ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SEGMENTATION OF THE LUNGS IN MICE CT SCANS BY DEEP LEARNING

BACHELOR PROJECT REPORT

Quentin Chappuis

4th January 2024

Contents

CHAPTER 1

INTRODUCTION

Nowadays, the use of Machine Learning is more important than ever in the field of medical imaging. It is being used increasingly in the analysis of medical images, enabling better diagnoses to be developed more quickly and with greater reliability. A lot of research is being done to develop and train models for medical purposes and one of the main problems is the size of the data and the resources needed to train such models.

In this project we are looking to develop an effective deep-learning based image processing pipeline that includes the automated detection and segmentation of the lungs in mice CT scans. This pipeline would help researchers studying the lungs region in X-ray computed tomography (CT scan) of mice, for the purpose of studying lung cancer treatments. To achieve this we aim to train a model based on three-dimensional images annotated by different classical algorithms.

This project is part of a larger project to find a treatment for lung cancer. Researchers have trained a model that detects tumours in CT scans of mouse lungs, but face certain problems. Sometimes tumours are detected outside the lungs, because the algorithm mistakes certain light-coloured tissues for tumours. To solve this problem, it would be convenient to have a clear segmentation of the lungs in three dimensions, so that tumours detected outside the lungs could be eliminated directly, saving researchers time. The benefit would be the ability to cut the scans directly around the lungs before passing them to the model predicting the tumours. It may also be easier to analyse the three-dimensional images without the noise around the lungs.

As the whole project is based on CT-scans, it is important to introduce this medical imaging technique. According to the Centre Hospitalier Universitaire Vaudois (CHUV) [CHU] a CT-scanner is: "[Translated from French] An X-ray machine coupled to a computer. It provides sliced images of different parts of the body and allows tissues of different densities, such as lungs, bones, soft tissues and blood vessels, to be visualised. It is used to diagnose cancers, trauma, and cardiovascular, infectious and osteoarticular pathologies."

It works in the following way: "[Translated from French] The principle of CT is based on the absorption of radiation by different parts of the body according to their composition. In a simple X-ray, the radiation passes through the patient and is then measured by a flat digital sensor to obtain a two-dimensional image of the inside of the body. In the case of CT, this sensor is replaced by several rows of detectors located in front of the X-ray tube. These measure the radiation passing through the patient as the tube and detectors rotate. At each turn, the detectors record thousands of pieces of information. From this digital information, a computer will reconstruct a transverse (i.e. sliced) image of the organ being examined. Using the same information, it is also possible to obtain images in three dimensions or in other planes of space."

In our case, the scans are performed on mice and give back a three-dimensional image of variable size. Figure 1.1 shows an example of the lungs of a mouse in a CT scan viewed in a three-dimensional viewer.

To visualize, annotate and analyze several images at a time, we used the Napari viewer [Nap]. This is a Python library that opens an interactive viewer for n-dimensional images. This tool has been used throughout the project, and we will be mentioning it again in various subsections. To learn more about Napari, I recommend exploring the workshop given by Mallory Wittwer from the EPFL Center for Imaging : `https://gitlab.epfl.ch/center-for-imaging/eias-2023-visualization-workshop`.

Before trying a deep learning solution, various conventional image processing techniques and algorithms were tried out. An attempt was made with a convex hull algorithm (Figure 1.2) applied in two dimensions to all z-plane slices. Another was carried out using Ilastik [Ila] by, first, manually colouring the lungs in some images, then applying the same process to the given images, before applying convex hull to keep only a clear segmentation of the lungs. The primary issue encountered with these two techniques was their occasional erratic behavior. Sometimes, the segmentation mask produced was the entire image, while in other cases, it resulted in the deletion of only one of the two lungs. We provide some unsuccessful samples in the following figures: Figure 1.3, Figure 3.1 (a) and Figure 3.3 (a). Also, when some tumours were on the edges of the lungs, they were sometimes forgotten and separated from the lungs. Finally, these two classic techniques are relatively time-consuming, which makes them unpractical to use.

We believe that deep learning can save time and improve accuracy, as well as making it easier to adapt to irregular shapes, structures within the lungs and different types of scan. Moreover, when new data is acquired, the model can be re-trained in order to improve it.

