UCLouvain

École polytechnique de Louvain

Convolutional neural network for blood vessel segmentation : 2D and 3D architectures

Author: **Mathilde Henry de Frahan**
Supervisors: **Christophe De Vleeschouwer, Benoît Macq**
Reader: **Laurent Jacques**
Academic year 2020–2021
Master [120] in Mathematical Engineering

Abstract

Recently, thanks to the success of deep learning models in image processing and computer vision, many image segmentation techniques using deep learning have emerged. Some of these models have a huge potential for medical imaging. In particular, convolutional neural networks are very efficient for the segmentation of blood vessels allowing to detect vascular diseases.

In this work we propose convolutional neural networks based on the well-known architecture U-Net which are intended to segment blood vessels in 3D volumes. Among the great challenges in the processing of 3D datasets by full 3D convolutional networks, we find the slow execution time and the high memory requirements. To try to remedy these problems, a method modifying the convolution operations performed in convolutional neural networks by using 2D orthogonal cross-hair filters is presented. This method, which depending on datasets and network configurations, can save time, memory or improve the accuracy of the predictions or at least obtained results very similar to those obtained with classic convolutions, while using 3D context information.

Acknowledgments

This work is the result of my university studies. These five years have allowed me to train on many levels. I would like to thank all the professors I have had for giving me so much.

This master thesis would not have been possible without the concrete help of dedicated people to whom I am extremely grateful.
Eliott Brion who taught me to understand some notions of deep learning and followed a part of my work.
Victor Joos de ter Beerst has been with me throughout the year. He did not hesitate to dedicate me his time, to explain me the use of GPUs on the Cassiopee server. His proofreading of the thesis and his many advice were decisive. I cannot thank him enough for his availability for my numerous questions.
I also thank the reader, Laurent Jacques, for the attention he pay to this work.

Finally, I express all my gratitude to my promoters, Christophe De Vleeschouwer as well as Benoît Macq. In complicated times, their encouragement were more than necessary. They devoted their time and provided crucial support in every step of this work.

I wish all students to know such a chance for support.

Contents

Abbreviations

Adam Adaptive moment estimation

CNN Convolutional neural network

CT Computed tomography

DVN DeepVesselNet-UNet

FCN Fully convolutional network

GPU Graphics processing unit

MLE Maximum likelihood estimation

ReLU Rectified linear unit

SGD Stochastic gradient descent

Introduction

Blood vessel segmentation is of crucial interest for medical imaging because it allows a better handling of vascular diseases. It is used in many applications. However, these applications require qualified prediction methods allowing the segmentation of vessels of various sizes, but also the detection of anomalies in the vessels in order to diagnose possible diseases. Segmentation can be done manually, but it is a cumbersome, complex and time-consuming process, especially when analyzing large and complex datasets. Therefore, automation of this process is important to save time and resources.

During the recent years, segmentation on the basis of a dataset is possible thanks to deep learning techniques. Deep learning is a subset of machine learning, which is itself a subset of artificial intelligence. Deep learning algorithms aim to draw the same conclusions as humans by analyzing data with a given logical structure thanks to neural networks, which are hierarchical structures with a series of layers.
Before deep learning, machine learning methods were used such as decision tree, support vector machine, logistic regression or K-nearest neighbors [1] but they cannot directly use raw data such as images, texts, and so on. A pre-processing step called feature extraction is required which is quite complex.
A major advantage of deep learning over machine learning is that feature extraction is not anymore necessary. Indeed, the features are automatically learned by the network itself by training to adapt to a set of data thanks to the layers present in the neural networks.
Another huge advantage of deep learning over machine learning is that it can be powered by large amounts of data. Indeed, traditional machine learning models saturate at a time and therefore no longer improve while with the increasing amount of data deep learning models improve their accuracy.
However, a major drawback of these neural networks is that to learn models correctly a large amount of training data is required as well as a significant computing power.

Although many automated methods have emerged due to the increasing demand for competent vessel segmentation techniques, they often face many challenges. Among these challenges we find the prediction accuracy due to the complexity of medical images with in particular the presence of artifacts such as noise, intensity inhomogeneity or motion artifacts. This complexity leads to false detections, i.e. elements that are not vessels are detected as being vessels, and missed detections, i.e. vessels that are not detected. In addition, processing large 3D data leads to memory consumption and speed challenges.

In the biomedical field, convolutional neural networks (CNNs) are among the most efficient deep learning image analysis models. CNNs are a class of deep neural networks, and as the name suggests use convolutional operations.

This master thesis mainly presents a variant of convolutional neural networks for the segmentation of blood vessels. The particularity of this variant lies in the convolution operation and aims at saving time and memory compared to the classical networks. This variant is implemented on the well-known convolutional neural network U-Net.
Moreover, a comparative analysis is presented between U-Net and this variant both in 2D and in 3D for two distinct 3D datasets, one synthetic and one real.
The objective of this work is therefore to establish the efficiency of the CNN variant implemented on U-Net as well as to determine which type of architectures (2D or 3D and U-Net or the variant) outperforms the others in terms of prediction accuracy and which one allows gains in speed and memory in order to find the best compromise between the prediction quality, use of memory and speed.

This report is structured as follows: chapter 1 presents the basics of neural networks and some of its key concepts. The functioning of convolutional networks is then explained in depth as well as the models more specifically used for blood vessel segmentation. In chapter 2, the variant of CNN is described. In this same chapter, the different datasets, the particularities of the architectures and their parameters used for the experiments are also presented. Finally, the last chapter (chapter 3) contains the results of the experiments carried out as well as a discussion of the results and the limitations of these experiments.

Chapter 1

Background

1.1 Deep learning for image segmentation

Image segmentation consists of the separation of images into several segments or objects. It occurs in many fields. For example, for autonomous vehicles such as the separation of roadways and pedestrians, for the analysis of medical images including the retrieval of tumor boundaries or the measurement of tissue volumes, for augmented reality and many other applications.
Different levels of segmentation are possible: semantic segmentation and instance segmentation. Unlike image classification where the whole image has only one label prediction, semantic segmentation, which is more complex, occurs at the pixel level. Indeed, it consists of the creation of a labeling with different categories (vessels, tumors, ...) for each pixel of the image. Instance segmentation goes beyond semantic segmentation by allowing for each pixel the identification of the object instance it belongs to. The main difference with semantic segmentation is that it differentiates objects with the same labels.
This master thesis will only deal with semantic segmentation.

Many algorithms allowing image segmentation have emerged, starting with methods such as thresholding, histogram-based bundling, region-growing, k-means clustering and watersheds. Subsequently, more complex algorithms were developed like active contours, graph cuts, conditional and Markov random fields and many others. [2]
In recent years, a new generation of segmentation models have emerged through deep learning models. These models have greatly improved segmentation performance. [3]

1.1.1 Deep neural networks overview

The use of deep neural networks to remedy many computer vision problems recently became the main tool.
To understand what is a deep neural networks it is useful to introduce briefly the notion of perceptron, an artificial neuron. With several binary inputs (x_1, x_2,...) a perceptron obtains a binary output. To the entries, weights (w_1, w_2,...) are added to represent the importance of the different inputs and a bias b is also introduced. More precisely perceptron is equivalent to the sum of all its inputs and their corresponding weights, to which is added the bias. To the result obtained is applied an activation function, i.e. a nonlinear function. This is illustrated on the figure 1.1 where both weights and bias are parameters of the artificial neuron.

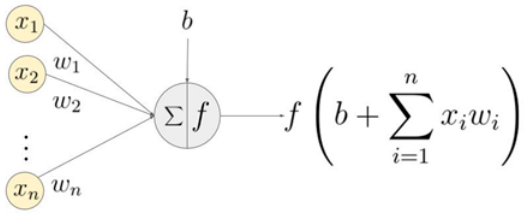


Figure 1.1: Example of a perceptron with the input (x_1,...,x_n), the corresponding weights (w_1,...,w_n), a bias (b) and the activation function (f). [4]

A neural network is simply a configuration of perceptrons organized in layers as it can be seen on the left of the figure 1.2.
The goal is to find an algorithm allowing to determine the weights and biases in order to have an output from the network which is as close as possible to the real output. To do this, a training sample is propagated through the network and the network returns a result. This output estimation process is known as forward propagation. Then the output error based on the gap between the target and the predicted values is calculated, i.e. the loss or cost. Subsequently this error is propagated through the network to allow efficient updating of weights and biases during training in the hope of reaching a minimum, namely the backpropagation algorithm. This update is done using an optimization algorithm that aims to minimize the loss.

The "deep" in deep learning means the depth of layers in a neural network. A neural network with many (two or more) hidden layers, i.e. the layers located between the input and the output, is called a deep learning network. Deep learning networks can be used to train complex data and predict the output. Such networks can be seen on the right of the figure 1.2.

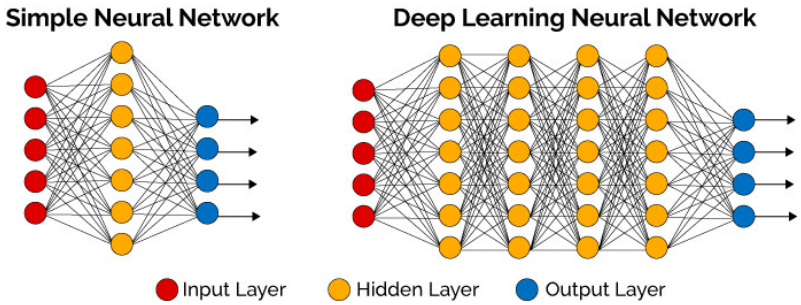


Figure 1.2: Illustration of layers in a neural network on the left and in a deep neural network on the right. [5]

1.1.2 Some important concepts for neural networks

Now that the basis of deep neural networks is established, it is important to mention some of their most important concepts.

Cross-validation

Networks can be made up of two main phases: the training phase and the testing phase. Dividing the dataset into two parts, one for training and the other for testing, is called the cross-validation technique. This technique makes it possible to evaluate the performance of the models obtained and also allows to compare and select the most appropriate model to a specific problem. There are many techniques for cross-validating a model but all are based on the same principle: separate the dataset with one part for training and the other for testing, train the model on the training set, validate the model on the test set and repeat these steps several times. If there is no repetition it consists in using the hold-out cross-validation. Hold-out has major disadvantages. When a dataset is not equivalent in terms of distribution, the training set may not represent the test set and the fact that the model is only tested once may lead to inaccurate results.
K-fold cross-validation is a technique that minimizes the disadvantages of hold-out method. This techniques consists to randomly divide the dataset into K parts, or folds, of approximately equal size. $K-1$ folds are used for the training and the remaining fold is treated as the test set. These steps are repeated K times. [6]

During the training phase many decisions about which parameters to use in the neural networks must be made. Determining these elements is an important step because each one has a significant impact on the network performance. The main parameters are listed below.
For the test phase, new inputs that are not used during the training, pass through the network to verify how accurate the network is with data that it does not know.

Optimization algorithms

Optimization algorithms are used to reduce the losses and to have the most accurate results possible by adapting the attributes of the neural network especially the weights.
The gradient descent is the most common algorithm used to train a neural network. It determines how to adjust the parameters in order to minimize its output deviation. Concretely, the parameters are, as a first step, arbitrarily chosen. The algorithm then determines the parameter variation depending on a loss function in order to have the highest decrease of the error.
Mathematically, the gradient is a vector giving the direction in which the loss function grows faster and can be expressed as

$$w = w - \alpha \frac{\partial c}{\partial w} \tag{1.1}$$

where $\frac{\partial c}{\partial w}$ is a partial derivative of cost c regarding weight w and α is the learning rate which helps to adjust the weights with respect to gradient descent. To minimize the loss function it is therefore necessary to move in the opposite direction. As long as a local minimum is not reached, the calculations are repeated.
However, depending on the initial parameters chosen, the algorithm does not always converge towards the same minimum. This represents one of the major drawbacks of the gradient descent: being stuck in a local minimum. [4]

The updated number of weights during training is called step size or learning rate. The learning rate plays an important role in the efficiency of the algorithm. In the case of the gradient descent, it can avoid to be stuck in a local minimum. In addition, it determines the speed at which the parameters are updated at each iteration by determining the step size that it takes into the direction of the local minimum.
For a high value of the learning rate, the parameters adapt faster and so the iteration number decreases. However a too high learning rate could lead to oscillations and not converge to the optimal solution. But choosing a learning rate that is too small considerably increases the number of iterations, greatly reduces speed and can lead to poorer generalizations. These situations are illustrated on the figure 1.3.
In view of this, the most appropriate choice is often to choose a variable learning rate, i.e. a rate which adapts during the training process. There are many ways to do this. One approach is to manually determine the rate as for example by reducing by half the learning rate after a certain number of epochs. Another way is to automatically find the rate as is the case in the method presented by Tom Schaul et al. [7] where the learning rate decreases or increases during the learning process. [4]

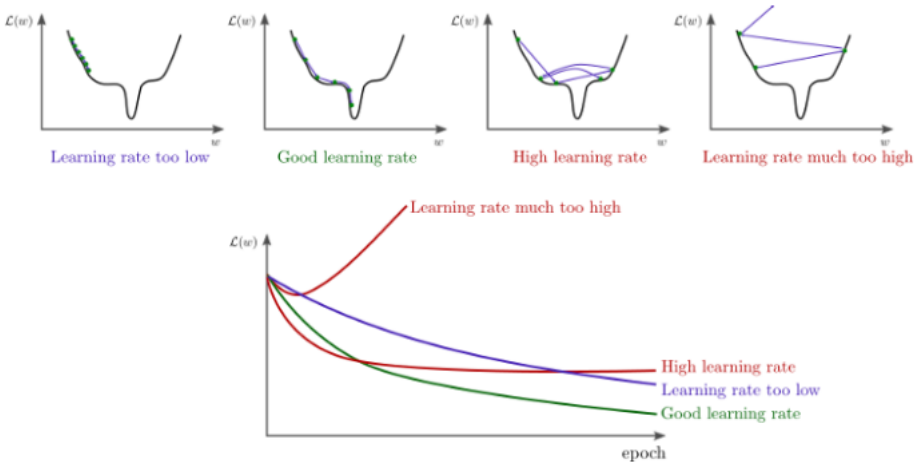


Figure 1.3: Different learning rates and their impact on the convergence of the algorithm. [4]

The batch size is another parameter that influences the optimization algorithms and which represents the number of examples from the training dataset used to evaluate the error gradient.

Loss function

When learning of neural networks, using an optimization algorithm involves choosing a loss function that calculates the model error. Many loss functions exist but its choice is important because it influences the performance of the model.
Maximum likelihood estimation (MLE) is usually used for the loss function to find the best statistical estimation for model parameters from training data. The loss function under maximum likelihood, evaluates how similar the predicted distribution of parameters are to the distribution of the ones in the training set. One of the advantages of this is its consistency property: the estimation of model parameters improves with the increase in the number of training data.
For binary or multi-class classification and segmentation problems a common used loss function is the cross-entropy loss which measures the difference between two probability distributions for a given set of events. [8] The cross-entropy loss will be seen in details in chapter 2.

Activation function

The activation function as already said before is based on the weighted sum and a bias (see figure 1.1) and determines whether a neuron should be activated or not. This function makes it possible to obtain models other than linear regressions by

applying a non-linear transformation to the output of a neuron. In this way, more complex tasks can be handled. [9]
Among the many existing activation functions, the most popular are mentioned below.
With the sigmoid activation function, any real input is converted into a small range such as between 0 and 1. A disadvantage of the sigmoid function is that it is susceptible to vanishing gradients.
Briefly, the vanishing gradient problem appears when training neural networks with gradient-based methods such as backpropagation. The issue comes from the fact that the amount of learning of the network is determined by the gradients. It happens that these gradients are very weak or even zero, the network will therefore have little or no training which can cause poor predictive performance.
The rectified linear activation function, or ReLU activation function, regularly used, returns only the input value if it is positive otherwise zero is returned and allows to better handle the problems of vanishing gradients. [10]
The softmax function transforms the input into a vector of values totaling one and which can be seen as probabilities of belonging to a class. It can be formulated as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{1.2}$$

where $\mathbf{z}$ is the input vector composed of the elements z_i which can be positive, zero or negative. e^{z_i} allows to have a strictly positive value. If the entry is negative its value will be very small. Conversely, if the input is large, it will be very high. The division by $\sum_{j=1}^{K} e^{z_j}$ is necessary to normalize and thus obtain valid probabilities, i.e. values in the range [0,1] and so that all the output values sum to 1. It thus allows to widen the gap between the highest values and the others. And finally K represents the number of classes. [11]

Data augmentation

In addition to the choices regarding network parameters, the way of pre-processing the input data plays an important role in the performance of the model. An example of data pre-processing is the data augmentation.
It happens that for some works, specially the ones with a limited dataset such as in the medical imaging field, the data augmentation is used in order to increase the number of labeled samples. More precisely, the data augmentation is applied to the input images and their corresponding labels to increase the number of training samples. To do this, a set of transformations are applied such as translation, rotation, scaling, reflection, warp, shift in the color space, and so on.
The benefits of increasing the number of training data when only a small dataset is available are improving model performance, accelerating convergence, improving

generalization and reducing risk of overfitting (training data is modeled too well) since it allows learning data not to be memorized. The data augmentation also allows to make the architecture invariant to elastic deformations, rotations, translations and so on. Compared to fine-tuning of network parameters, better results in terms of accuracy can be obtained with pre-processing and data augmentation. [3]

1.2 Convolutional neural networks

Convolutional neural networks (CNNs) are one of the most successful and popular architectures in the deep learning world, mainly in computer vision.

1.2.1 How do CNNs work?

Convolutional neural networks are made up of several layers of artificial neurons. As previously said these neurons attempt to mimic biological neurons and compute the weighted sum of several inputs and provide an activation value. The weights allow to control the behavior of their respective neuron. Depending on the values of each pixel, artificial neurons select different visual features.
CNNs consist of a stack of various layers which few types are discussed below.

- Convolutional layers : As its name suggests, a CNN is mainly made up of a series of convolutions. Convolution consists of sliding the filter on the input. During this operation, for all the elements, an element-wise matrix multiplication is performed and the results obtained are then summed. In other words, the extraction of the features is obtained following the convolution of a kernel or a filter of weights. An example of a convolution operation is illustrated on the figures 1.4 and 1.5.
  The neural network learns the values of the filters but the size of the filters and their number are manually selected. The type of selected features is influenced by the values of the filter and the number of filters determines the number of selected features.
  This layer is very important in this master thesis and will be further developed in chapter 2.

