3: Outputs examples

References

[1] Milesial and other contributors. *UNet: semantic segmentation with PyTorch*. 2020. URL: https://github.com/milesial/Pytorch-UNet.

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