FIGURE 1.1
Example of a CT-scan viewed in a three-dimensional viewer

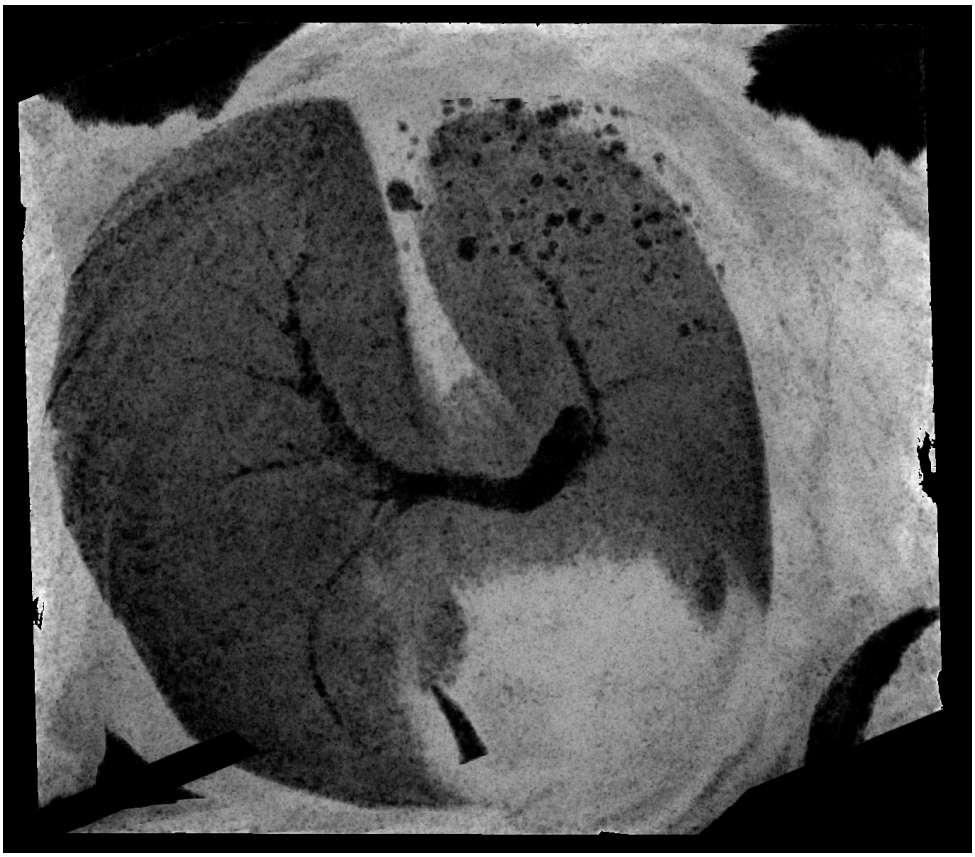


FIGURE 1.2
Example of an usage of a convex hull algorithm

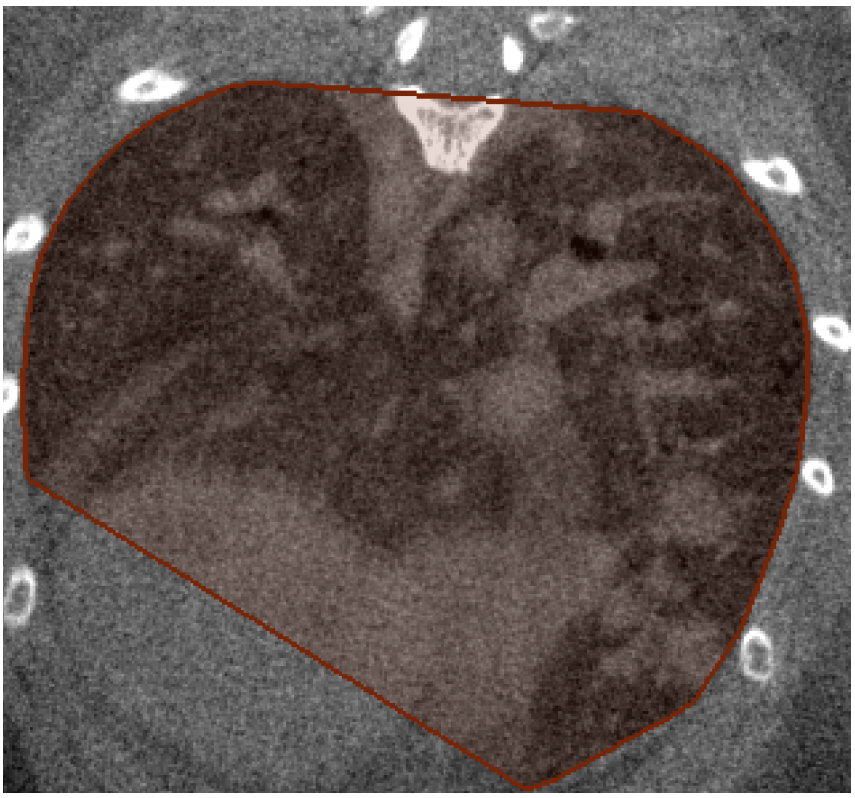
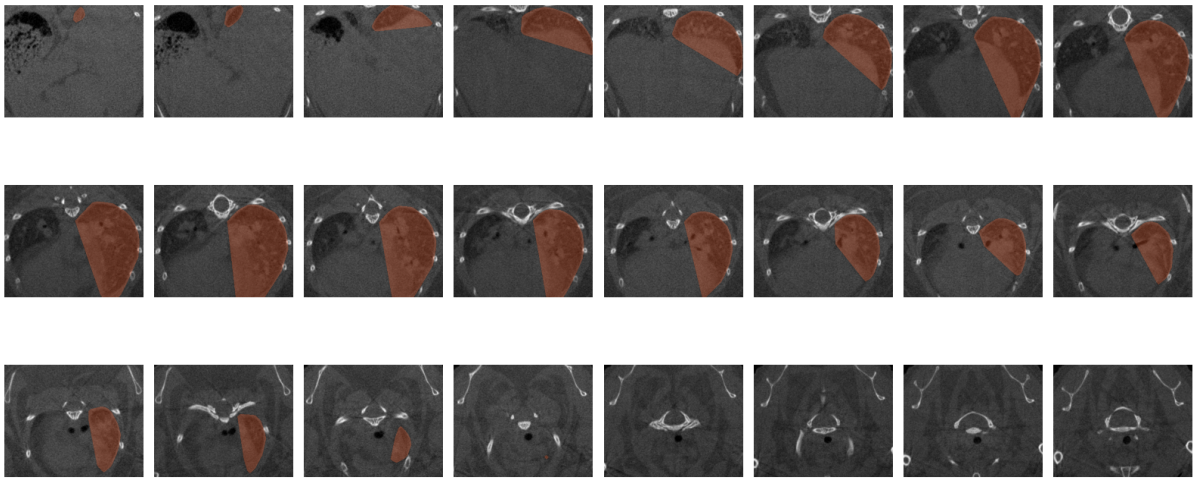


FIGURE 1.3
Example of fails of the classical algorithms



CHAPTER 2

METHODS

2.1 DATASET AND GROUND TRUTH GENERATION

To generate the dataset we used the two classical algorithms mentioned in the introduction. First we ran all the images through the second technique using Ilastik. Then we analysed by hand the resulting segmentations in Napari, keeping only for our training and validation set the segmentations that were consistent, particularly for the main part of the lungs (where the lungs are largest when viewing the image through the z-plane). All this was then transferred to a .csv file listing and keeping track of all the images made available by Professor de Palma's laboratory (500 images). Next, all the images that had not been validated after the first algorithm were run through the convex hull algorithm. We then analysed the resulting images in the same way and transferred them to the .csv file. The images that were not effectively segmented by either of the two algorithms were retained for testing the final model's performance in order to determine whether improvements have been achieved compared to the classical algorithms.