- Activation layers : An activation function is applied on the feature maps, which represent the result of applying filters to an input image, so that the network can model nonlinear functions.

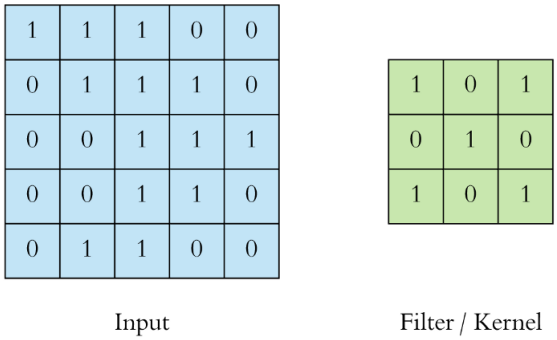


Figure 1.4: The input of the convolution layer is illustrated on the left, and on the right it is the kernel or the filter. [12]

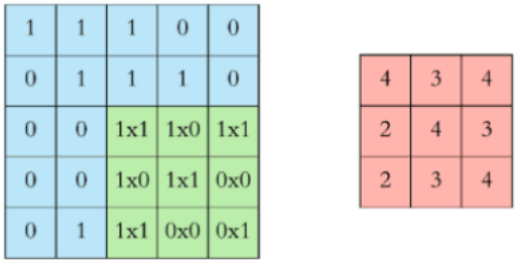


Figure 1.5: Illustration on the left of a convolution operation and on the right the corresponding feature map which is the result of the element-wise matrix multiplication of the input and the filter and the sum. [12]

- Upsampling layers : With convolutional layers the inputs are downsampled since the convolution output has fewer rows and columns as the input. But sometimes the opposite is wanted, i.e. reconstructing the original input. This is possible thanks to the upsampling layers which try to expand the inputs compressed by convolution operations by using transposed convolution in particular. The transposed convolution is reverse of the convolution but the way to learn weights is the same. For transposed convolution, the input values are multiplied successively by the kernel weights to produce the upsampled input. [13]
  A representation of a convolution and a transposed convolution can be seen on the figures 1.6a and 1.6b respectively.

(a) Convolution (b) Transposed convolution

Figure 1.6: Illustration of a convolution of a 3×3 kernel over a 4×4 input and a transposed convolution which is equivalent to a convolution of a 3×3 kernel over a 2×2 input padded (introduction of new pixels around the edges of an image) with a 2×2 border of zeros [14]

A CNN takes an image as input and some feature maps are generated at each layers. These maps make it possible to obtain the appropriate features of the input image. All neurons start from a patch of pixels, sum the products of the color values by their weights and pass them through an activation function.
Generally, during the first layer, the principal features, namely the horizontal, vertical and diagonal edges, are detected. After this layer a little more complex features are extracted, i.e. corners and combinations of edges. Higher levels of features such as objects and faces can be distinguished in the deeper layers of the neural network.
During the last layer of a CNN, namely the classification layer which consists of fully-connected layers, the features obtained are combined in order to classify the image. At the end, a set of confidence scores is obtained and this set of values between 0 and 1 specifies the probability that the image belongs to a class. [15]

1.2.2 The CNN limitations

Even though the power and complexity of CNNs allow them to detect visual elements that might be invisible to the naked eye, they remain pattern-recognition machines. They are not adapted to understand the meaning of the images and work poorly when the context varies slightly as for example when images are viewed from new angles.
A problem that neural networks also face is their failure to make connections between different objects and understand their relations.
Another one is the vulnerability of networks to data disturbances which may be undetectable to human eyes but which impact them, as for example the addition of an imperceptible layer of noise.

Moreover, CNNs cannot be better than the input labels and are therefore dependent on their precision. They also need a large amount of images to be able to train models correctly.

Despite the restrictions of convolutional neural networks for certain applications, in the field of medical image processing, sufficiently well-trained CNNs sometimes surpass expert eyes. [15]

1.3 Deep learning-based image segmentation models

A large number of categories of algorithms for image segmentation tasks in deep learning exists. But this section is limited to the different methods used to segment medical images with encoder-decoder convolutional architectures and more particularly with the FCN and U-Net architectures. A network using a variant of convolution operations is also discussed in this section.

1.3.1 Encoder-Decoder models

A very popular category of algorithms for image segmentation tasks of medical images in deep learning is based on the encoder-decoder convolutional architecture. Among the most famous architectures there is U-Net [16].
Currently this type of models is no longer only reserved for the segmentation of medical images.

Fully Convolutional Networks

In 2015, Long et al. were among the first to do semantic segmentation of images by deep learning with fully convolutional networks (FCNs) [17]. In FCNs (see figure 1.7), convolutional layers are the only components so the fully-connected layers of CNN architectures were replaced by fully convolutional layers. These models therefore do not return classification scores but a spatial segmentation map.
Skip connections, which upsample the feature maps of the last layers and merge them with those of the previous layers, are used. This allows for a detailed and precise segmentation as the model mixes semantic information derived from deep layers with appearance information from relatively shallow layers. [3]

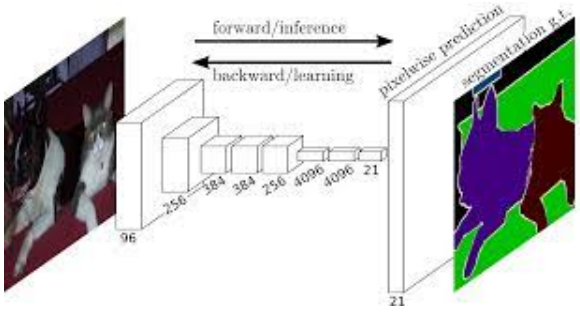


Figure 1.7: Illustration of a fully convolutional network. [17]

U-Net

The primary purpose of U-Net, created by Ronneberger et al., was to segment biological microscopy images. U-Net is a network which is made up of two parts: a contracting path as well as an expanding path.
In the contracting part, the images are given as input and the context is extracted. The information on the features is increased but this leads to a reduction of the spatial information. This first part has a FCN-like architecture and therefore is a traditional convolution network which mainly consists of the successive application of convolutions and maximum pooling. In a maximum pooling layer, a small neighborhood of a feature map is replaced by statistical information, i.e. the maximum, on the neighborhood. An illustration of a maximum pooling is shown on the figure 1.8.

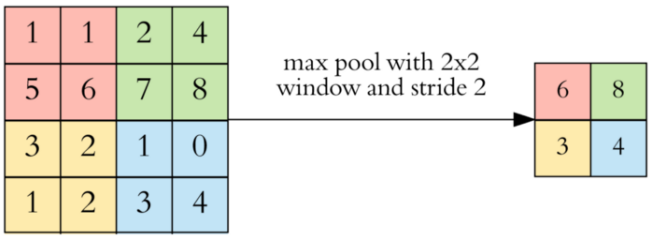


Figure 1.8: Result of a maximum pooling using a 2×2 window and a stride of 2 where only the maximum value of each window, represented by the different colors, is kept. [12]

Both convolution and maximum pooling operations downsample the image, thus convert a high-resolution image to a low-resolution image.
The expansive channel is used to have a precise localization by using, in particular, transposed convolutions which upsample to convert a low resolution image into a high resolution image. This part of upsampling increases the dimensions of the feature maps while reducing their numbers.

In order to avoid losing pattern information, feature maps from contracting path are replicated to the expansive path.
The last step consists of using an activation function which transforms a real vector into a probability vector allowing each pixel of the input image to be categorized. A general overview of the U-Net architecture in two dimensions can be seen in the figure 1.9.

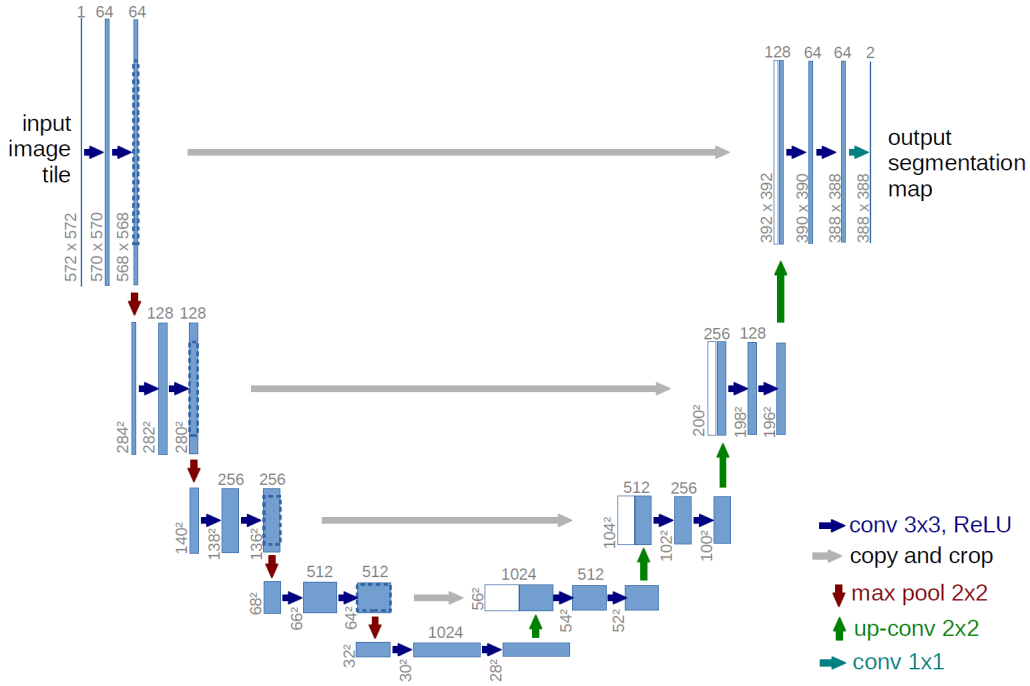


Figure 1.9: U-Net architecture where the blue boxes represent multi-channel feature maps. On the top of the boxes, the number of channels is indicated and at the lower left edge the x-y-size is given. The white boxes correspond to the copied feature maps. The different operations are denoted by the arrows. [16]

Variants of U-Net

Many adaptations of the U-Net architecture have emerged and some of which are listed below. The first four methods change at the level of the the layers and the others change at the level of the blocks.

3D U-Net : The three-dimensional images can be processed with the 3D U-Net of Ö. Çiçek [18] where the core structure is similar but all the 2D operations have been replaced by the corresponding 3D operations.

Attention U-Net : The attention U-Net allows to focus on main features useful for a specific task and ignore areas that are not important by using the attention gate [19]. An attention gate eliminates features that are irrelevant to the task at hand. As a result of repeated applications of the attention gate after each layer in the expansive path, segmentation performance is considerably improved without increasing computational complexity too much. [20]

Inception U-Net : A problem that most models face is the use of fixed-size filters for convolutions that are not suitable for images with variable size prominent regions. Reading high-level details over a range of sizes and shapes using deeper networks is a computationally expensive way to overcome this problem. Inception networks [21] offer an alternative to efficiently process images with different salient parts by using, for the same network layer, filters with variable sizes. A concatenation of the outputs of the filters is made. And in order to reduce the computational complexity, a 1×1 convolution is done before the filters of size 3×3 or more. [20]

V-Net : Another popular network based on the FCNs that allows segmentation of 3D medical images is V-Net which has been developed by Milletari et al. [22]. The objective function, for model training, has been replaced by a new one based on the dice coefficient. This modification allows to deal with data where a great imbalance is present between the classes representing the foreground and the background.

Residual U-Net : Increasing the number of layers improves performance and more particularly the speed of convergence. However, it is possible that increasing the number of layers leads to saturation. In addition, in deeper networks, the decrease of gradients in the weight vector causes the loss of feature identities, which is why some increases degrade performance. The objective of residual architectures [23] is to remedy the problem of forming relatively deep networks. Through the use of skip connections that add the feature map from one layer to another deeper, feature maps in deeper neural networks are better preserved and better performance is achieved. More precisely, a skip connection is made for each block of the network to add the input of the first convolutional layer to the output of the second one. The vanishing gradient problem is mitigated by these residual skip connections. [20]

Dense U-Net : With residual architectures, deeper networks are obtained but the problem of the vanishing of gradients persists. The dense architecture [24] is based on that of the residual one but with two fundamental differences: for each layer of a block the identity or feature maps of all its previous layers are sent to it and unlike the residual network which adds the identity cards element by element, a concatenation by channel-wise into tensors is done in order to combine these cards.

All the identity cards of the previous layers are thus preserved and the propagation of the gradient is favored. As the information is more easily kept between the layers, fewer channels are needed for each layer which reduces the computational cost while improving the precision. Therefore the number of layers of the models can be increased. This particularity is useful in the medical field given that in medical images there are regular overlays of objects which are considerably close. [20]

Recurrent U-Net : In recurrent neural networks feedback loops are used, i.e. the output of one node influences the next output of the same nodes. Applied to the whole layer, the contextual information of the previous data can thus be processed by the network. For the recurrent convolutional neural network [25] the feedback loops are incorporated after the convolution and the activation function and feed the feature map of the associated layer. Units thus know the context of adjacent units and can therefore update their features maps accordingly. [20]

U-Net++ : Zhou et al. [26] developed a nested U-Net architecture based on the dense architectures where a series of nested and dense skip pathways connect the encoder and decoder sub-networks. The difference with the classical U-Net therefore lies in the design of the skip pathways which allow to decrease the semantic gap between the sub-networks. This method facilitates the processing of a learning task for feature maps being semantically similar.

1.4 nnU-Net

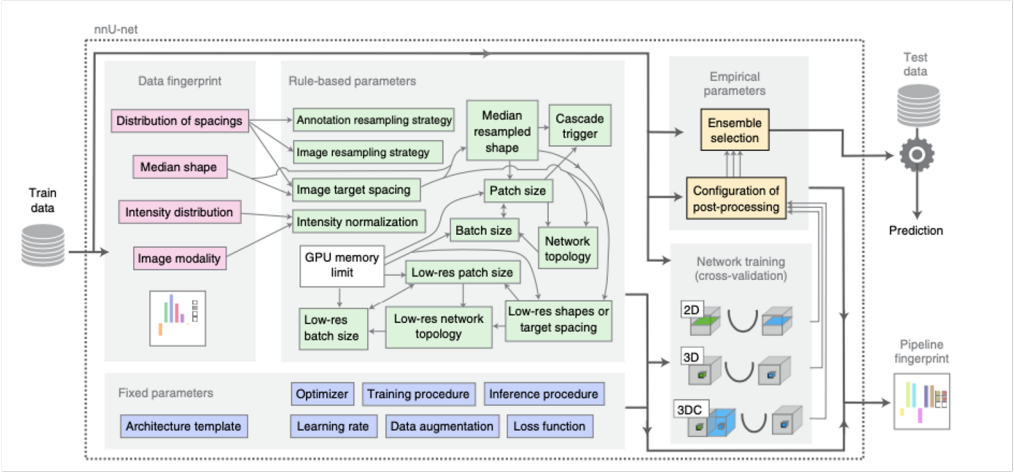
Despite the many deep learning-based segmentation methods, in most cases there are limits to their suitability for specific analysis problems of end-users. High levels of experience and expertise are required to design and configure a method for a specific task since small errors can dramatically impact performance. This is even more observed for 3D biomedical imaging due to the many different properties of the datasets such as the modality of the image, its size, the spacing of the voxels or the strong variations of the class ratio. Optimal configurations for one dataset are therefore rarely optimal for another. The segmentation results obtained are often sub-optimal since the design of the method is determined by the users through a manual and iterative process via trial and error and therefore depends on individual experience.

The segmentation method based on deep learning nnU-Net (no-new U-Net) [27] developed by F. Isensee et al. therefore has all its importance and establishes a new state of the art in various challenges of semantic segmentation because this method is automatically configured itself for any new task in the biomedical field, whether for pre-processing, network architecture, training or post-processing.

More precisely, the nnU-Net method contains three groups of parameters and allows to systematize the configuration process independently of the task. In other words, when a new task is given the search space for the empirical design is drastically reduced. The three different groups of parameters are listed below.

- *Fixed parameters* are responsible for determining a robust common configuration by identifying design decisions common to all datasets.
- *Rule-based parameters* relate to explicit relationships between specific properties of a dataset with design choices that are formulated as heuristic rules. This allows adaptation on application almost instantly. Examples of such rules are as follows: increasing the size of the batches improves the estimates of the gradients, the number of contextual information known by the network increases with the size of the patches, or the depth of the topology of a network must be sufficient in order that the contextual information is taken into account.
- *Empirical parameters* are used to learn the remaining decisions from the training data, namely model selection and post-processing.

The figure 1.10 shows how nnU-Net works. First, the fingerprint of the dataset is extracted followed by the identification of heuristic rules. Three U-Net configurations are tested: 2D U-Net, 3D U-Net and a 3D U-Net cascade where a first 3D U-Net is applied on downsampled data then the segmentation maps created are refined with a second 3D U-Net at full resolution. Once the five-fold cross-validation is complete, the optimal configuration is determined empirically.
The configuration is automatic, so there is no need for manual intervention when a new dataset is applied. The only computational costs therefore come from the standard network learning procedure and the few remaining empirical decisions.