For the training, all the images were loaded into a custom class called `CustomImageDataset`. We then applied some pre-processing, starting by resizing the images to a size of 128x128x128, then rescaling them to values between 0 and 1. Before passing the training images into the model we also augmented the data by randomly rotating them by an angle between -20 and 20 degrees.

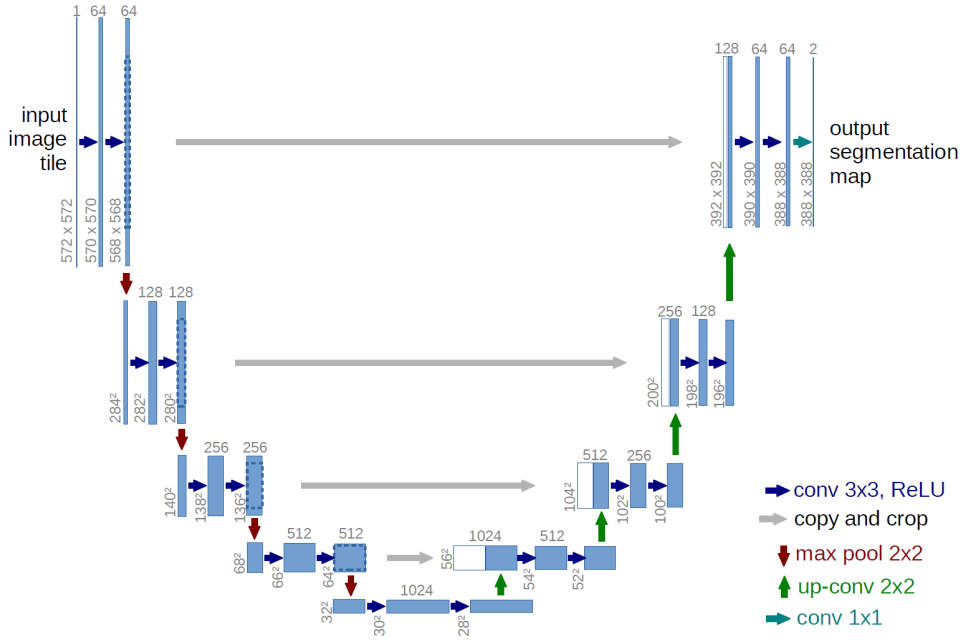
For this project, we had a total of 355 images for the training set and 62 images, chosen at random, for the validation set. In addition, there were 83 images that could not be used and were therefore part of the test set. Furthermore, these images were taken from two different CT scans, making the trained model more versatile.

2.2 MODEL ARCHITECTURE

For this project, we chose the U-Net model architecture [Fre], developed by researchers from the University of Freiburg. This convolutional neural network (CNN) is designed for image segmentation and is particularly used in the field of medical imaging. The network (Figure 2.1) is U-shaped and consists of two parts: the contracting path (encoder) and the expansive path (decoder), which are connected by a bottleneck. The encoder part consists of a block of 2 convolutions followed by a max pool. This is repeated four times before performing the bottleneck part, consisting of two convolutions. The decoder part is composed of a transposed convolution concatenated with the result of the convolutions of the encoder part. It is followed by 2 convolutions and applied 4 times. We then have an output 2x2 convolution followed by a Sigmoid activation function in order to get back to the original image shape. You can find our PyTorch implementation in the file `src/UNet_lungs_segmentation/model.py` in the

GitLab repository.

FIGURE 2.1
U-Net Architecture



2.3 MODEL TRAINING

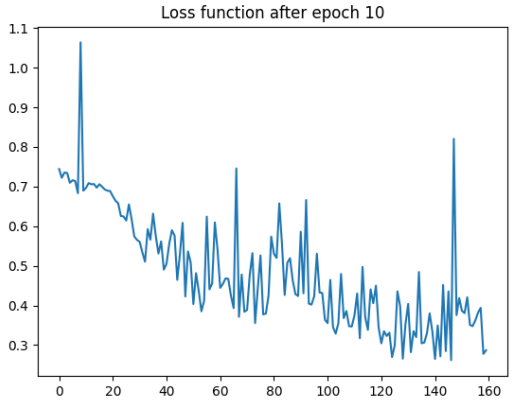
For this project we used the PyTorch Python library and the Adam optimizer, which is is an extension to stochastic gradient descent commonly used for deep learning applications in computer vision. We also used Binary Cross Entropy Loss, which is fairly standardised to use with the U-Net architecture and is implemented in the PyTorch library as follows:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

with N the batch size, x_n the input, y_n the target and w_n the weight [PyT].

Initially, we trained the model on less data and on a smaller version of the model: the number of bottleneck channels having been reduced from 1024 (as shown in Figure 2.1) to 128 or 256. To do this, we trained for ten epochs and checked whether the loss function of the training set converged. Then we ran an image of the test set in the model and looked at the output on Napari. This allows us to see if there were any consistency in what was coming out of the model and therefore if we had chosen the right model, the right optimizer and the right loss function. As you can see in Figure 2.2, the training set loss of one of our first training converged, which was a good sign for a start.

FIGURE 2.2
First training over 10 epochs and about 16 images



As the results were encouraging, we started to train a larger model for longer and on more data in order to find the right hyperparameters. Very quickly, we noticed that training the model took up a lot of memory on the GPU and we were therefore forced to keep the batch size very low. Due to this, the training was very long and it was one of the crucial points of the project: finding the right hyperparameters configuration without being able to run the training over and over again.