Design choice	Required input	Automated (fixed, rule-based or empirical) configuration derived by distilling expert knowledge (more details in online methods)
Learning rate	–	Poly learning rate schedule (initial, 0.01)
Loss function	–	Dice and cross-entropy
Architecture template	–	Encoder–decoder with skip-connection ('U-Net-like') and instance normalization, leaky ReLU, deep supervision (topology-adapted in inferred parameters)
Optimizer	–	SGD with Nesterov momentum ($\mu = 0.99$)
Data augmentation	–	Rotations, scaling, Gaussian noise, Gaussian blur, brightness, contrast, simulation of low resolution, gamma correction and mirroring
Training procedure	–	1,000 epochs × 250 minibatches, foreground oversampling
Inference procedure	–	Sliding window with half-patch size overlap, Gaussian patch center weighting
Intensity normalization	Modality, intensity distribution	If CT, global dataset percentile clipping & z score with global foreground mean and s.d. Otherwise, z score with per image mean and s.d.
Image resampling strategy	Distribution of spacings	If anisotropic, in-plane with third-order spline, out-of-plane with nearest neighbor Otherwise, third-order spline
Annotation resampling strategy	Distribution of spacings	Convert to one-hot encoding → If anisotropic, in-plane with linear interpolation, out-of-plane with nearest neighbor Otherwise, linear interpolation

Image target spacing	Distribution of spacings	If anisotropic, lowest resolution axis tenth percentile, other axes median. Otherwise, median spacing for each axis. (computed based on spacings found in training cases)
Network topology, patch size, batch size	Median resampled shape, target spacing, GPU memory limit	Initialize the patch size to median image shape and iteratively reduce it while adapting the network topology accordingly until the network can be trained with a batch size of at least 2 given GPU memory constraints. for details see online methods.
Trigger of 3D U-Net cascade	Median resampled image size, patch size	Yes, if patch size of the 3D full resolution U-Net covers less than 12.5% of the median resampled image shape
Configuration of low-resolution 3D U-Net	Low-res target spacing or image shapes, GPU memory limit	Iteratively increase target spacing while reconfiguring patch size, network topology and batch size (as described above) until the configured patch size covers 25% of the median image shape. For details, see online methods.
Configuration of post-processing	Full set of training data and annotations	Treating all foreground classes as one; does all-but-largest-component-suppression increase cross-validation performance? Yes, apply; reiterate for individual classes No, do not apply; reiterate for individual foreground classes
Ensemble selection	Full set of training data and annotations	From 2D U-Net, 3D U-Net or 3D cascade, choose the best model (or combination of two) according to cross-validation performance

Figure 1.10: Illustration of the automated method configuration. For a new task, a dataset fingerprint (pink) is extracted. Parameter inter-dependencies (represented by thin arrows) are modeled by some heuristic rules which acts on this fingerprint to deduce the rule-based parameters (green). Then the fixed parameters (blue) are added. Three different configurations are trained and nnU-Net finishes with the empirical parameters (yellow) to choose the optimal ensemble and to determine if post-processing is needed. Explicit values and summarized rule formulations of each configured parameters are listed in the table on the bottom. [27]

nnU-Net has been tested on 11 international biomedical imaging segmentation challenges containing 23 distinct datasets and 53 segmentation tasks. These datasets are composed of different organs, tumors, lesions with both 2D and 3D images from various modalities such as resonance imaging or computer tomography. The training of nnU-Net was made from scratch without the use of data other than those of the challenge. From results obtained it appears that nnU-Net is able to manage many different target structures and image properties.
Moreover, nnU-Net outperforms specialized pipelines for various tasks. The fact that nnU-Net is generic does not prevent it from surpassing the segmentation solutions specially optimized for the respective task. In 33 of 53 tasks nnU-Net establishes a new state of the art and for the others performance it is quite similar to those of the top of the ranking.

From the results of nnU-Net some conclusions can be drawn. Performance is more impacted by configuration details than by architectural changes. To illustrate this point, the results obtained in the 2019 Kidney and Kidney Tumor Segmentation (KiTS) challenge [28] can be analyzed. It emerges from this ranking that the 15 methods with the best performance are variants of 3D U-Net from 2016 [29], [30]. In addition, good performance on the KiTS task can be obtained without architectural changes such as attention mechanisms [19], residual connections [23] or even dense connections [24]. For the KiTS dataset nnU-Net experimentally highlights that the method configuration has more impacts than the architectural changes by establishing a new state of the art with a simple 3D U-Net architecture on the open ranking.
Another conclusion is that different pipeline configurations are necessary for distinct datasets. The fact that the optimal model is influenced by the data fingerprint is the reason why there is a lack of out-of-the-box segmentation algorithms since a new optimization must be done for a new dataset.

However, the configuration of nnU-Net may not always be optimal. For example, the fact that nnU-Net evaluates performance primarily on the basis of the dice score may lead to sub-optimal results for tasks involving very domain-specific target metrics.
Sub-optimal performance can also be due to dataset properties which are not yet taken into account. This situation has been met with the synaptic cleft segmentation task of the CREMI challenge [31] where the performance may be outperformed with a manual adaptation of the loss function and an expectation-maximization-specific pre-processing [32]. Extending the heuristics for recurring cases or considering nnU-Net as a good starting point are ways to remedy cases that are not sufficiently addressed by nnU-Net.

In summary, nnU-Net is a self-configuring method which can be applied to a wide variety of medical imaging datasets and does not require user intervention. nnU-Net sets a new state of the art in many semantic segmentation challenges and has important generalization attributes. Good performance is not obtained thanks to new neural network architecture, training scheme or loss function but thanks to the fact that the process of manual configuration is systematized thereby avoiding onerous manual tuning and empirical procedures with practical limitations.

Now that deep neural networks with more particularly convolutional neural networks and the self-configuring method, nnU-Net have been developed in depth, we can see more precisely how they are going to be used in this work to segment blood vessels.

Chapter 2

Methodology

In the case of processing 3D medical volumes, working with 2D volumes by applying 2D CNNs in a slice-wise way does not preserve the important 3D context information which is essential to allow tracking of three-dimensional curvilinear structures. This is why it is preferable to process 3D volumes directly. However, using 3D datasets leads to challenges in terms of memory consumption and speed.

2.1 The DeepVesselNet method

In this section, a deep learning approach named DeepVesselNet [33] which uses cross-hair filters is presented and will be used in the implementation of neural network architectures to segment blood vessels. One of the particularities of cross-hair filters of DeepVesselNet is that it tries to remedy memory and speed problems.

The works [34], [35] and [36] introducing the notion of separable filters and using intersecting 2D planes served as a basis in the development of cross-hair filters from three intersecting 2D filters. The purpose of these filters is to preserve the 3D information of the volumes while dealing with the problems of memory and speed of standard 3D networks.

As seen in chapter 1, convolutions are used for CNNs. In the case of three-dimensional biomedical imaging, where I is a 3D volume, M is a 3D convolutional kernel with a shape (k_x, k_y, k_z), 3D convolutions are defined as

$$I * M = A = \{a_{ijk}\},\ a_{ijk} = \sum_{r=1}^{k_x} \sum_{s=1}^{k_y} \sum_{t=1}^{k_z} I_{(R,S,T)} M_{(r,s,t)}, \tag{2.1}$$

$$R = i + r - \left(1 + \left[\frac{k_x}{2}\right]\right),\ S = j + s - \left(1 + \left[\frac{k_y}{2}\right]\right),\ T = k + t - \left(1 + \left[\frac{k_z}{2}\right]\right) \tag{2.2}$$

with $*$ a convolutional operator, $\{a_{ijk}\}$ a position element of matrix A, $I_{(R,S,T)}$ the intensity value of volume I at voxel position (R, S, T), $M_{(r,s,t)}$ the value of kernel M at position (r, s, t) and $[x]$ represents the greatest integer less or equal to x.
For a standard 3D convolution the computational complexity is consequent. Equation 2.1 shows that for each voxel of the resulting image, $k_x k_y k_z$ multiplications and $k_x k_y k_z - 1$ additions are necessary. To illustrate this, with a $128 \times 128 \times 128$ volume and a kernel of size $5 \times 5 \times 5$, about 262×10^6 multiplications and 260×10^6 additions are used.

The interest of cross-hair filters, which are made up of three 2D crossed filters, is to manage this complexity and approximate the classical 3D convolution as follows

$$a_{ijk} = \sum_{s=1}^{k_y}\sum_{t=1}^{k_z} I_{(i,S,T)} M^i_{(s,t)} + \sum_{r=1}^{k_x}\sum_{t=1}^{k_z} I_{(R,j,T)} M^j_{(r,t)} + \sum_{r=1}^{k_x}\sum_{s=1}^{k_y} I_{(R,S,k)} M^k_{(r,s)} \tag{2.3}$$

with M^i, M^j and M^k the 2D convolutional (cross-hair) kernels in order to approximate the 3D kernel M from equation 2.1 along the ith, jth and kth axes respectively.
By replacing the classical 3D kernels by the cross-hair kernels, the number of multiplications and additions decreases to $k_y k_z + k_x k_z + k_x k_y$ and $k_y k_z + k_x k_z + k_x k_y - 1$ respectively. With the example used previously, less than 158×10^6 multiplications and 156×10^6 additions are necessary for the convolution therefore a decrease in calculation by more than 100×10^6 for both is observed. The increase in the size of the volume or the kernel widen the gap between computation cost of 2.1 and 2.3 even more.

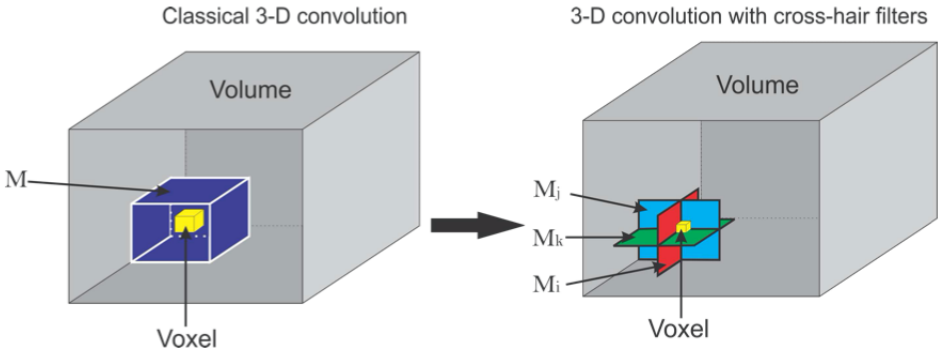


Figure 2.1: Graphical representation of cross-hair filters for 3D convolutional operation. On the left, a classical 3D convolution with filter M and on the right a cross-hair 3D convolutional with 2D filter stack M_i, M_j, M_k. [33]

Neighborhood information is shared between voxels close to each other, therefore applying the cross-hair filter as described in the equation 2.3 independently to each voxel is not optimal in terms of implementation since multiple copies of it are made and so leads to redundant memory usage. A more memory efficient implementation (shown in figure 2.2) can be obtained by extracting the sagital, coronal and axial planes as I^s, I^c and I^a respectively. And can be expressed as

$$I \diamond M = A = \beta_c A^c + \beta_s A^s + \beta_a A^a \tag{2.4}$$

$$A^c = I^c * *M^i,\ A^s = I^s * *M^j,\ A^a = I^a * *M^k \tag{2.5}$$

with $**$ denotes a 2D convolution of the left matrix along the first and second axes over all slices in the third one, $\diamond$ refers to the cross-hair filter operation and with the weights β_c, β_s and β_a that manage the planes input towards the final sum, for instance, when the spatial resolutions of the planes are distinct. In this work, $\beta_c = \beta_s = \beta_a = 1$ is fixed. What changes in memory usage is that instead of having copies of a volume, only one volume is considered for voxels from the same neighborhood and the kernels are rotated so that the slices correspond in the different orientations. With this implementation, the memory consumption is therefore reduced.

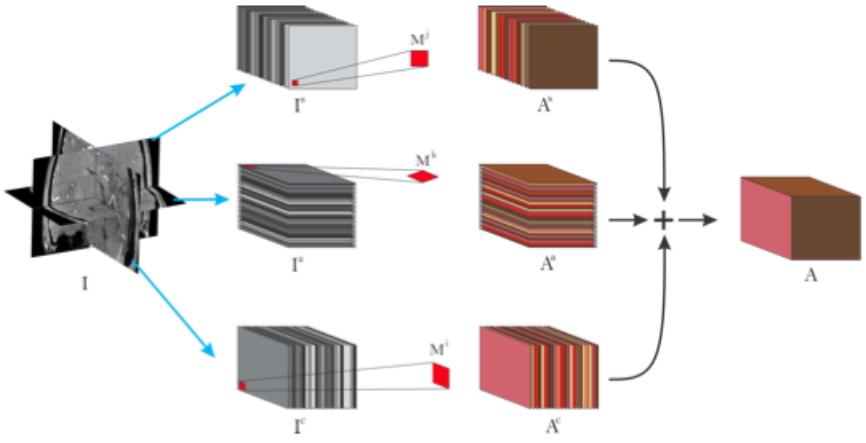


Figure 2.2: Pictorial view of efficient implementation of cross-hair filters. The gray-scaled stacks refer to input to the layer, the red shaped squares refer to 2D kernels used for each plane. The brown slices refer to extracted features after convolution operations and the + symbol refers to matrix addition. [33]

Since 2D architectures will be also used in the experiments of this master thesis, the 3D convolution described in the equation 2.1 can simply be adapted to the 2D cases as

$$I * M = A = \{a_{ij}\},\ a_{ij} = \sum_{r=1}^{k_x} \sum_{s=1}^{k_y} I_{(R,S)} M_{(r,s)}, \tag{2.6}$$

And the cross-hair filters expressed by the equation 2.3 can be adapted as

$$a_{ij} = \sum_{s=1}^{k_y} I_{(i,S)} M^i_{(s)} + \sum_{r=1}^{k_x} I_{(R,j)} M^j_{(r)} \tag{2.7}$$

It is important to note that the 3D networks with cross-hair filters differ from 2.5D networks.
To use a 2.5D network on a 3D dataset, first of all a pre-processing stage must be done to extract the 2D slices which will then be given as inputs to the network. The network classifies, for each slice of the volume, all the pixels. Finally a post-processing stage is achieved in order to reform the 3D volume from the 2D results obtained. In summary, 2.5D networks work like 2D networks with pre-processing and post-processing methods. What is predicted by these networks is therefore only based on 2D contextual information.
2.5D networks have for example been used to segment liver and lesion in computed tomography (CT) volumetric dataset with a U-Net architecture [37] or to annotate lumbar vertebrae [38]. Some networks add a pre-processing method to extract several 2D planes and to use them as input channels [35], [36].
For 3D networks implemented with cross-hair filters, the input is the whole 3D volume so there is no need for a pre-processing or post-processing stage. Unlike the 2.5D networks, the 3D context information is used by the cross-hair filters at each convolutional layer. But this method also differs from standard 3D networks since there are no full 3D convolutions so less parameters and memory are used.
However, the filters of 2.5D networks and 3D networks with cross-hair filters operate the same for scaling, i.e. in only two directions, regarding changes in filter and volume sizes. For example, with a cubic filter of size k, there are k^2 parameters with a 2.5D network and $3k^2$ for the cross-hair filter based network. In general, adding a factor r to the size of the filter increases quadratically for both networks, i.e. $(k+r)^2$ for 2.5D and $3(k+r)^2$ for cross-hair filter while for classical 3D networks the parameter size cubically scales, i.e. $(k+r)^3$.

In summary, unlike the existing 2.5D networks that use 2D slices, obtained from a pre-processing stage, as inputs in a 2D network, the cross-hair filters based networks preserve the crucial 3D information thanks to the implementation of

the cross-hair filters on a layer level. Which is more advantageous for detecting curvilinear 3D objects. In addition, compared to 3D architectures, they improve speed and memory usage.
In the rest of this work, 2.5D networks will be mentioned as 2D networks for ease.

2.2 Datasets

In this master thesis two distinct 3D datasets, representing blood vessels, are used to train and test different networks. The particularity of the blood vessel datasets is the great imbalance of the classes: a minority of the total voxels represent the vessels. This problem of bias towards the background class is recurrent in medical data.

2.2.1 Synthetic data

In a first time a synthetic dataset is used and can be downloaded from the DeepVesselNet paper's github page at `https://github.com/giesekow/deepvesselnet/wiki/Datasets`. It contains 136 volumes of size $325 \times 304 \times 600$ where voxel values are between 0 and 255, and their corresponding labels for vessel segmentation.
The generation of this dataset is inspired by the method of [39], [40] that consists in a simulator of a vascular tree following a generative process based on the biology of angiogenesis. Currently this simulator produces physiologically realistic vascular trees that can be used for deep learning vessel segmentation tasks. A cylindrical tube of radius r and length l in 3D space is used to model each vessel segment, depicted by two nodes linked by a directed edge. The Murray's law, or bifurcation law is applied to generate the tree by determining the relation between the radius of parent bifurcation r_p and the radius of left r_l and right r_r daughter branches as

$$r_p^\gamma = r_l^\gamma + r_r^\gamma \tag{2.8}$$

with γ the bifurcation exponent.
Moreover the bifurcation angles of the left ϕ_l and right ϕ_r are determined by respectively

$$\cos(\phi_l) = \frac{r_p^4 + r_l^4 - r_r^4}{2r_p^2 r_l^2} \quad and \quad \cos(\phi_r) = \frac{r_p^4 + r_r^4 - r_l^4}{2r_p^2 r_r^2} \tag{2.9}$$

The figure 2.3 illustrates the tree generation model and the bifurcation configuration.

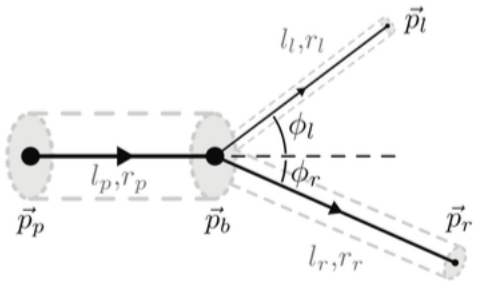


Figure 2.3: A representation of the bifurcation configuration, as presented in [39], where l_p, l_r and l_l are the length of the parent, right daughter, and left daughter segments, respectively. p_p, p_r and p_l are the parent, the right and the left daughter nodes respectively [33]

For non-vessel intensities, a random selection is made in the range [0, 100] and for vessel intensities it is in the interval [128, 255]. Afterwards, for each generated volume, a Gaussian noise is applied so that the mean varies randomly in the range [-5, 5] and the standard deviation changes in the range [-15, 30]. 2.1% of total voxels represent the vessel labels. The figures 2.4a and 2.4b show an image of size $325 \times 304 \times 600$ of the synthetic dataset and its corresponding true mask.

(a) Input image

(b) True mask

Figure 2.4: Example of a 3D input image of size $325 \times 304 \times 600$ of the synthetic dataset and its corresponding true mask

2.2.2 Real data

The second dataset used is the Hepatic Vessel dataset from the Medical Segmentation Decathlon (D8) and can be downloaded on the web page `https://drive.google.com/drive/folders/1HqEgzS8BV2c7xYNrZdEAnrHk7osJJ--2`.

It contains 303 volumes with voxel values are between -1024 and 4543 and with sizes varying from $512 \times 512 \times 24$ to $512 \times 512 \times 181$, with the median shape of $512 \times 512 \times 49$. The images are CT scans and the rarest class ratio is about 1.1×10^3. The dataset contains 3 classes: background, vessels and liver tumors but this master thesis dealing only with the segmentation of vessels, the voxel values belonging to the tumor class have been set to zero.
A semi-automatically segmentation was made thanks to the Scout application [41]. Briefly, via a level-set based method, the seed point drawn on the region of interest has been grown. Then an expert abdominal radiologist manually refined the contours. [42] An example of the target structure can be found at the figure 2.5. [27] Another example of an image with its corresponding label is also shown on the figures 2.6a and 2.6b.

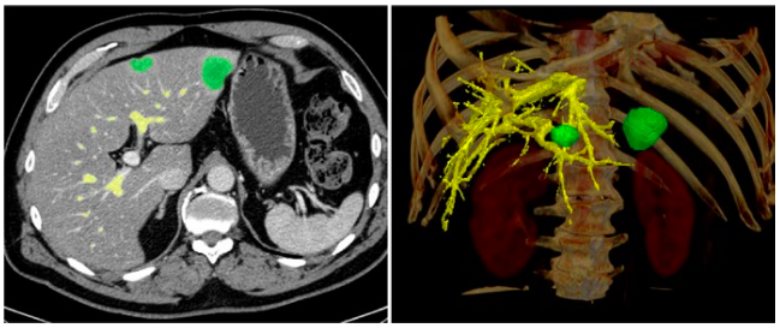
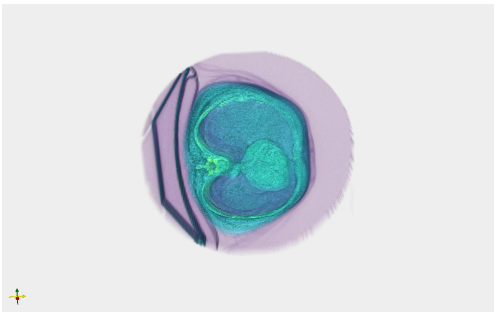
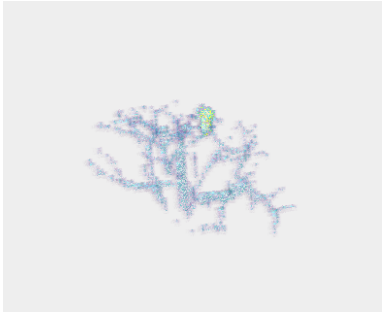


Figure 2.5: Target structure for the real dataset projected onto the raw data in 2D on the left and in 3D on the right. The visualizations were created with the MITK Workbench [43]. The hepatic vessels are represented in yellow and liver tumors in green. [27]



(a) 3D image



(b) Zoom of the mask

Figure 2.6: Example of a 3D image of size $512 \times 512 \times 49$ of the hepatic vessel dataset and its corresponding true mask enlarged

2.3 Necessary information for experiments

2.3.1 DeepVesselNet-UNet

This master thesis aims to compare 2D and 3D architectures performance in terms of blood vessel segmentation. This comparison will be made with one of the most popular architectures introduced in chapter 1, i.e. U-Net. However, it is not the 2D U-Net architecture presented on the figure 1.9 that will be used. But the one illustrated on the figure 2.7, a 3D U-Net where the mainly difference is the number of channels after each layer.
In this 3D U-Net, the convolutions are replaced by cross-hair filters based convolutions to obtain 3D DeepVesselNet-UNet (DVN). 2D U-Net and 2D DeepVesselNet-UNet which will also be used for the comparison are obtained by replacing both classic 3D convolutions and cross-hair convolutions by their corresponding 2D convolutions.
It is important to note that the cross-hair filters can be used for all types of convolutional architectures, not only U-Net, by replacing traditional convolutions by those using the cross-hair kernels.

The initialization of the network parameters is random and is based on the method suggested in [44], the biases are set to zero and the weights W_{ij} at each layer follow the uniform distribution

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{k_x k_y k_z}}, \frac{1}{\sqrt{k_x k_y k_z}}\right] \tag{2.10}$$

with $\left(-\frac{1}{\sqrt{k_x k_y k_z}}, \frac{1}{\sqrt{k_x k_y k_z}}\right)$ the interval and $k_x \times k_y \times k_z$ the size of the kernel.

2.3.2 Parameters used for experiments

In chapter 3, several experiments are carried out with different network configurations. Some basic parameters are already presented in chapter 1 but more specific ones used in this master thesis are briefly explained below.

Optimization algorithms

Adam : Adaptive moment estimation (Adam) [45] can be used to get results faster, including reducing the number of function evaluations or getting better results. For all parameters, the step size is optimized and during the whole search process, they are automatically modified based on the gradients. [46]

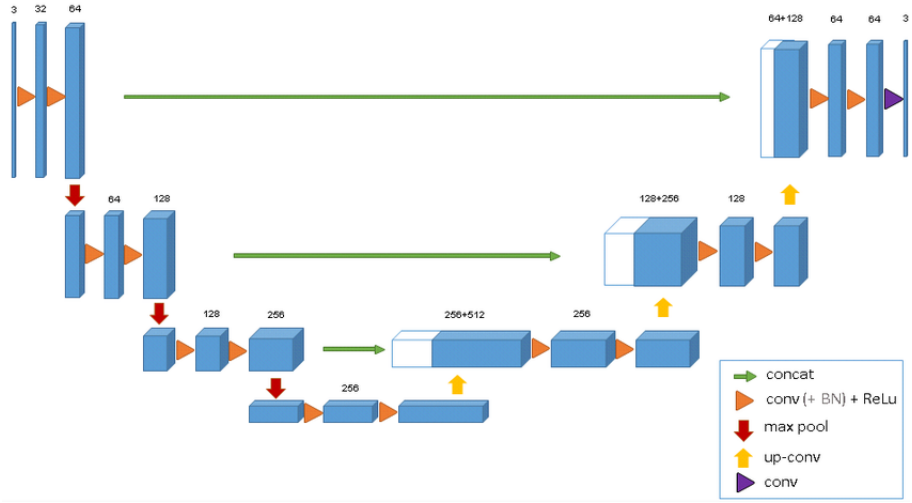


Figure 2.7: 3D U-Net used for the experiments where blue boxes represent feature maps and the number of channels is denoted above each feature map. [18]

Stochastic gradient descent : Unlike gradient descent (see chapter 1) which takes all the training dataset per epoch, stochastic gradient descent (SGD) only uses one randomly selected sample. It reduces the training time and the computational cost even though more iterations are needed to reach the minima. However this random selection leads to a large fluctuation of the loss function. These oscillations avoid being stuck in a local minimum by jumping to potentially better one but reaching the exact minimum is therefore more complicated since SGD risks to overtake it. [47]
Add a momentum to the SGD allows to have a faster convergence of the gradient in the relevant direction and to dampen oscillations by increasing the dimensions where gradients have the same directions and decreasing the others. In particular, the Nesterov momentum, introduced by Y. Nesterov, works better than classical momentum since the gradient is computed in function of approximation of future position of the parameters and not in function of the current parameters. [47] The update rule for parameter w with the Nesterov momentum can be formulated as

$$\text{velocity} = \text{momentum} * \text{velocity} - \text{learning rate} * g \tag{2.11}$$

where g is the current gradient,

$$w = w + \text{momentum} * \text{velocity} - \text{learning rate} * g \tag{2.12}$$

Regarding the learning rate, two different ways to adapt it will be used. The first one, which we will call the simple decay for ease, is simply a learning rate which decays as

$$\text{learning rate} = \text{initial learning rate} * \frac{1}{1 + \text{decay} * \text{iterations}} \tag{2.13}$$

The second learning rate is reduced by using the 'poly' learning rate schedule depicted in [48] which decreases almost linearly to zero

$$\text{learning rate} = \text{initial learning rate} * \left(1 - \frac{\text{iteration}}{\text{max iterations}}\right)^{\text{power}} \tag{2.14}$$

where power is fixed to 0.9.

Loss function

Cross-entropy : The cross-entropy can be expressed as the negative average of the log of corrected predicted probabilities. The use of the log value allows to penalize more strongly the large differences between predicted probability and corrected probability compared to the smallest ones. With cross-entropy, the loss function L is computed with

$$L(W) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log\left[P(y_i = 1|X;W)\right] + (1 - y_i)\log\left[1 - P(y_i = 1|X;W)\right]) \tag{2.15}$$

with N the number of classes, P the probability of getting the ground truth label given the data X and the parameters W, y_i the label of the ith class and X the feature set. The cross-entropy loss can also be expressed as

$$L(W) = -\frac{1}{N}\left(\sum_{i\in Y_+}\log\left[P(y_i = 1|X;W)\right] + \sum_{i\in Y_-}\log\left[P(y_i = 0|X;W)\right]\right) \tag{2.16}$$

where Y_+ represents the set of positive labels and Y_- is the background, i.e. the set of negative labels. [49], [33]

Class-balancing cross-entropy : The classes that are extremely unbalanced leads to a bias towards the identification of the background voxels to the detriment of vessels. Therefore a variant of the categorical cross-entropy (equation 2.16) is used, the class-balancing cross-entropy [50]. This loss L has the following formulation

$$L(W) = -\beta\sum_{i\in Y_+}\log\left[P(y_i = 1|X;W)\right] - (1 - \beta)\sum_{i\in Y_-}\log\left[P(y_i = 0|X;W)\right] \tag{2.17}$$

with β and $1 - \beta$ the class weighting multipliers computed as $\beta = \frac{|Y_-|}{|Y|}$ and $1 - \beta = \frac{|Y_+|}{|Y|}$. [33]

2.3.3 Evaluation criteria

Usually several criteria are used to evaluate a model such as quantitative accuracy, storage requirements and speed. Nevertheless until now the majority of work for the segmentation algorithms mainly uses metrics to assess the accuracy of the model to the detriment of the other criteria.
For the metrics, the predicted labels are compared with the real ones in order to evaluate the predictions of the models. Among the most popular metrics that allow the accuracy of models to be evaluated and that are used in this work, there are the specificity, the precision, the recall and the dice score.
In addition to these quantitative metrics, the visual quality of the results obtained by the models is an important factor to take into consideration when choosing the best model.

Some concepts are important to mention before introducing the different metrics because they are used in the calculations of the latter.
A pixel which is rightly identified as belonging to the given class (according to the target mask) is a true positive (TP). While a true negative (TN) is a pixel rightly predicted as not to belong to the given class. The false positive (FP) and the false negative (FN) represent a prediction which is not in accordance with reality.

Specificity is a statistical measures used in the medical field to evaluate the percentage of negative values which are correctly identified. It can be expressed as

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{2.18}$$

Precision represents the "purity" of the positive predictions compared with the true mask, i.e. how many predicted pixels have a corresponding ground truth annotation. [51] For each class the precision can be determined by

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.19}$$

Recall represents the "completeness" of the positive predictions compared with the ground truth, i.e. how many annotated pixels in the ground truth are considered as positive predictions. [51] For each class the recall can be defined as

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.20}$$

F1-score combines precision and recall rates and which is established as the harmonic mean of precision and recall [3]

$$\text{F1-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.21}$$

Dice coefficient is a frequently used metric in the field of medical imaging specially for the segmentation and is determined by the division between the double of the overlap region of predicted and ground truth maps, with the total number of pixels of the two images.

$$\text{Dice score} = \frac{2|A \cap B|}{|A| + |B|} \tag{2.22}$$

with A representing the ground truth and B the predicted segmentation maps.
The dice coefficient is similar to the F1-score, in the case of boolean data and in this master thesis, with the foreground corresponding to the positive class and the background to the negative class. [3]

$$\text{Dice score} = \frac{2TP}{2TP + FP + FN} = \text{F1-score} \tag{2.23}$$

In this work attention will be also paid to the use of memory and speed in order to evaluate not only the performance of the models but also the performance of the cross-hair filters.
The execution time is compared with the average and the standard deviation over 10 epochs obtained by processing 10 volumes.
To evaluate the impact of the cross-hair filters, and more particularly for the memory consumption, the number of convolutional parameters used in the network are computed. For one convolution in a 3D U-Net it can be computed as

$$\text{number of parameters} = (k_x k_y k_z l + 1)m \tag{2.24}$$

with a kernel of size $k_x \times k_y \times k_z$, l the number of the feature inputs, m the number of feature outputs and the addition of one is for the bias term.
For one convolution with cross-hair filters, the number of convolutional parameters is given by

$$\text{number of parameters} = (k_y k_z l + 1)m + (k_x k_z l + 1)m + (k_x k_y l + 1)m \tag{2.25}$$

For the number of parameters, in the 2D cases, the equations 2.24 and 2.25 can be adapted respectively as

$$\text{number of parameters} = (k_x k_y l + 1)m \tag{2.26}$$

and

$$\text{number of parameters} = (k_y l + 1)m + (k_x l + 1)m \tag{2.27}$$

It is also interesting, during the model training, to check how the model behaves. For this, learning curves representing the learning performance of a model are reviewed. These analyzes can be useful in detecting learning problems, such as

underfitting (training data is not well modeled and not good generalization for new data) or overfitting (training data is modeled too well) the model. Since during the training, two sets (training and validation) are used there are two types of curves: the train learning curves, obtained on the basis of the training data, give an overview on the learning quality model and validation learning curves, based on validation data, provide an idea into the effectiveness of model generalization.
Here are used the optimization learning curves, i.e. the loss curves, to analyze if the model behaves badly or well after each optimization iteration.

In summary, we have seen that in theory, feeding the network with 3D data made it possible to have better predictions than with 2D slices obtained by pre-processing the 3D dataset.
However, the use of 3D data increases time and memory consumption, hence the interest of using cross-hair filters. It is therefore interesting to verify whether this is confirmed in practice. This is what is done in the next chapter (Application and Results).

Chapter 3

Application and Results

In this chapter, the use of 2D and 3D DeepVesselNet-UNets (DVNs) in the tasks of vessel segmentation is analyzed. For this purpose, comparisons with classical 3D and 2D U-Nets are made. More precisely, comparisons of the number of parameters and the execution time are achieved to analyze the properties of the cross-hair filters, i.e. if there are improvements in memory usage and gain in speed. An analysis is also done to evaluate their influence on the prediction accuracy on the two datasets, synthetic and real presented in chapter 2.

3.1 Application on synthetic data

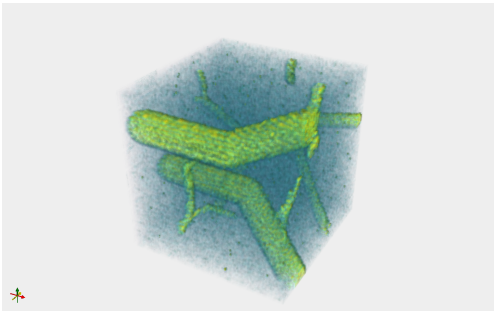
First the analyzes on the synthetic data are achieved. For these experiments, the choices of the different parameters were made on the basis of some trial and error in order to obtain good performance.

3.1.1 3D DeepVesselNet-UNet and 3D U-Net

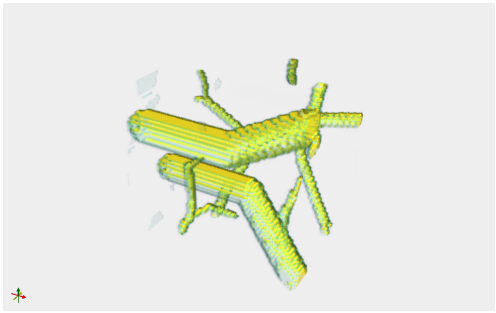
Data pre-processing

Before passing the images through the network, they are first pre-processed.
The images are normalized, voxel values are rescaled from the range of [0,255] to the range [0,1]. For each volume the size has been adjusted ($336 \times 312 \times 600$ instead of $325 \times 304 \times 600$) by zero padding to facilitate the extraction of 360 patches of size $64 \times 64 \times 64$. Patches are extracted to fine-tune the parameters by feeding them through the network. An example is shown on the figures 3.1a and 3.1b (one patch and its corresponding labels) and the figures 3.1c and 3.1d represent one of their slices. The patches are useful to have a faster training and to use more efficiently the computation memory.

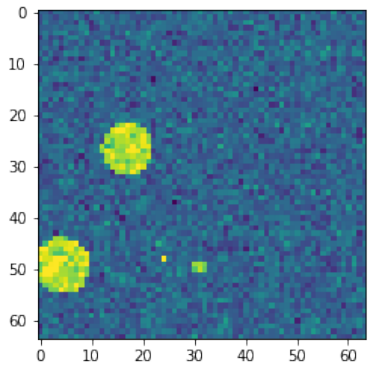
The dataset is split into three different groups: training, validation and testing. The training set is composed of only 20 volumes, i.e. 7200 patches, since the task is quite easy. The validation set represents 20% of the training set (4 volumes or 1440 patches) and the test set contains 20 volumes.



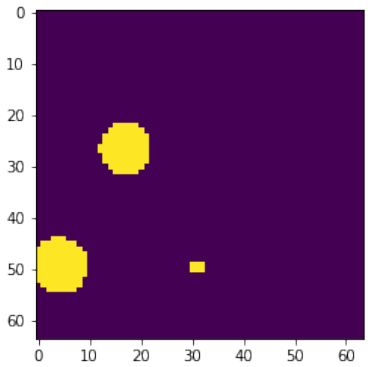
(a) Normalized input image



(b) True mask



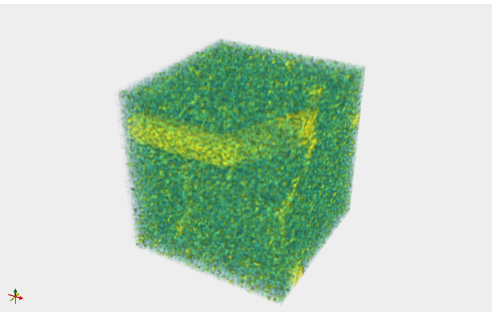
(c) Slice of the normalized input image



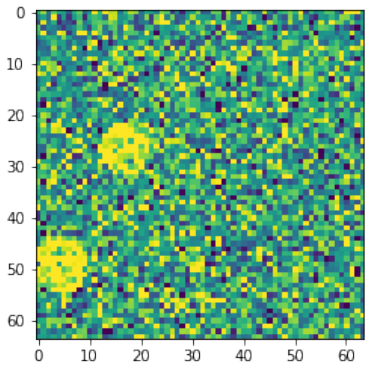
(d) Slice of the true mask

Figure 3.1: One patch of size $64 \times 64 \times 64$ of a normalized input image, its corresponding labeled patch and the corresponding slices of size 64×64

Since the segmentation of this normalized dataset is very easy, another Gaussian noise is added to the normalized inputs with a randomly varying mean in the range [-0.9,0.9] and a standard deviation in the range [0,0.9]. The figures 3.1a and 3.1c with such a Gaussian noise are illustrated on the figure 3.2a and 3.2b.