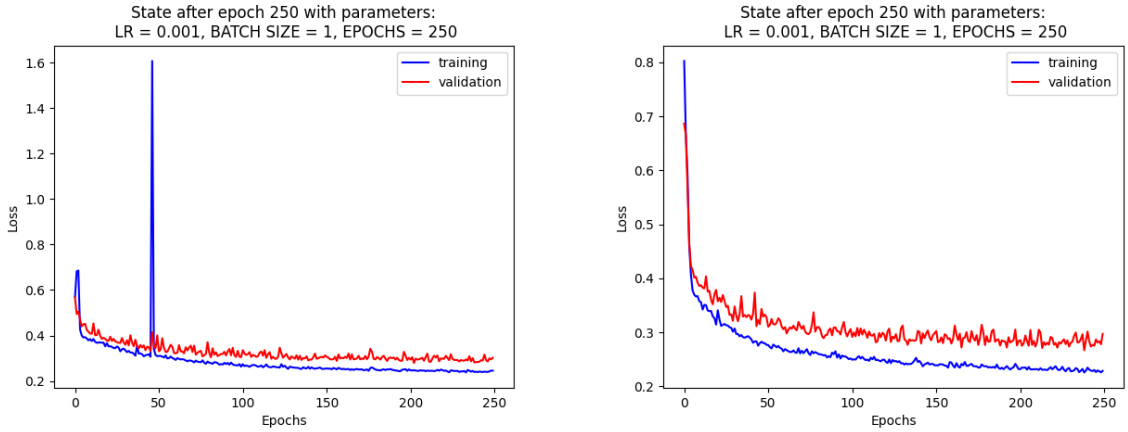
Furthermore, we trained on 76 images in order to compare the model with a bottleneck of 1024 channels and a batch size of 1 (maximum without exceeding the GPU memory) and the model with a bottleneck of 512 and a batch size of 2, to see whether or not the time saving, by using a smaller model, affected the accuracy of the model. With training over 250 epochs lasting 11 hours and 26 minutes for the 1024 bottleneck and 7 hours and 11 minutes for the 512 bottleneck, we saw that the losses (Figure 2.3) were similar and converged in the same way. However, despite this significant time saving, after analysing the two models on different images of the test set we saw that the segmentations was more accurate with a bottleneck of 1024 channels. Therefore, we decided to keep the original model from the University of Freiburg and its 1024 channel bottleneck for the future and final trainings. Nevertheless, it is interesting to use smaller models for testing some parameters (such as the learning rate), in order to save resources.

By dividing the learning rate by 10 (Figure 2.4), we could see that losses were converging faster (around epoch 50) and lower, before starting to overfit. This is why, by stopping the training before it overfits, it was possible to obtain better results more quickly. As we were near the end of the project, we trained our final model on all the available data, maintaining the learning rate mentioned earlier. We ran 100 epochs of training (Figure 2.5) and we observed that the losses converged between the 40th and 50th epochs. Hence, we were able to obtain the best weights after implementing a few lines of code that automatically save the weights if the loss validation is at the lowest ever captured.

To finish, we tried different training with the initial learning rate (0.001), but the validation loss was not going down at all. We tried adding weight decay, but it was not helping.

As there was not much time left before the end of the project, we decided to keep the training on the 100 epochs, mentioned above (Figure 2.5), with a learning rate of 0.0001 as the final result. In fact, after testing this model on different test set images, it gave us promising results, which we will discuss in the next chapter. The final training lasted 21 hours and 31 minutes.

FIGURE 2.3
Comparison of the training between a 512 and a 1024 channels bottleneck



(A) Loss functions for the 512 bottleneck model

(B) Loss functions for the 1024 bottleneck model

FIGURE 2.4
Training over 250 epochs with a lower learning rate

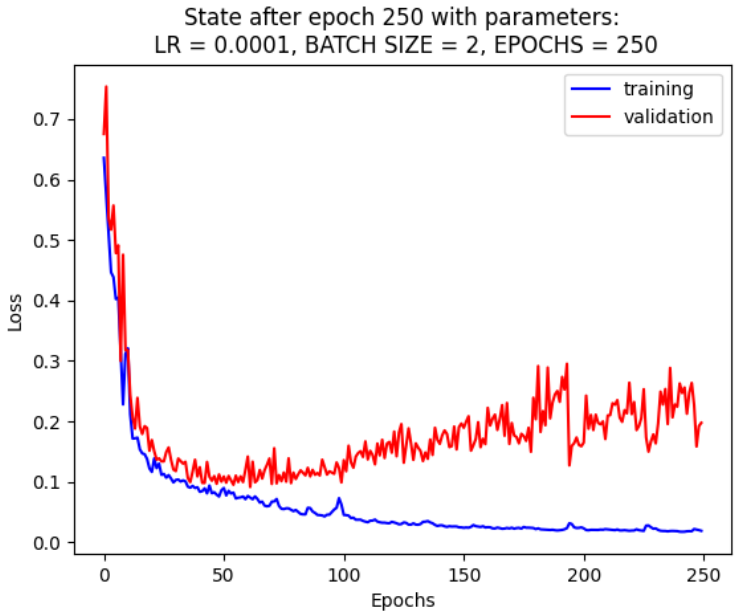
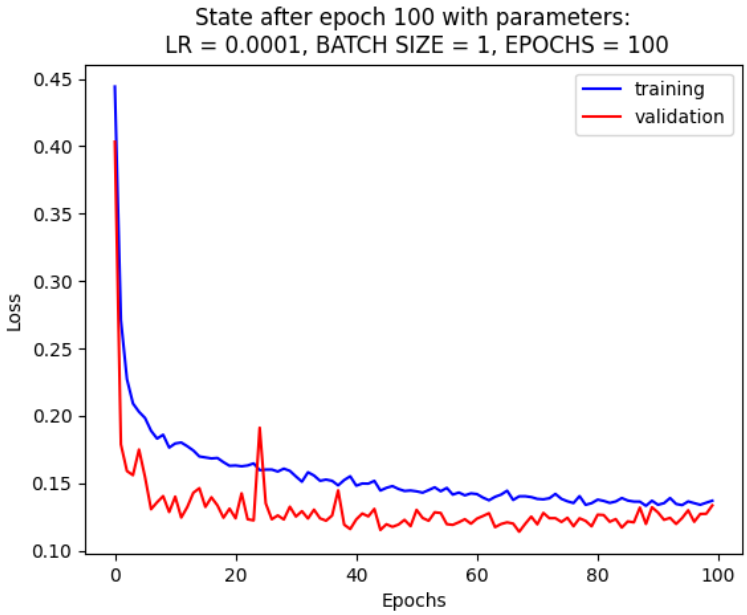