(a) Normalized input image with noise



(b) 2D normalized input image with noise

Figure 3.2: Same patch as the figure 3.1a with a Gaussian noise and a corresponding slice

Network configurations

Once the dataset is pre-processed, the DeepVesselNet-UNet and the U-Net architectures are configured. For the vessel segmentation of the synthetic dataset a binary network is implemented.
Optimization algorithm : The optimization algorithm used is Adam with a learning rate of 0.0001.
Loss function : The loss function used is the binary cross-entropy one. For binary cross-entropy, comparisons between the predicted probabilities and the real classes where outputs are binaries (0 or 1) are made.
Activation function : The activation function used is ReLU.

Moreover the network composition can be seen on the figure 3.3 where the number of channels is denoted above each feature map. In this experiment, in particular 4 levels are used and the initial number of features is 8. For the next experiments the composition is almost the same, only the number of levels and initial feature may vary.

The model is trained during 50 epochs with a batch size of 8 and the kernels have a size of $5 \times 5 \times 5$.

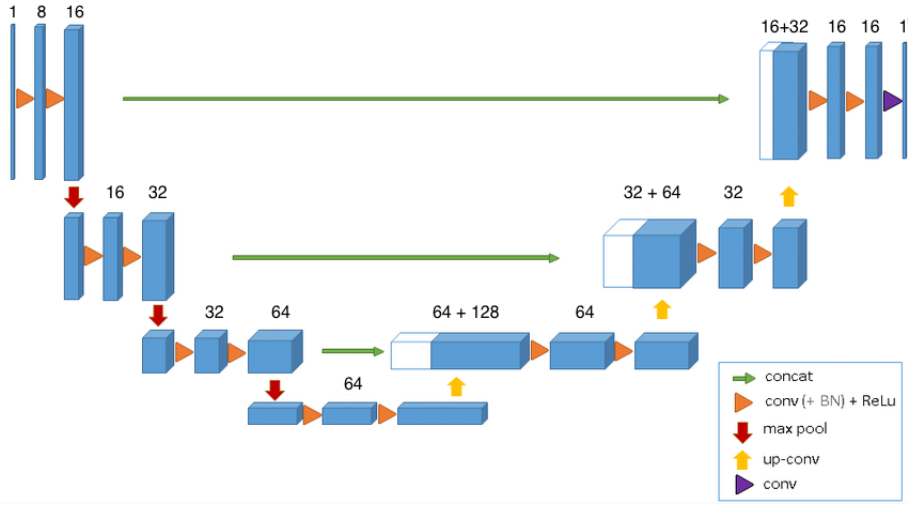
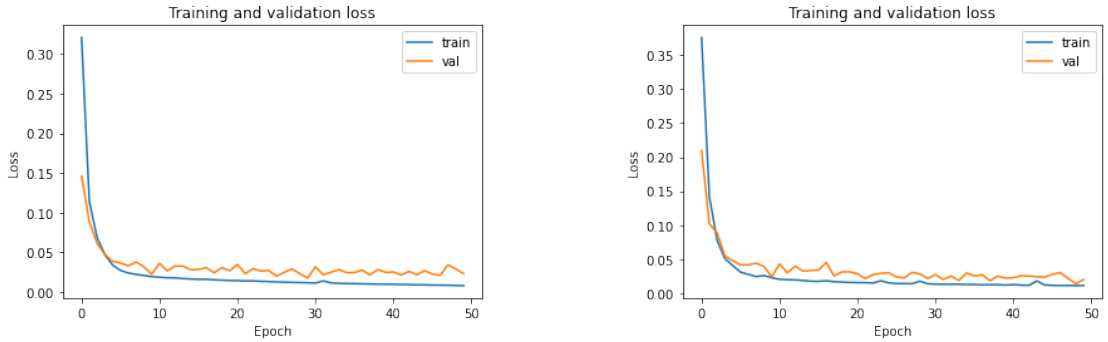


Figure 3.3: 3D architecture used where blue boxes represent feature maps and the number of channels is denoted above each feature map

Results

First, during the model training, we can observe the training and validation loss for 3D DeepVesselNet-UNet and 3D U-Net which are represented on the figures 3.4a and 3.4b respectively.
From these figures we see that the curves are relatively good since the loss decreases and training and validation curves are quite similar.



(a) 3D DVN

(b) 3D U-Net

Figure 3.4: Training and validation loss for 50 epochs

The number of convolutional parameters (only the parameters for the convolutional layers) in the networks is used to compare both architectures. In 3D U-Net,

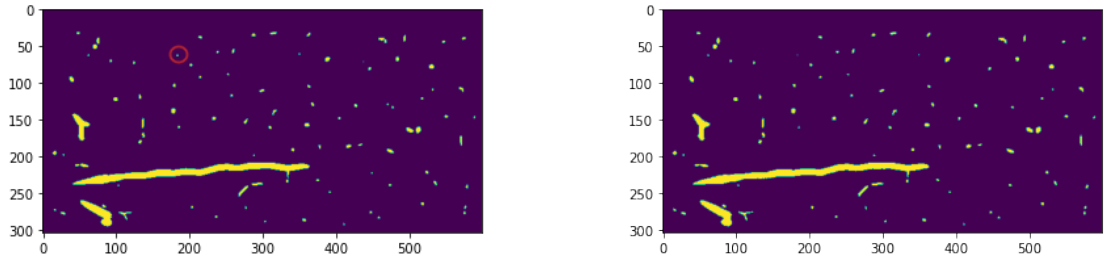
for the first convolution we have $(5 * 5 * 5 * 1 + 1) * 8 = 1,008$ parameters so in total 4,721,584 are used for our 3D U-Net.
For the first convolution using cross-hair filters, the number of convolutional parameters is 624 $3 * ((5 * 5 * 1 + 1) * 8)$ and in total 2,834,352 are used for 3D DVN, i.e. a gain of almost 1.67.
A comparison between the classical architecture and that using cross-hair filters can also be made with the average time over 10 epochs required to process 10 volumes of the synthetic dataset. For the classical 3D U-Net 443.92 seconds (with a standard deviation of 3.38) are needed and the 3D DVN takes in average 427.02 seconds (with a standard deviation of 4.77). In this case the cross-hair filters allow a speed gain of 1.04 to process 10 volumes.

Until now, the comparison tools (learning curves, execution time and number of parameters) were obtained during the training of the model. Subsequently, the metrics as introduced in section 2.3.3 and obtained during the model testing are analyzed (note that the F1-score in our case is equivalent to the dice score). The various results (averaged over the test set) obtained by the 3D DVN and 3D U-Net for 50 epochs are listed in the table 3.1. In view of all these metrics, both architectures have quite similar results with very slightly better performance for 3D U-Net, except for the recall. The specificity tells us that more than 99% of the background is predicted, the recall shows that 86% of blood vessels are detected. Thanks to the precision we know that 93% of elements predicted as vessels actually are and the dice score, which measures the similarity between the true mask and the predictions is 0.89.

	Specificity	Recall	Precision	Dice score
3D DVN	0.9988	0.8664	0.9313	0.8940
3D U-Net	0.9989	0.8648	0.9377	0.8953

Table 3.1: Average metrics for 3D DeepVesselNet-UNet and 3D U-Net for 50 epochs

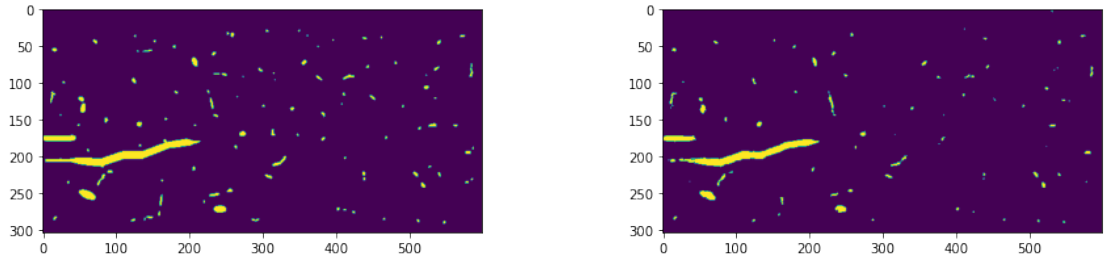
Model predictions can also be analyzed visually. Figure 3.5b is an example of one slice of the predicted mask obtained with the 3D U-Net and where the corresponding ground truth mask is illustrated on the figure 3.5a. Among the 20 predicted masks, it is this predicted mask which achieves the best performance in terms of dice score (0.9298 for 3D U-Net).
We can observe than both masks (3.5a and 3.5b) are quite similar with very few differences perceptible to the human eye (one voxel that is not predicted as a vessel with 3D U-Net is circled in red in the true mask).



(a) True mask (b) Predicted mask obtained with 3D U-Net

Figure 3.5: One slice of a ground truth mask of size 304×600 and the corresponding predicted mask obtained with 3D U-Net which achieves the best dice score

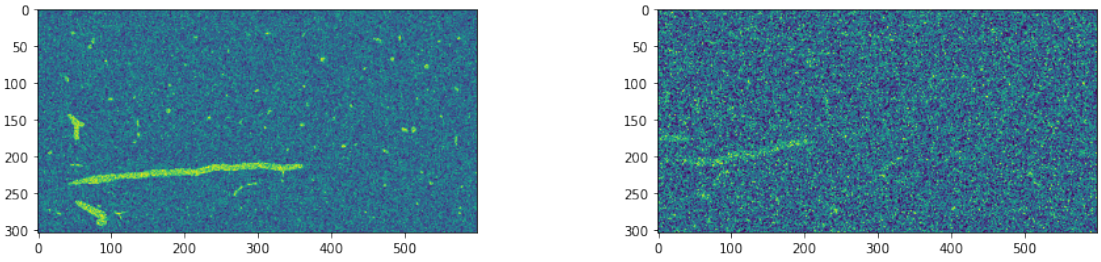
It is also interesting to observe the prediction which achieves the lower performance in terms of dice score (0.6649 for 3D U-Net) on the figure 3.6b and where the true mask is illustrated on the figure 3.6a. We see directly that the predictions are worse than previous ones, more voxels are less well predicted as being vessels.



(a) True mask (b) Predicted mask obtained with 3D U-Net

Figure 3.6: One slice of a ground truth mask of size 304×600 and the corresponding predicted mask obtained with 3D U-Net which achieves the lower dice score

By observing the input images of the figures 3.5 and 3.6 respectively on the figures 3.7a and 3.7b, we can easily understand why: the second image is more noisy.
The segmentation obtained with 3D DVN is not illustrated here since the differences with the one of 3D U-Net are quite unnoticeable but there are in the Appendices with also some additional results. Appendices also contain some additional results to other experiments.



(a) Input image with higher dice score

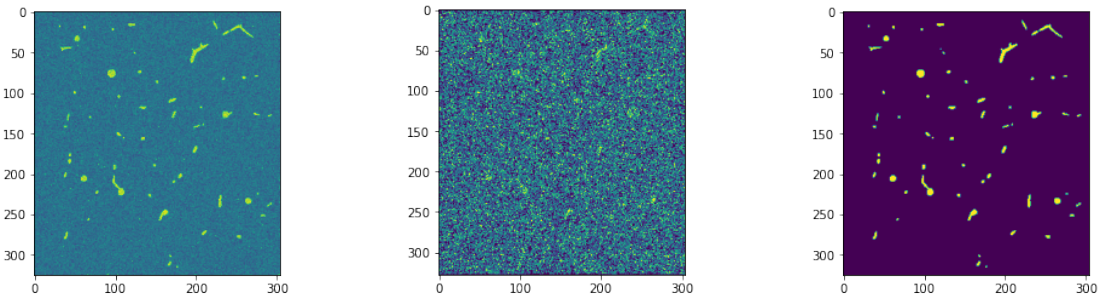
(b) Input image with lower dice score

Figure 3.7: One slice of the input image which achieves the best dice score and one slice of the input image which achieves the lower dice score of size 304×600

3.1.2 2D DeepVesselNet-UNet and 2D U-Net

Data pre-processing

For the 2D architectures, there is no extraction of patches but the volumes are sliced in order to have, for each volume, 600 2D images of size 328×304 which are also normalized with an additional Gaussian noise. Examples of an image without additional Gaussian noise, with Gaussian noise and the corresponding labeled image are illustrated on the figures 3.8a, 3.8b and 3.8c respectively.



(a) 2D input image

(b) 2D input image with Gaussian noise

(c) 2D true mask

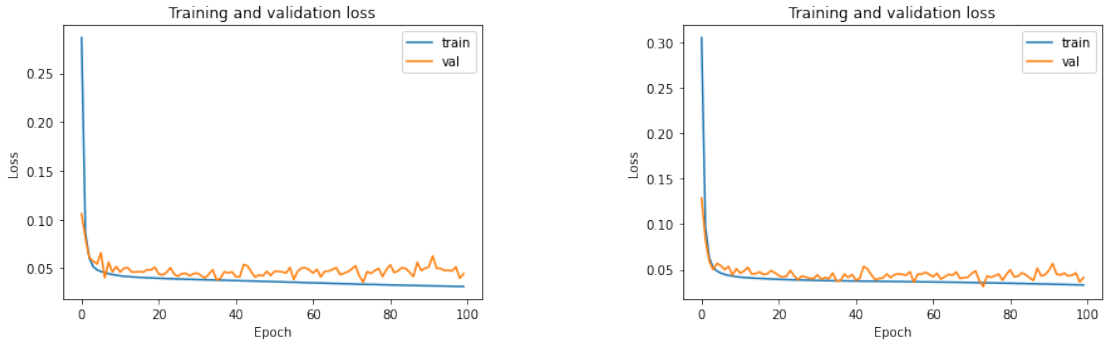
Figure 3.8: One slice of size 328×304 of an image without and with additional Gaussian noise and the corresponding labeled slice

Network configurations

The configurations are the same as for the 3D cases except that the kernel size is not anymore $5 \times 5 \times 5$ but 5×5 since it is in two dimensions.

Results

From the learning curves on the figures 3.9a and 3.9b, same observations as for the 3D cases can be made, namely they are relatively good.



(a) 2D DVN (b) 2D U-Net

Figure 3.9: Training and validation loss for 100 epochs

For the number of parameters, based on the equations 2.26 and 2.27, we found that 944,784 parameters are used with 2D U-Net and only 378,848 for 2D DVN, i.e. a gain of 2.49.
The average execution time for 10 volumes with 2D U-Net is 122.99 seconds with a standard deviation of 4.21 and with 2D DVN is 110.04 seconds with a standard deviation of 3.02 which allows a gain of 1.12 thanks to the cross-hair filters.

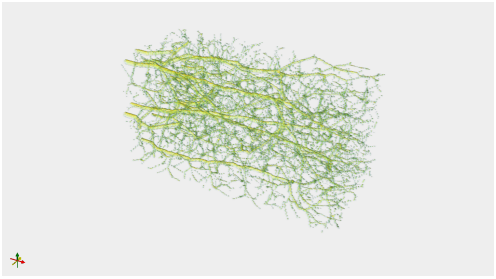
As for the 3D architectures, 2D U-Net has slightly higher metrics (see table 3.2) than 2D DVN except for the recall. However, for the 2D cases, the predictions are much worse, e.g. the dice score dropped to only 0.5 instead of 0.89 for the 3D cases. The probable reason for such a drop will be seen below with the visual analysis of the predictions. In addition, more epochs are needed, i.e. we stop after 100 epochs instead of 50.

	Specificity	Recall	Precision	Dice score
2D DVN	0.9970	0.3991	0.7111	0.5064
2D U-Net	0.9979	0.3884	0.7709	0.5103

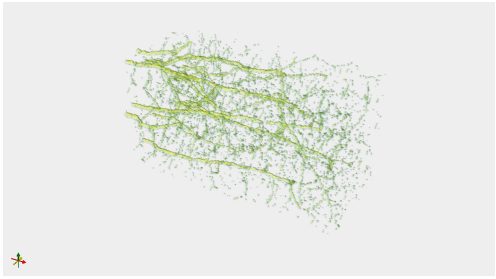
Table 3.2: Average metrics for 2D DeepVesselNet-UNet and 2D U-Net for 100 epochs

Less good performance are also observed with the predicted mask which achieves the best dice score and one of its slice on the figures 3.10b and 3.10d. These

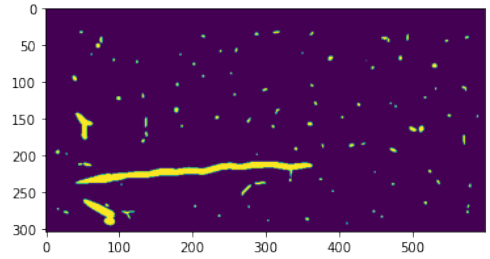
predictions obtained with 2D U-Net are further from true masks (figures 3.10a and 3.10c).
Thanks to these images we can think that the reason for this decrease in performance is the Gaussian noise added to the input image. Indeed, sometimes the information present in a single slice is not sufficient as can be seen in the figure 3.10d with the long vessel which is sometimes broken. But with the use of 3D patches, the model can assume that there is a continuity of the vessels which allows not to have fissures.



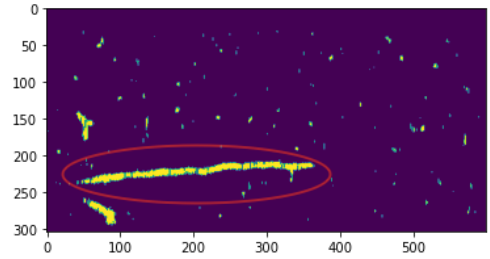
(a) True mask



(b) Predicted mask obtained with 2D U-Net



(c) Slice of the true mask



(d) Slice of the predicted mask obtained with 2D U-Net

Figure 3.10: One ground truth mask of size $325 \times 304 \times 600$, the corresponding predicted mask obtained with 2D U-Net and corresponding slices of size 304×600

3.1.3 Results Analysis

What emerges from these experiments is that in 3D as well as in 2D, U-Net obtains slightly better segmentation results than DeepVesselNet-UNet however the use of cross-hair filters allows gains of time and memory, specially in 2D. In addition, 2D cases have difficulty in achieving the performance of 3D cases due to the noise even if the number of epochs is increased. Hence, to have good performance with this dataset, the interest of using 3D information rather than 2D even if the latter saves time and memory.

It may therefore be preferable for these synthetic dataset to use 3D DeepVesselNet-UNet, since in 3D, cross-hair filters only induce a very slight decrease in the accuracy of the prediction and 2D architectures struggle to obtain suitable results.