FIGURE 2.5
Final training of the model



2.4 POST-PROCESSING

The only post-processing we applied to the images was a thresholding of 0.5 to convert the segmentation into a binary mask. There was no need for further post-processing as the model already rendered an accurate single-block segmentation.

CHAPTER 3

RESULTS AND DISCUSSION

3.1 SUCCESS OF THE MODEL

All the results presented here are available in a notebook named `results.ipynb` placed in the GitLab repository. The results are more meaningful from the notebook, as it is possible to view the results in three dimensions using the Napari viewer.

In this section we will analyse three examples (taken from the test set) which demonstrate that our model drastically improves the accuracy of segmentations compared to the two classical approaches. To compare this, we generate two plots comparing the segmentation generated with the first technique (using Ilastik), with the one of our model.

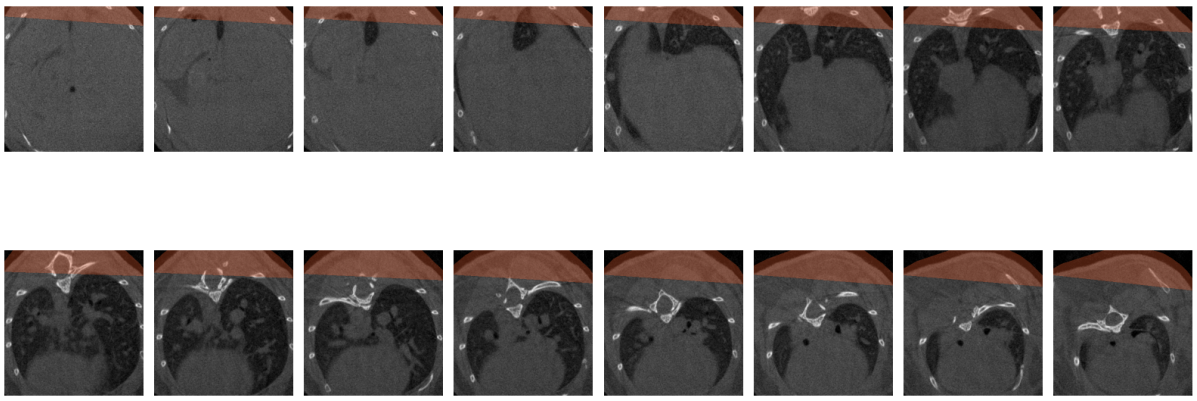
First, Figure 3.1 shows that Ilastik's segmentation, which lasted 37 seconds, is not working at all. On the contrary, our model produced a very good segmentation, however a little inaccurate at the extremities, in around 2 seconds.

Our second example (Figure 3.2) shows a classical segmentation which took 34 seconds omitting one of the two lungs. Our model again shows a very good segmentation, with, again, a few problems at the extremities in a time of 2 seconds.

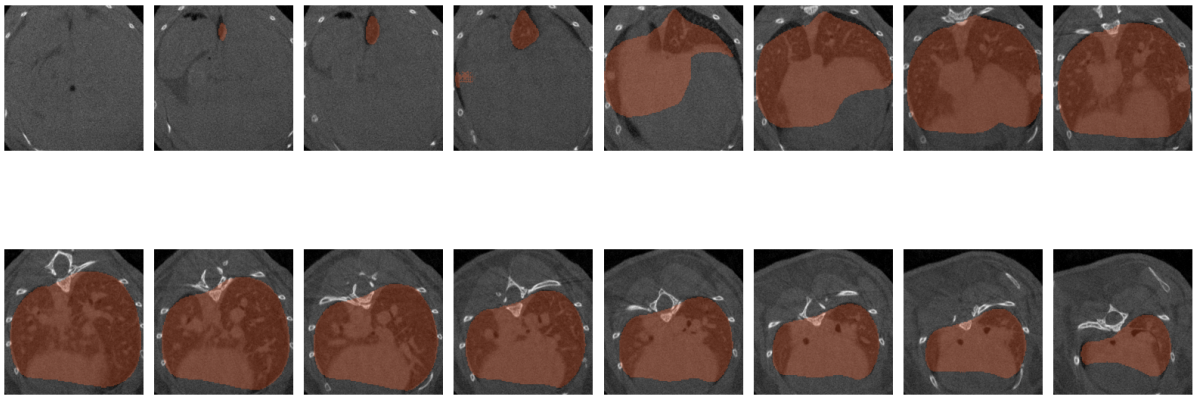
The third example (Figure 3.3) shows that the classical algorithms (segmentation carried out in 42 seconds) does not understand at all where the lungs are located and confuse the mouse's entire body with the lungs. Our model's segmentation, again, generated in 2 seconds, is successful and shows very few errors, apart from a few inaccuracies at the edges towards the end of the image.

We can therefore see a huge improvement in the speed of the algorithm, as well as greater reliability in finding the lungs and segmenting them correctly in our model. However, we will analyse some of the limitations of our model, mentioned briefly here, in the next chapter.

FIGURE 3.1
First comparison of the results between the ilastik technique and our model

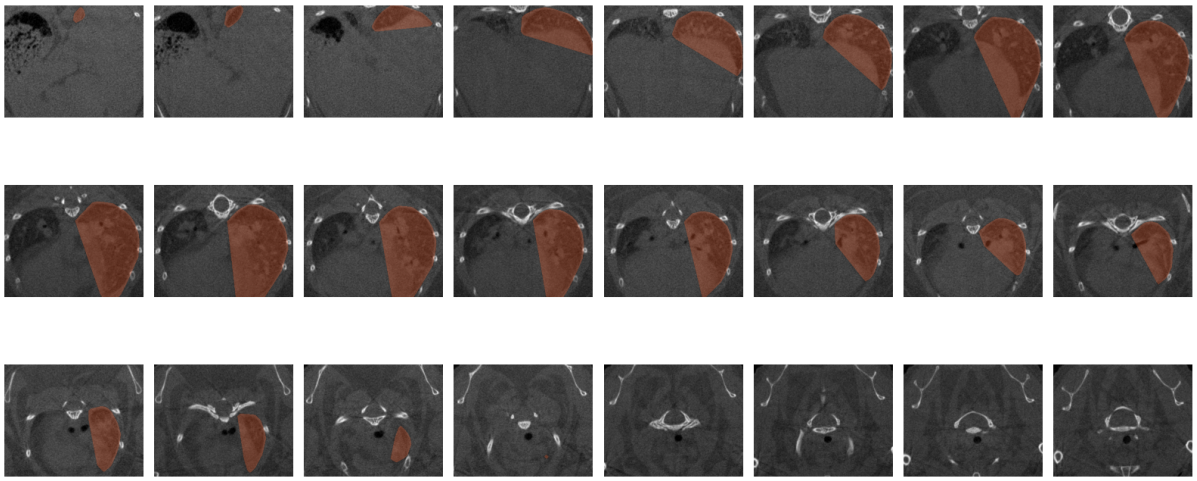


(A) Segmentation after the Ilastik technique

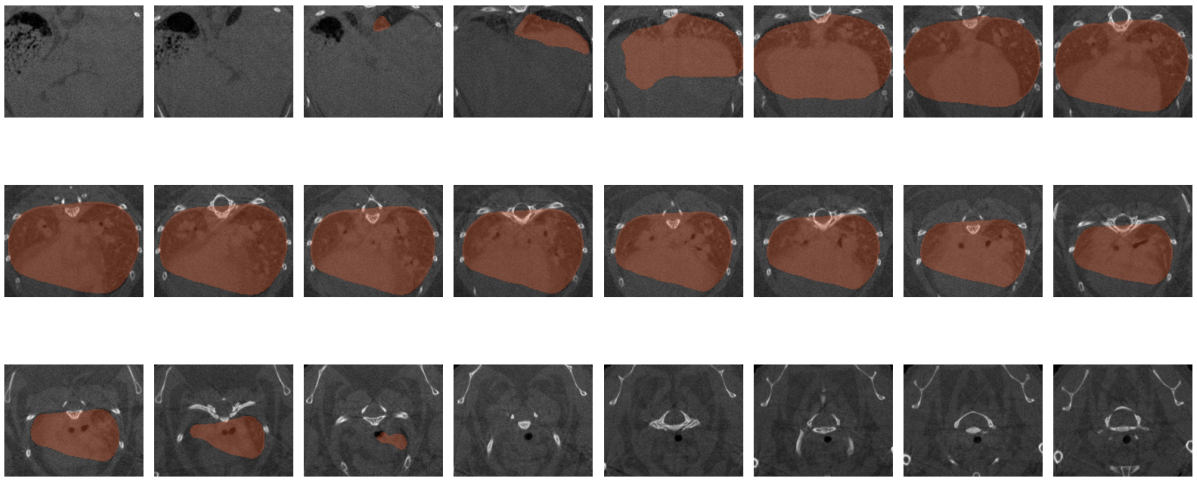


(B) Segmentation after passing through our model

FIGURE 3.2
Second comparison of the results between the ilastik technique and our model