3.2 Application on real data

3.2.1 Trial and error 3D DeepVesselNet-UNet and 3D U-Net

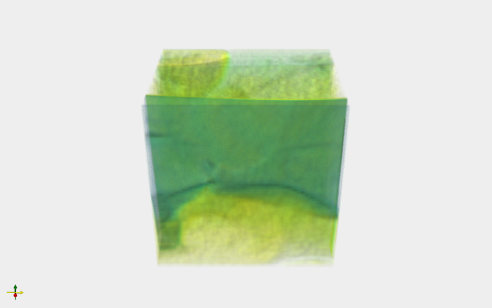
In a first time all parameter choices were made by trial and error and are thus not optimal which clearly illustrates one of the problems presented in section 1.4, namely human designs often lead to sub-optimal results. Moreover the choices are also influenced by the execution time and the GPU memory which also limit performance.

Data pre-processing

The images are standardized, i.e. the voxel values are scaled to have a zero mean and an unit variance by subtracting by the mean and dividing by the standard deviation.
For execution time reasons only 120 volumes are used (volumes with a size between $512 \times 512 \times 40$ and $512 \times 512 \times 52$) and their size is changed by zero padding or by cropping to consider only volumes of size $512 \times 512 \times 50$. Then 64 patches of size $64 \times 64 \times 64$ are extracted from each volume. An example is shown on the figures 3.11a and 3.11b and the figures 3.11c and 3.11d illustrate one slice of these patches. The training set is composed of 100 volumes, i.e. 6400 patches, the validation set represents 20% of the training set and the testing set has 20 volumes.

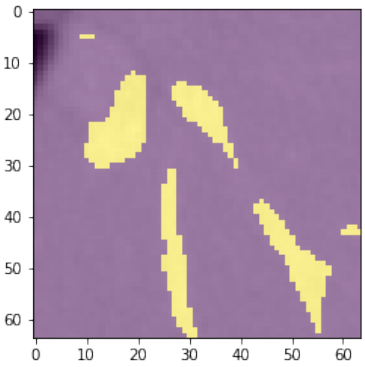
Network configurations

Optimization algorithm : As for the synthetic dataset, Adam is the optimization algorithm used. As said in chapter 1, the choice of the learning rate influences the speed of convergence. Here an initial learning rate of 0.0001 with a simple decay of 0.0001 (see section 2.3.2) is used instead of a constant learning rate due to the complexity of the dataset.
Loss function : Since the classes are extremely unbalanced the class-balancing cross-entropy is used instead of the cross-entropy.
Activation function : The ReLU activation function is used for the hidden layers and for the output layers it is the softmax function.

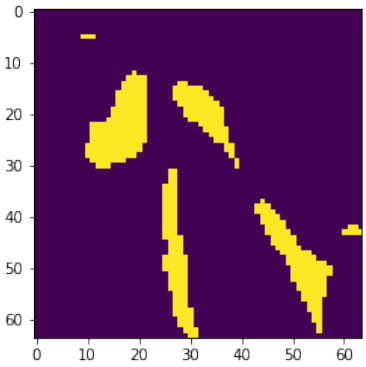


(a) Standardized input image

(b) True mask



(c) Slice of the standardized input image with the true mask



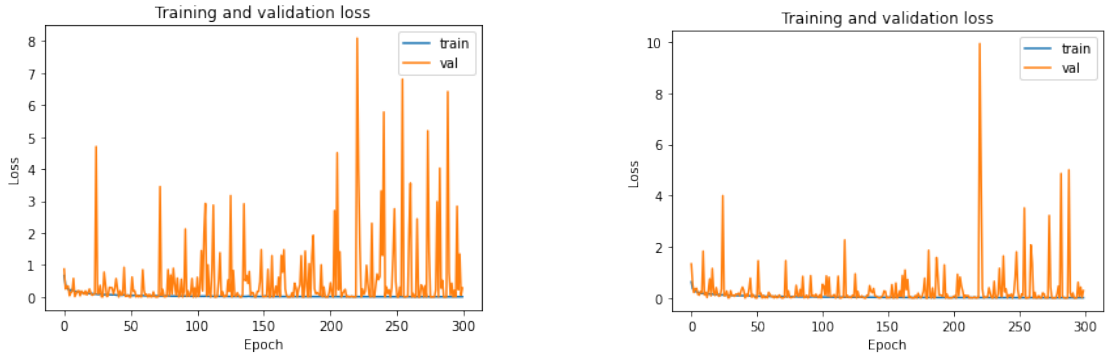
(d) Slice of the true mask

Figure 3.11: One patch of size $64 \times 64 \times 64$ of a standardized input image, its corresponding labeled patch and corresponding slices of size 64×64

Moreover, 5 levels are used for the network depth. The interest of using deep networks is that they allow to have good generalization capacities unlike wide but shallow networks which are good for memorizing but not for generalizing. The $3 \times 3 \times 3$ kernel size is chosen in order to reduce computational costs and execution time since 5 levels are used. And the initial number of features is 16 and the batch size used is 8.

Results

The graphs representing the evolution of the loss on the figures 3.12a and 3.12b have a very noisy validation loss mainly for 3D DVN. Despite many different parameters tested, this noise was still present and may be due in large part to the fact that only 100 volumes are used during the training which does not allow the network to have good generalization capacities.



(a) 3D DVN

(b) 3D U-Net

Figure 3.12: Training and validation loss for 300 epochs

The fact that the kernel size is only of $3 \times 3 \times 3$ does not allow to benefit from the advantages of cross-hair filters. This is explained by the fact that the number of multiplications and additions necessary to make a convolution for each voxel (see section 2.1) is the same for classical convolutions and for those using cross-hair filters with a $3 \times 3 \times 3$ kernel. Indeed for each voxel, 3D U-Net uses $3 * 3 * 3 = 27$ multiplications and $3 * 3 * 3 - 1 = 26$ additions and 3D DVN uses $3 * 3 + 3 * 3 + 3 * 3 = 27$ multiplications and $3 * 3 + 3 * 3 + 3 * 3 - 1 = 26$ additions. In addition, the number of parameters is greater for 3D DVN, 16,472,160 against only 16,467,264 for 3D U-Net. The execution time to process 10 volumes is then impacted, 102.34 seconds with a standard deviation of 3.21 with 3D DVN and only 69.49 seconds with a standard deviation of 4.15 with 3D U-Net.

Trial-error finding of parameters result in not really good performance (see table 3.3). The fact that 3D DVN uses more parameters and time allows it to have better predictive performance than 3D U-Net except for the recall which means that less blood vessels are detected (66% against 77%). But 38% of the predicted elements as vessels actually are against only 25% for 3D U-Net.

	Specificity	Recall	Precision	Dice score
3D DVN	0.9983	0.6563	0.3773	0.4654
3D U-Net	0.9962	0.7705	0.2505	0.3658

Table 3.3: Average metrics for 3D DVN and 3D U-Net for 300 epochs

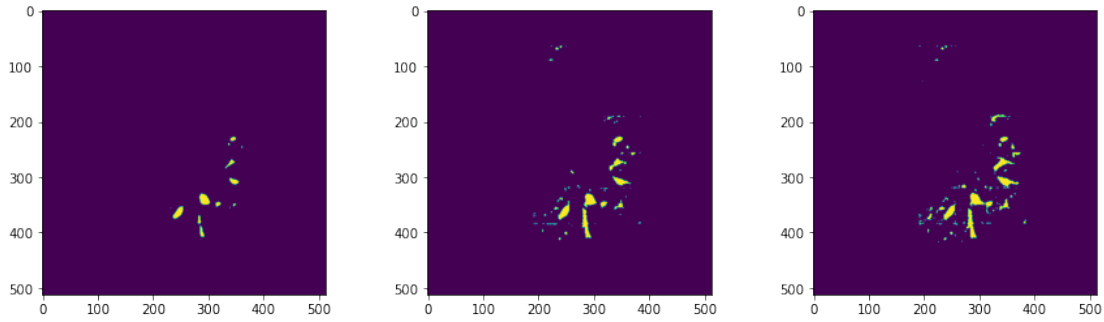
Weak performance can be observed on the prediction which achieves the best dice score (0.56 with 3D DVN). For 3D DVN it is represented on the figure 3.13b and for 3D U-Net on the figure 3.13c where the true mask is illustrated on the figure

3.13a. The figures 3.13d, 3.13e and 3.13f show one of their slices.
Weaker performance is also seen on the predictions which achieves the lower dice score (0.35 with 3D DVN). For 3D DVN it is illustrated on the figure 3.14b and for 3D U-Net on the figure 3.14c where the actual labeled image is represented on the figure 3.14a. The figures 3.14d, 3.14e and 3.14f represent one of their slices.
For both we can observe that too much voxels are predicted as vessels mostly with 3D U-Net. But better dice score for the first one can be explained by the fact that the vessels seems larger and that the class imbalance is slightly less important (0.26% of the total voxels represent vessels against 0.21%).
However it is interesting to notice that the networks seem to predict tree-like networks. We can therefore wonder if these other networks are types of vessels not labeled by the annotator. This can be observed in the predictions of all other experiments and will be discussed in section 3.3.

(a) True mask

(b) Predicted mask obtained with 3D DVN

(c) Predicted mask obtained with 3D U-Net



(d) Slice of true mask

(e) Slice of the predicted mask obtained with 3D DVN

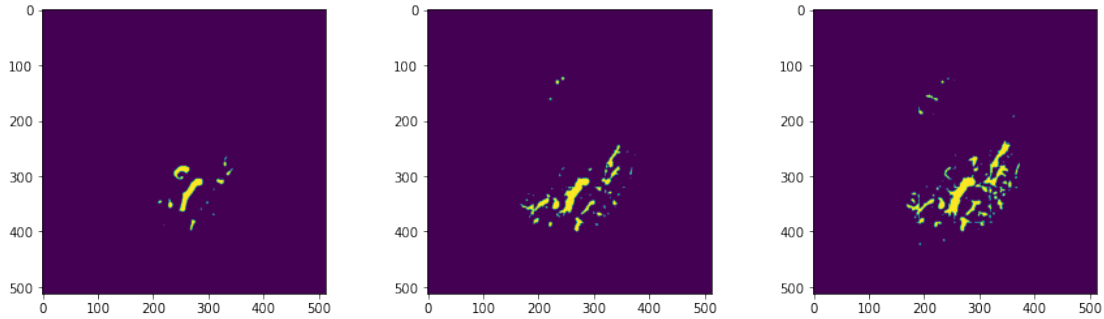
(f) Slice of the predicted mask obtained with 3D U-Net

Figure 3.13: One ground truth mask of size $512 \times 512 \times 40$, the corresponding predicted masks which achieves the best dice score obtained with 3D DVN and 3D U-Net and corresponding slices of size 512×512

(a) True mask

(b) Predicted mask obtained with 3D DVN

(c) Predicted mask obtained with 3D U-Net



(d) Slice of the true mask

(e) Slice of the predicted mask obtained with 3D DVN

(f) Slice of the predicted mask obtained with 3D U-Net

Figure 3.14: One ground truth mask of size $512 \times 512 \times 40$, the corresponding predicted masks which achieves the lower dice score obtained with 3D DVN and 3D U-Net and corresponding slices of size 512×512

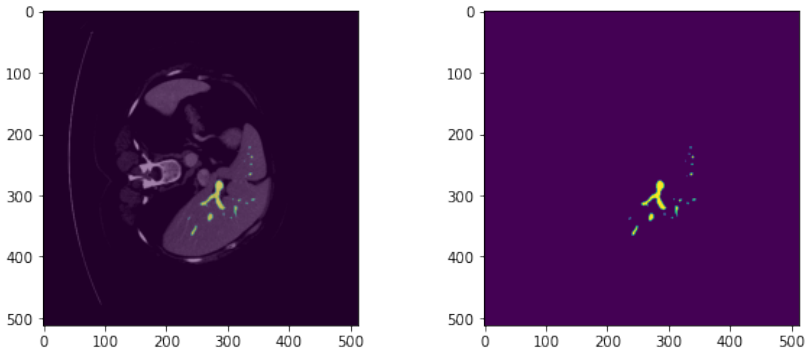
3.2.2 Trial and error 2D DeepVesselNet-UNet and 2D U-Net

Data pre-processing

The volumes are sliced to have, for each one, 50 2D images of size 512×512 as it can be seen on the figures 3.15a for a standardized image and 3.15b for the corresponding labels.

Network configurations

The configurations are the same as for the 3D architectures except for the kernels which are in 2D.



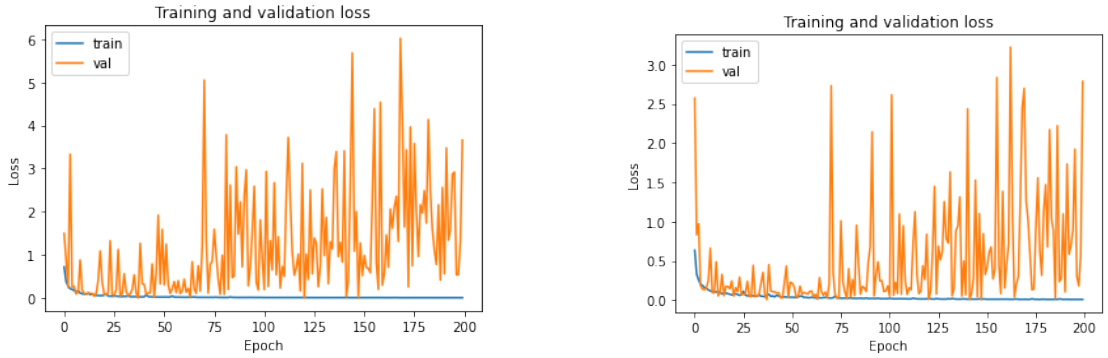
(a) 2D standardized input image with its true mask

(b) 2D true mask

Figure 3.15: One slice of size 512×512 of a standardized input image and the corresponding labeled slice

Results

As for the 3D cases the validation loss is noisy and moreover after only a hundred epochs it seems that there is overfitting since in addition to oscillations, minima of the validation loss move further and further away from the training loss curve. This can be observed on the figures 3.16a for 2D DVN and 3.16b for 2D U-Net.



(a) 2D DVN

(b) 2D U-Net

Figure 3.16: Training and validation loss for 200 epochs

Unlike 3D cases, there are less parameters used for 2D DVN (3,663,744) than for 2D U-Net (5,490,720) and less multiplications and additions for the convolution of one voxel for 2D DVN. Indeed, for 2D DVN $3 * 1 + 3 * 1 = 6$ multiplications and $3 * 1 + 3 * 1 - 1 = 5$ additions are used and for 2D U-Net $3 * 3 = 9$ multiplications

and $3 * 3 - 1 = 8$ additions are used. Despite this, the execution time to process 10 volumes is greater for 2D DVN, 39.88 seconds with a standard deviation of 2.50 against 32.04 seconds with a standard deviation of 3.08 for 2D U-Net. Therefore, even if less parameters, multiplications and additions are used for 2D DVN, it takes more time to process the volumes than 2D U-Net which leads us to think that the standard 2D convolution is a better implementation: performing a convolution with a 3×3 kernel is faster than performing two convolutions with a 3×1 kernel and making a sum of the results.

To have better generalization properties and avoid overfitting, the selected models are taken after 100 epochs and the results obtained for the different metrics are listed in the table 3.4. Same observations can be made as for the 3D cases, namely 2D DVN has better results than 2D U-Net. It is interesting to note that the results are higher than those obtained for the 3D configurations, e.g. 2D DVN has a dice score of 0.4984 and 3D DVN has a dice score of 0.4654 (see table 3.3). The probable reason is explained in the next section (Comparison between 2D and 3D architectures).

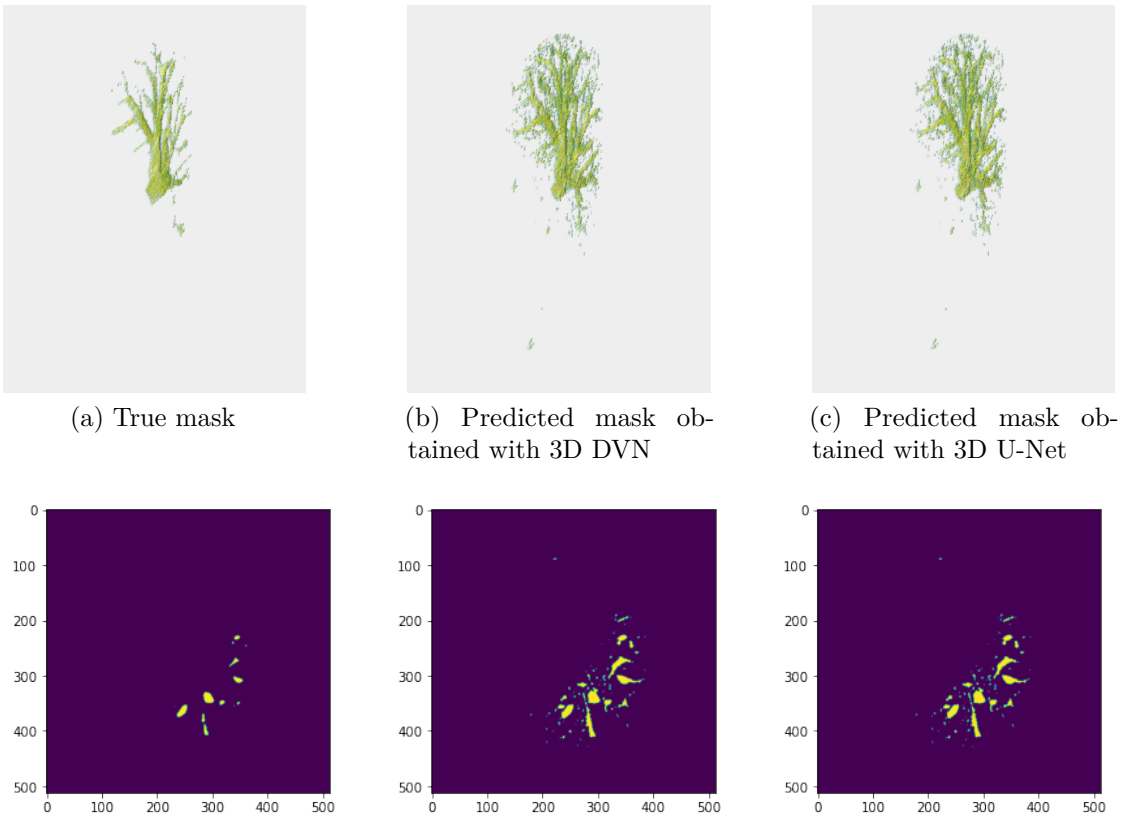
	Specificity	Recall	Precision	Dice score
2D DVN	0.9985	0.6661	0.4185	0.4984
2D U-Net	0.9970	0.8005	0.3084	0.4320

Table 3.4: Average metrics for 2D DVN and 2D U-Net for 100 epochs

In accordance with the results obtained for the different metrics, visually the predictions are better than those of 3D architectures as we can see on the figures 3.17b and 3.17e for 2D DVN and on the figures 3.17c and 3.17f for 2D U-Net. Based on the figures 3.17a and 3.17d, representing the real labels in 3D and 2D respectively, we can see that less background is detected as being vessels compared to 3D cases.