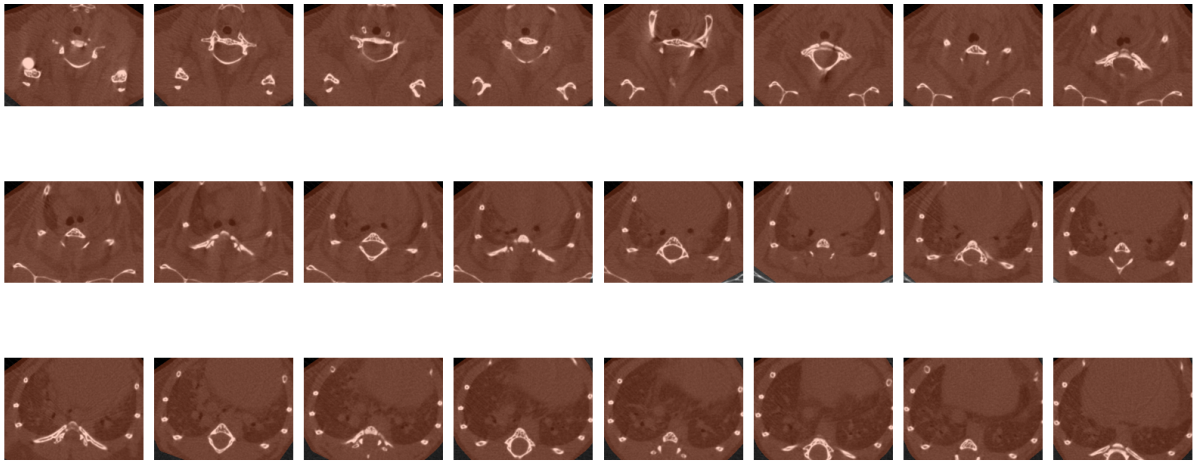


(A) Segmentation after the Ilastik technique

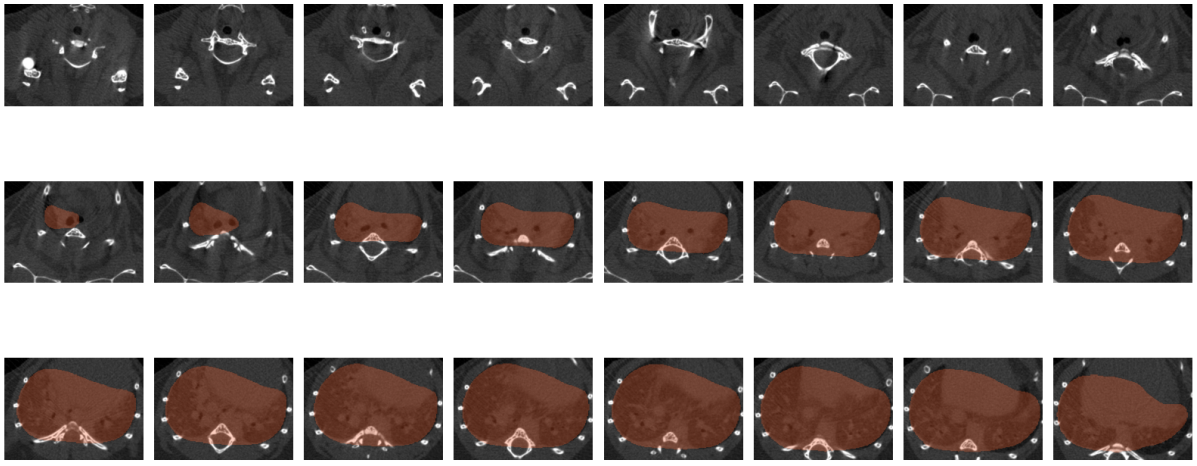


(B) Segmentation after passing through our model

FIGURE 3.3
Third comparison of the results between the ilastik technique and our model



(A) Segmentation after the Ilastik technique



(B) Segmentation after passing through our model

3.2 LIMITATIONS OF THE MODEL

In this section we will look at the limitations of the model and discuss possible improvements. All images are again taken from the test set.

Firstly, the segmentation produced by the model is sometimes a little imprecise at the edges. This can be seen in Figure 3.1, on the first row of images. This is also linked to the fact that the extremes (the 2D images at the beginning and end of the z plane) are poorly segmented. The reasons for this are inaccuracies in the training set: the training set images were generated using conventional algorithms and have some inaccuracies at the extremes. This is not a major problem since the crucial part of the lungs is the central one. One way of dealing with this would be to annotate the data set by hand, but this would be extremely time-consuming.

Secondly, another problem can be seen in Figure 3.4. This is the fact that in images where there is a black background all around the mouse body, the model segments a little too widely and imprecisely in some places. Nonetheless, this is not too bad, as it is better to have a too wide segmentation than losing some part of the lungs. The reason for this issue is that there were no images of this type in the training set, as the classical algorithms used to generate the training set were absolutely unable to locate the lungs and kept segmenting the whole body.

Figure 3.5 shows that segmentation did not work at all. In fact, the model doesn't seem to understand where the lungs are located and this is once again due to the fact that the training set doesn't include any usable images where the lungs are on the edges.

The solution to the two problems mentioned above would be a training set with more images covering the edge cases. This would certainly require some images to be annotated by hand.

Another limitation is the use, in pre-processing, of the `rescale_intensity()` method of the skimage library [Sci]. As all the values are scaled between 0 and 1, this could cause problems if, for example, an image passed to the model has valid values everywhere except for a certain voxel, which has a very high value. The valid values will therefore become very small and will not be well understood by the model. There is no solution to this, as we have trained the model on images that were pre-processed using this method, but it is worth mentioning.