3.2.3 Comparison between 2D and 3D architectures

What we can observe in these experiments is that 2D architectures get better performance on all points: less memory used, faster data processing, need less epochs to converge, better metrics and visually a segmentation corresponding more to reality. The difference lies in the size of the input data. For 2D networks the images have a size of 512×512 and for 3D networks the patches used are of size $64 \times 64 \times 64$. This therefore leads us to think that using patches of this size is not optimal and thus too small for this dataset and with the network configurations described previously.



(a) True mask

(b) Predicted mask obtained with 3D DVN

(c) Predicted mask obtained with 3D U-Net

(d) Slice of the true mask

(e) Slice of the predicted mask obtained with 3D DVN

(f) Slice of the predicted mask obtained with 3D U-Net

Figure 3.17: One ground truth mask of size $512 \times 512 \times 40$, the corresponding predicted masks which achieves the best dice score obtained with 3D DVN and 3D U-Net and corresponding slices of size 512×512

3.2.4 5x5x5 kernel 3D DeepVesselNet-UNet and 3D U-Net

Since best models obtained by trial and error in terms of accuracy use kernels of size $3 \times 3 \times 3$, the advantages of cross-hair filters, namely saving time and memory, cannot be observed. The goal of this experiment, where the size of the kernels is increased to $5 \times 5 \times 5$, is to verify if for this type of dataset we still have these benefits while avoiding a drop in the accuracy of predictions.

Data pre-processing

The data pre-processing is similar to the one of the first 3D experiments.

Network configurations

What changes compared to the first 3D experiments is the size of the kernels which are not anymore $3 \times 3 \times 3$ but $5 \times 5 \times 5$. Since the kernel size is increased, the number of levels is decreased to 4 out of concern for execution time.

Results

This configuration leads to a drop in performance both in terms of the quality of predictions and in terms of time. Indeed, on the comparative table 3.5 between the network configuration of 3D DVN of the previous experiment with this new network configuration for 300 epochs, we can mainly observe that the previous network has a higher dice score, better detects blood vessels and is faster.

	Specificity	Recall	Precision	Dice score	Time
3D DVN with 5 levels, $3 \times 3 \times 3$ kernel size	0.9983	0.6563	0.3773	0.4654	102.34
3D DVN with 4 levels, $5 \times 5 \times 5$ kernel size	0.9986	0.5057	0.3649	0.4168	158.93
3D U-Net with 4 levels, $5 \times 5 \times 5$ kernel size	0.9961	0.4949	0.2912	0.3407	169.58

Table 3.5: Average metrics and execution time for 3D DVN with 5 levels and $3 \times 3 \times 3$ kernel size and 3D DVN and 3D U-Net with 4 levels and $5 \times 5 \times 5$ kernel size for 300 epochs

But between 3D DVN and 3D U-Net with a $5 \times 5 \times 5$ kernel size there is a decrease in the number of parameters thanks to the use of cross-hair filters. The number of parameters with 3D U-Net is 18,883,168 and with 3D DVN this number drops to 11,332,704, i.e. a gain of 1.66. There is also a gain of time since to process 10 volumes 3D U-Net takes 169.58 seconds with a standard deviation of 5.85 and 3D DVN takes 158.93 seconds with a standard deviation of 6.92, i.e. a gain of 1.07. Thus for this real dataset, the cross-hair filters also save time and money when the kernel size is not too small, i.e. $3 \times 3 \times 3$ while allowing better predictions than 3D U-Net as it can be seen on the table 3.5.
We can therefore conclude that the augmentation of the kernel size increases the gap between the computational complexity of classical 3D convolution and convolution using cross-hair filters.

3.2.5 nnU-Net based 3D DeepVesselNet-UNet and 3D U-Net

These experiments aim to increase performance by using a maximum of parameters found by the self-configuring method, nnU-Net (see chapter 1) for the hepatic vessel segmentation task and which came first in the ranking of the Medical Segmentation Decathlon challenge (`http://medicaldecathlon.com/results.html`) for this dataset. However due to GPU memory limits and execution time not all parameters could be considered.

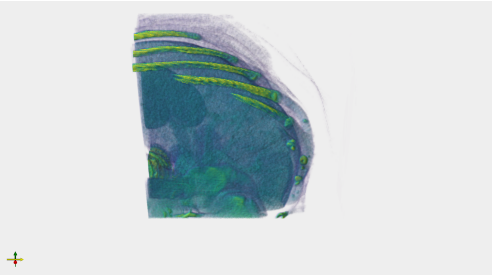
Data pre-processing

The patch size is modified to $192 \times 192 \times 64$ so for one volume 16 patches are extracted. nnU-Net also found that a global normalization based on foreground voxel intensities for all training cases is the best way to process this data, i.e. data are first clipped to $[-3,\ 243]$, then reduced by 104.37 and finally divided by 52.62. Following this global normalization, the intensity of the voxels is between -2.05 and 2.64. An example of the 3D pre-processed data is shown on figures 3.18a and 3.18b and one of their slices is represented on the figures 3.18c and 3.18d.

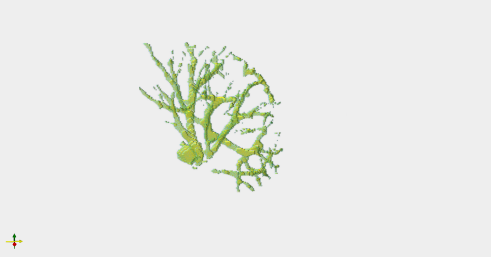
Network configurations

Optimization function : SGD is used for the optimization function. For the initial learning rate a high one of 0.01 with a large Nesterov momentum of 0.99 is chosen.
Moreover, during the training the learning rate is reduced by using the 'poly' learning rate schedule. The figure 3.19 shows the almost linear evolution of the learning rate over the epochs for 3D DVN.
Loss function : The loss function does not change, i.e. the class-balancing cross entropy is still used.
Activation function : The activation function used for hidden layers is the rectified linear unit which is briefly introduced in the 1.1.2 section. ReLu mainly has the advantage of reducing the problem of the gradient to vanish.
For the output layer it is always the softmax function which is applied.

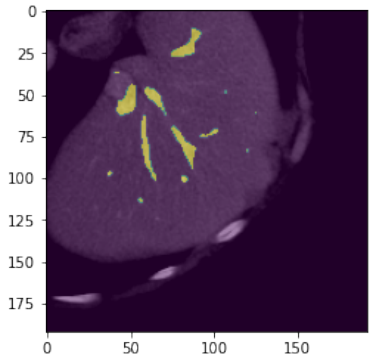
In addition, for execution time reasons 4 levels are used which is not the optimal number of levels found by nnU-Net, i.e. 6 levels. As nnU-Net, the $3 \times 3 \times 3$ kernel size is chosen and the batch size is 2. The number of initial features is 8 since it cannot be increased due to GPU memory limits.



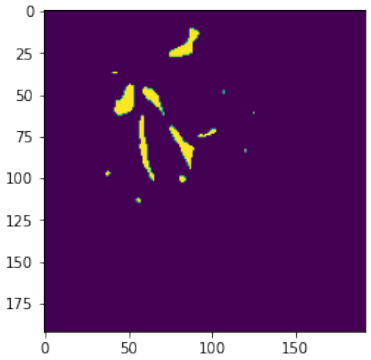
(a) Normalized input image



(b) True mask



(c) Slice of the standardized input image with its true mask



(d) Slice of the true mask

Figure 3.18: One patch of size $192 \times 192 \times 64$ of a normalized input image, its corresponding labeled patch and corresponding slices of size 192×192

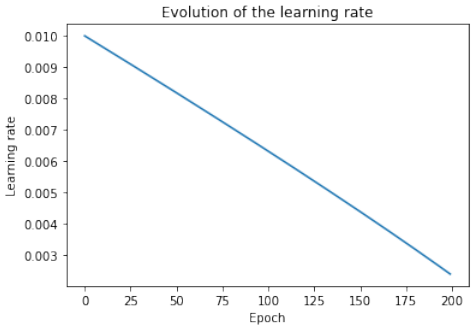
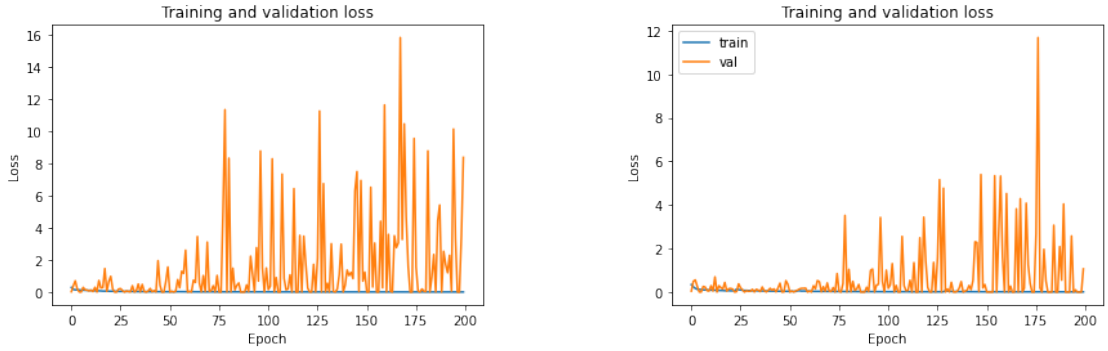


Figure 3.19: Evolution of the learning rate for 3D DVN

Results

For this configuration of the network the loss curves (see figure 3.20a for 3D DVN and 3.20b for 3D U-Net) oscillates a lot with higher peaks than the first configuration.



(a) 3D DVN

(b) 3D U-Net

Figure 3.20: Training and validation loss for 200 epochs

Since we have kernels of size $3 \times 3 \times 3$, the time to process 10 volumes with 3D DVN is higher, i.e. 216.98 seconds with a standard deviation of 8.26 against 102.71 seconds with a standard deviation of 4.73 for 3D U-Net, as well as the number of parameters, 1,021,488 against 1,020,320.

As for the first experiments, but where metrics were lower, 3D DVN has better performance than 3D U-Net (see table 3.6).

	Specificity	Recall	Precision	Dice score
3D DVN	0.9986	0.6376	0.4434	0.5033
3D U-Net	0.9981	0.6731	0.3961	0.4665

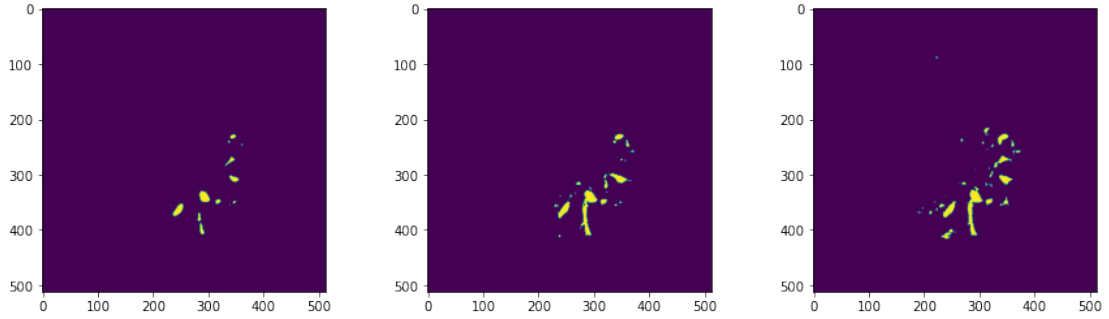
Table 3.6: Average metrics for 3D DVN and 3D U-Net for 200 epochs

Compared to previous experiments, visually segmentation of blood vessels are also better. Indeed less background voxels are detected as being vessels, as we can see on the figures 3.21b and 3.21e for 3D DVN and the figures 3.21c and 3.21f for 3D U-Net, where the figures 3.21a and 3.21d are the actual labeled images.

(a) True mask

(b) Predicted mask obtained with 3D DVN

(c) Predicted mask obtained with 3D U-Net



(d) Slice of the true mask

(e) Slice of the predicted mask obtained with 3D DVN

(f) Slice of the predicted mask obtained with 3D U-Net

Figure 3.21: One ground truth mask of size $512 \times 512 \times 40$, the corresponding predicted masks which achieves the best dice score obtained with 3D DVN and 3D U-Net and corresponding slices of size 512×512

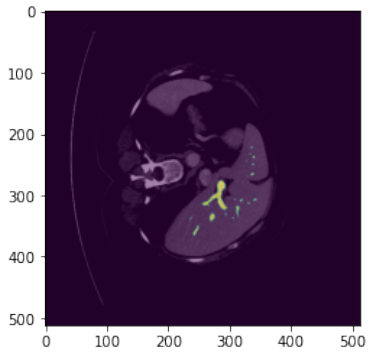
3.2.6 nnU-Net based 2D DeepVesselNet-UNet and 2D U-Net

Data pre-processing

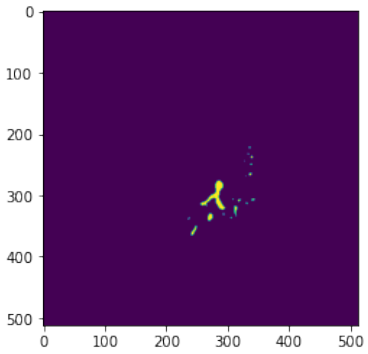
Each volume is sliced to have 50 2D images of size 512×512 as it can be seen on the figures 3.22a and 3.22b.

Network configurations

The configurations are the same as for the 3D architectures except for the kernels which are in 2D.



(a) 2D normalized input image with its true mask

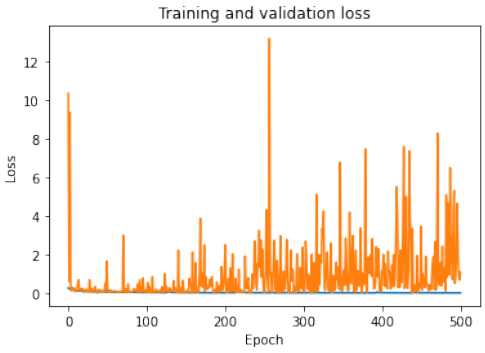


(b) 2D true mask

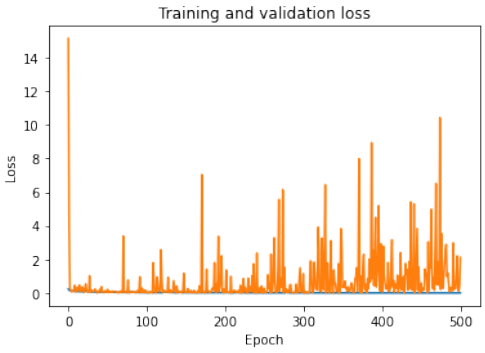
Figure 3.22: One slice of size 512×512 of a normalized input image with its true mask and the corresponding true mask slice

Results

Loss curves on the figures 3.23a and 3.23b are also noisy in 2D.



(a) 2D DVN



(b) 2D U-Net

Figure 3.23: Training and validation loss for 500 epochs

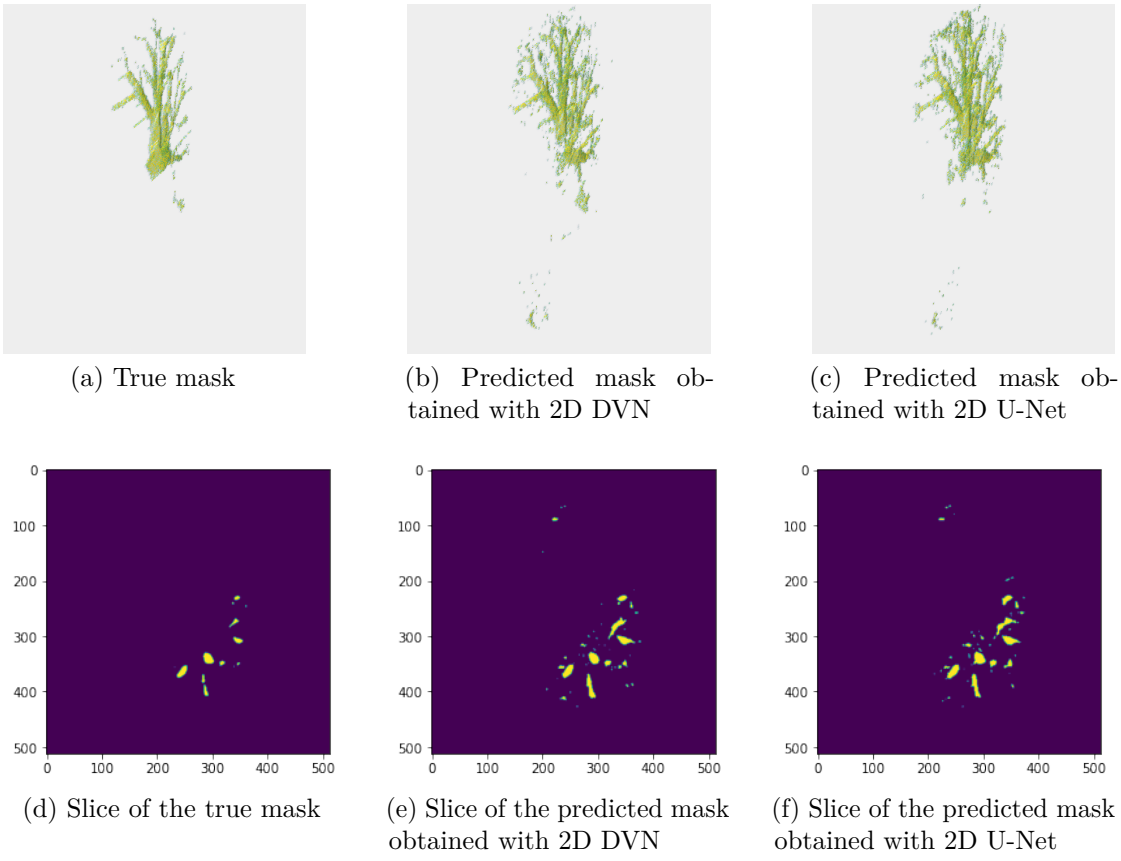
Regarding the execution time, 2D U-Net takes 14.47 seconds with a standard deviation of 1.54 to process 10 volumes and 2D DVN takes 18.05 seconds with a standard deviation of 1.35. For the number of parameters, 340,496 are used with 2D U-Net and only 227,776 for 2D DVN. As for the first network configuration, 2D DVN needs more time but uses less parameters.