FIGURE 3.4
First case of limitation of our model

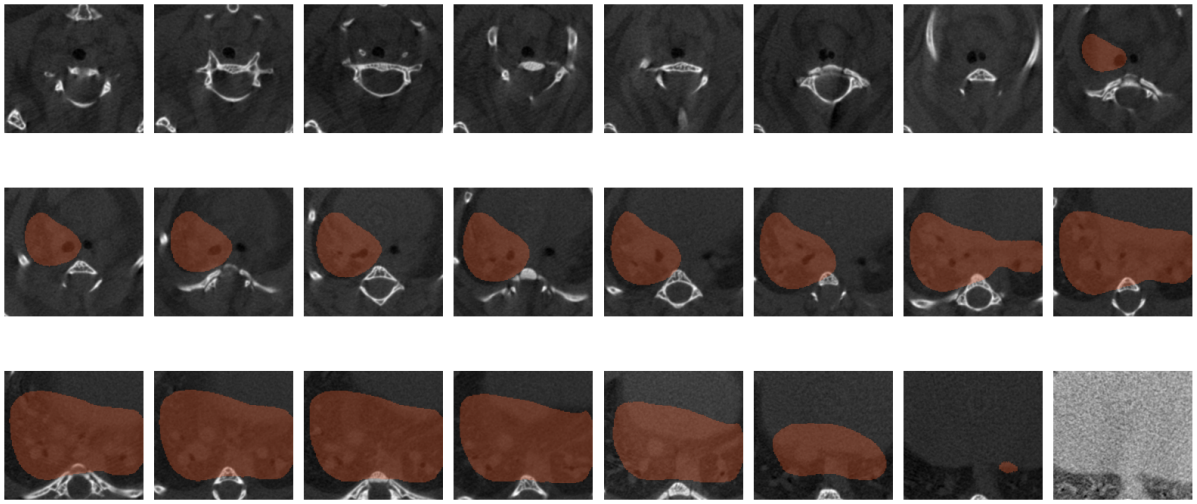
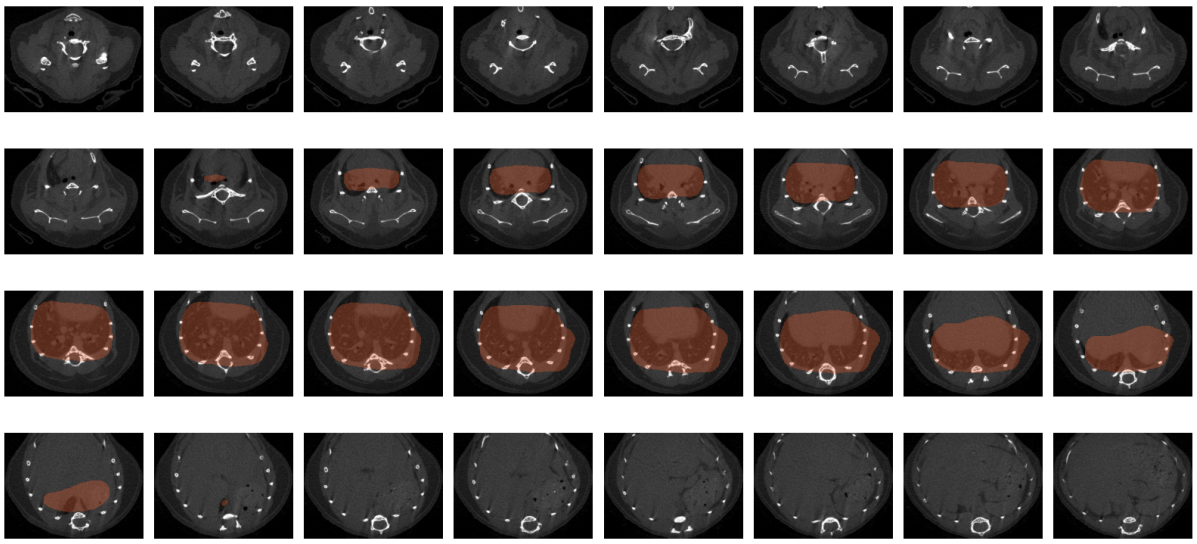


FIGURE 3.5
Second case of limitation of our model



CHAPTER 4

CONCLUSION

In this project, we have trained a model for segmenting the lungs in mice CT scans, with the aim of improving the solutions already in place using classical algorithms. To do this, we trained a U-Net model on three-dimensional images.

The project was divided into 3 main parts:

- Preparation of the model and data set
- Training of the model and hyperparameters optimization
- Analysis of the results on the test set

To generate the data set we used the two classical algorithms already in place, so that we didn't have to annotate the data ourselves. We had to deal with some problems, such as a lack of data for certain types of scan and the malfunctioning of our two algorithms.

During the training phase, we had to deal with overfitting and optimise the parameters to obtain the most accurate results while keeping a reasonable training time.

In the end, we obtained very good results for our model, as well as significant time savings. The time taken to generate the segmentation of an image is on average 2 seconds. Furthermore, our model covers most of the recurring errors obtained with the two conventional solutions.

However, certain aspects could still be improved and our model has a few limitations. For example, the model's lack of precision in the extremes of the lungs, due to inaccurate training data. Or the slightly broad prediction of the model on certain types of images, due to the total absence of such images in the training set.

In conclusion, the model shows better results than the previous methods and can assist researchers in saving time on certain tasks.

ACKNOLEDGEMENTS

We wish to thank Mallory Wittwer and Edward Andò for supervising this project and for all the interesting discussions we had together!

CHAPTER 5

APPENDIX

All the code and corresponding documentation can be found in the following repository : `https://gitlab.epfl.ch/center-for-imaging/tumor-lungs`.

We have also developed a napari plugin. To test it, you can run it on our provided sample image (in Napari, open the image from `File > Open Sample > Mouse lung CT scan`) or drag-and-drop an image into the viewer window. Next, in the menu bar select `Plugins > Lungs segmentation (UNet_lungs_segmentation)` and then select an image and run it by pressing the "Segment lungs" button.

It is also possible to use it as a Command Line Interface (CLI). Run inference on an image from the command-line:
`uls_predict_image -i /path/to/folder/image_001.tif [-t <threshold>]`

The `<threshold>` will be applied to the predicted image in order to have a binary mask. A default threshold of 0.5 will be applied if none is given. Should be a `float` between 0 and 1.

Run inference in batch on all images in a folder:
`uls_predict_folder -i /path/to/folder/ [-t <threshold>]`

You will find more information about these implementations by reading the `README.md` file!

BIBLIOGRAPHY

[CHU] CHUV. *CT Scanner - Computed Tomography*. URL: https://www.chuv.ch/fr/rad/rad-home/patients-et-familles/nos-examens/ct-scanner.

[Fre] University of Freiburg. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. URL: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/.

[Ila] Ilastik. *ilastik: the interactive learning and segmentation toolkit*. URL: https://www.ilastik.org/.

[Nap] Napari. *napari: a fast, interactive viewer for multi-dimensional images in Python*. URL: https://napari.org/stable/.

[PyT] PyTorch. *BCELOSS*. URL: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html.

[Sci] Scikit-Image. *skimage.exposure.rescale_intensity(image, in_range="image", out_range="dtype")*. URL: https://scikit-image.org/docs/stable/api/skimage.exposure.html#skimage.exposure.rescale_intensity.