In terms of metrics (see table 3.7), results are slightly better for 2D DVN. And unlike the first configuration, 2D architectures do not surpass those in 3D, e.g. for 3D DVN the dice score is 0.5033 and for 2D DVN the dice score is 0.4521. Increasing the size of the patches for 3D architectures ($192 \times 192 \times 64$ instead of $64 \times 64 \times 64$) therefore allows better segmentation results than using 2D ones.

	Specificity	Recall	Precision	Dice score
2D DVN	0.9976	0.7562	0.3368	0.4521
2D U-Net	0.9974	0.7789	0.3280	0.4480

Table 3.7: Average metrics for 2D DVN and 2D U-Net for 500 epochs

Same observations can be done on the figures 3.24b, 3.24c, 3.24e and 3.24f which illustrate the predictions of the true masks (figures 3.24a and 3.24d).



(a) True mask

(b) Predicted mask obtained with 2D DVN

(c) Predicted mask obtained with 2D U-Net

(d) Slice of the true mask

(e) Slice of the predicted mask obtained with 2D DVN

(f) Slice of the predicted mask obtained with 2D U-Net

Figure 3.24: One true mask of size $512 \times 512 \times 40$, the corresponding predicted masks which achieves the best dice score obtained with 2D DVN and 2D U-Net and corresponding slices of size 512×512

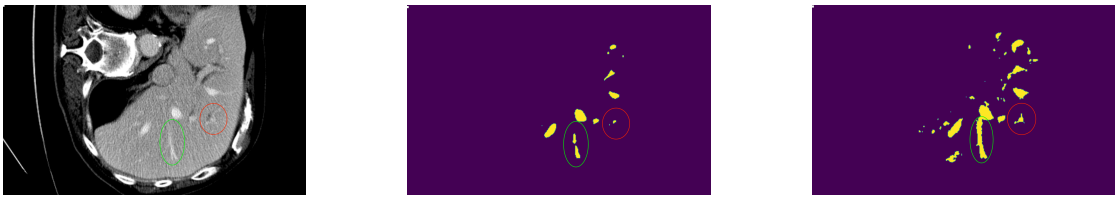
3.3 Discussion and limitations

A first general observation is that compared to synthetic data, the real data is much more complex, therefore, with a search for the parameters by trial and error, it is difficult to have good performance. Even with a part of the parameters found by nnU-Net we are still far from their performance obtained for the Medical Segmentation Decathlon (`http://medicaldecathlon.com/results.html`), i.e. 0.63 in terms of dice score. This is partly due to the fact that some parameters were not deliberately applied due to the required execution time (e.g. the number of data used or the depth of the network) and others could not be used due to the limitations of the GPU (e.g. increasing the initial number of features).
In addition, performing data augmentation (see chapter 1), as done in nnU-Net, could greatly improve the performance of the models but this was not carried out within the framework of this master thesis.
Moreover, doing a hold-out cross-validation is not optimal. It would be more interesting to do a K-fold cross-validation, as explained in chapter 1, in order to avoid inaccurate results due to fixed training and validation sets and a small fixed test set.
Thus it would be interesting to train the network with all the data from the hepatic vessel dataset, to use data augmentation, to increase the depth of the network and the number of initial features and to use a K-fold cross-validation in order to have better learning curves and better predictions of the test set thanks to the increase in generalization capacities.

Another observation can be made on the performance obtained with DeepVesselNet-UNet and U-Net. For synthetic data, where networks use $5 \times 5 \times 5$ size kernels and therefore benefit from the speed and memory gains of cross-hair filters, performance of U-Net is very slightly better than that of DVN for the same number of epochs. However for the real data, where kernels have a size of $3 \times 3 \times 3$ and thus networks need more time to process data and use more parameters in 3D, it is the opposite effect, DVN outperforms U-Net.

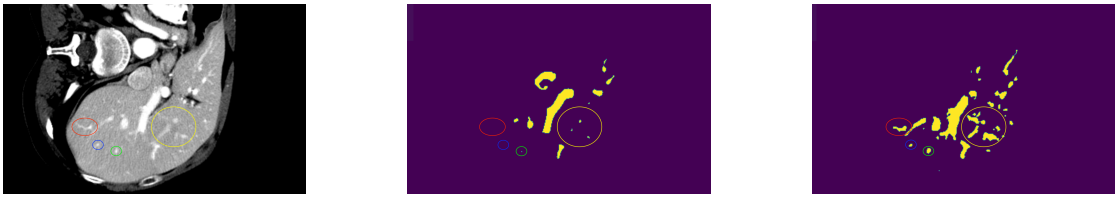
A very interesting thing that has already been noted is the fact that for real dataset, although the predictions obtained are not very good (the best dice score is 0.5), they are tree-like networks. We can therefore ask ourselves if these networks have a meaning: do they represent capillaries (smallest and thinnest blood vessels) or a type of vessel that have not been annotated, so do the manual annotations lack precision or is it simply the models that are not precise enough? To try to answer it, we analyze some slices of the input images (figures 3.25a and 3.26a) and compare them with their real mask (figures 3.25b and 3.26b) as well as with predictions (figures 3.25c and 3.26c), where some differences are circled. Based on

these figures we can see that the labeling omits many elements which would seem to be vessels, especially for the figures 3.26. In addition, false detections seem to correspond to similar areas to those labeled as being vessels, which could lead us to think that the low prediction performance is in part due to the lack of precision in manual labeling. However, this cannot be affirmed with certainty due to the limited knowledge in the field of manual segmentation of blood vessels.



(a) Image (b) True mask (c) Predicted mask

Figure 3.25: Comparison between one slice of an input image, its true mask and the corresponding predicted mask obtained with 3D U-Net



(a) Image (b) True mask (c) Predicted mask

Figure 3.26: Comparison between one slice of another input image, its true mask and the corresponding predicted mask obtained with 3D U-Net

An additional finding for the real dataset, is that 2D models found by trial and error achieve similar results than 3D models using the parameters of nnU-Net, i.e. 2D DVN has a dice score of 0.4984 and 3D DVN a dice score of 0.5033. However, comparing them directly is difficult since the capacities of the latter could not be exploited to the maximum. Indeed, only 4 levels are used against 5 for 2D and 8 initial features are considered against 16 for 2D. It would therefore be interesting to see if the performance achieved with the 3D model, for a network depth and an initial number of features similar to the 2D model, justifies its increase in terms of costs and time compared to 2D architectures.

Finally, it is not easy to draw general conclusions to know which model is more efficient since the results vary greatly from one dataset to another and from one configuration to another. However, for networks using filter sizes greater than 3×3 or $3 \times 3 \times 3$, using cross-hair filters allow to save time and memory. And for the real dataset DVN outperforms U-Net.

Conclusion

In this master thesis we have performed segmentations of blood vessels in an attempt to improve speed and memory consumption while achieving suitable prediction accuracy. This was done in order to respond to the main challenges encountered with the processing of large 3D datasets, namely computational cost and speed.

To do this we first analyzed theoretically the cross-hair filters which are composed of three intersecting 2D filters. The purpose of these filters is to preserve the 3D information of the volumes while dealing with the problems of memory and speed of standard 3D networks. In theory, this method seems very promising. This is why the implementation of these cross-hair filters in convolutional layers has been carried out on the well-known U-Net architecture, which is considered to be one of the main tools for the segmentation tasks in medical imaging.

With the results obtained for the synthetic data, we have seen that it was preferable to use cross-hair filters with a 3D architecture. Indeed, in 3D cross-hair filters only induce a very slight decrease in the accuracy of the predictions but save time and memory. And the 2D architectures struggle to obtain suitable results because of the noise present in the input images.

The experiments carried out for the real data showed in a first time that if the patches extracted from the 3D volumes are too small, poorer predictions are obtained compared to the use of 2D slices. Moreover, for the architectures having kernels of size 3×3 or $3 \times 3 \times 3$, the cross-hair filters compared to classical convolutions obtain lower performance in terms of speed and for the 3D cases also in terms of memory. This therefore leads us to think that the implementation of cross-hair filters in 2D is less efficient than standard 2D convolutions because, despite using less parameters and operations for the convolutions, they do not allow a gain in speed.
But surprisingly the prediction accuracy is better with cross-hair filters than with classic convolutions both in 3D and 2D.
However, the dice score obtained with the configuration based on the optimal results

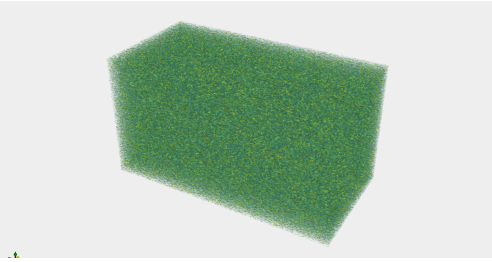
found by nnU-Net does not reach the one obtained with the nnU-Net method itself. This explains partly due to the GPU limitations and some sub-optimal choices that can be improved such as using data augmentation.

As part of this master thesis we started with U-Net architectures, 2D and 3D, using classic convolutions and cross-hair filters and we tried to improve them by playing on different parameters in order to obtain good performance on a specific dataset. But it would be interesting to implement cross-hair filters on 2D and 3D architectures already achieving good results for a particular dataset, such as nnU-Net, in order to analyze their impact on the performance of such architectures and perhaps be able to draw more general conclusions.

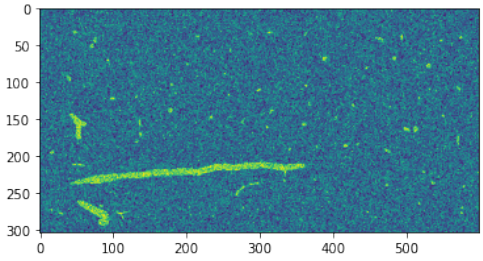
We can conclude by saying that we are only at the beginnings of a new era in deep learning applications in the medical field. Deep learning is therefore only at the start of its potential, but progress can go very quickly and tremendous quantum leaps can be accomplished to allow the diagnosis and treatment of many diseases. Particular attention will be paid to future studies on this subject.

Appendix A

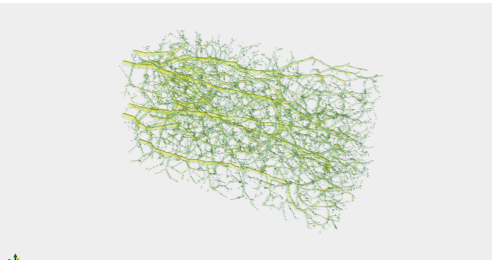
Some results for synthetic dataset



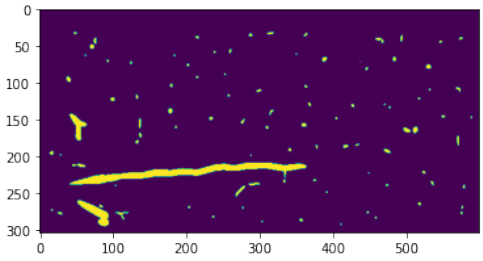
(a) Input image



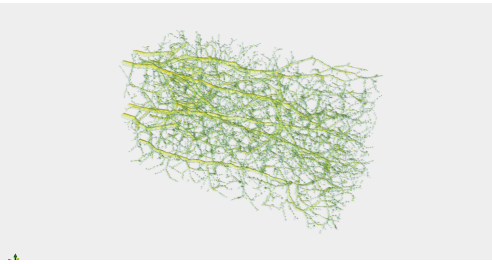
(b) Slice of the input image



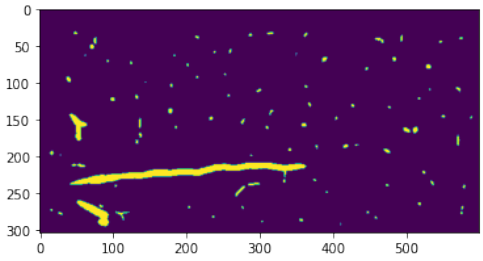
(c) True mask



(d) Slice of the true mask

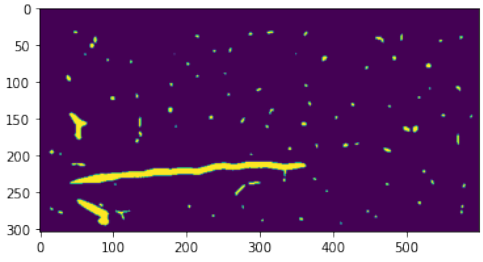


(e) Predicted mask obtained with 3D DVN

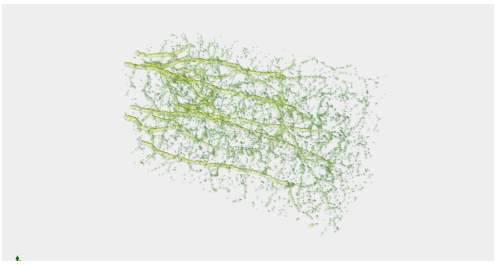


(f) Slice of the predicted mask obtained with 3D DVN

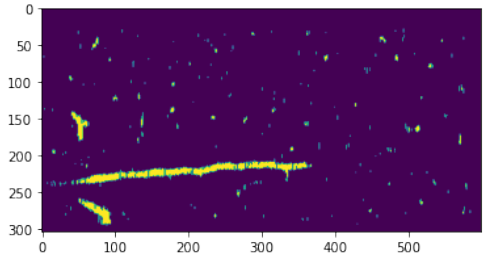
(g) Predicted mask obtained with 3D U-Net



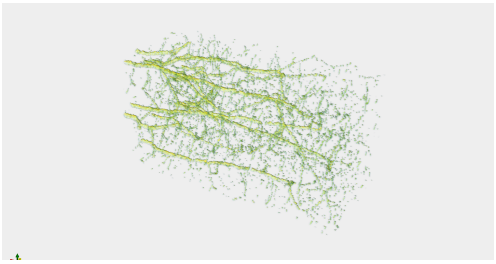
(h) Slice of the predicted mask obtained with 3D U-Net



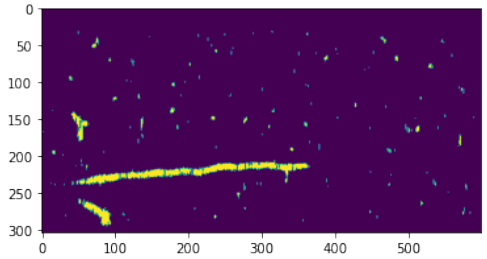
(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN

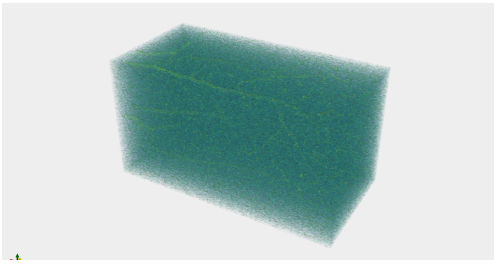


(k) Predicted mask obtained with 2D U-Net

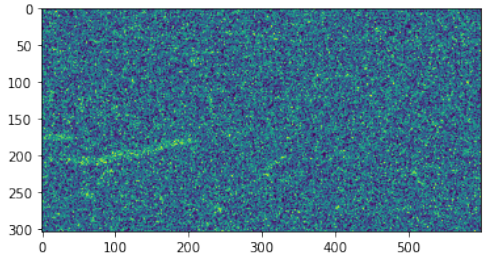


(l) Slice of the predicted mask obtained with 2D U-Net

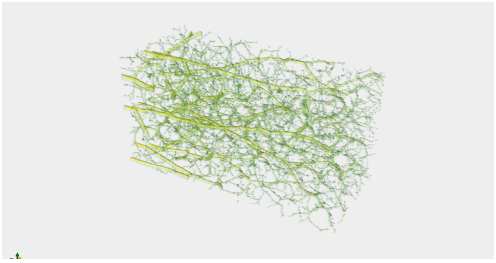
Figure A.0: Input image corresponding to the best dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks



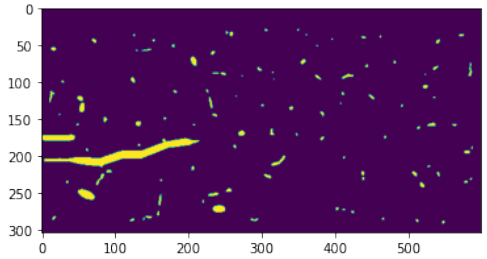
(a) Input image



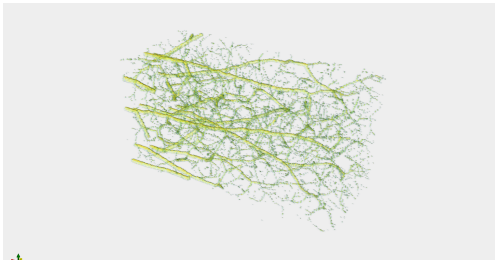
(b) Slice of the input image



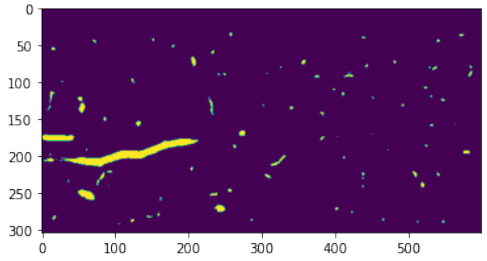
(c) True mask



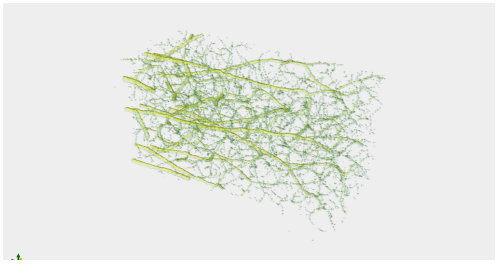
(d) Slice of the true mask



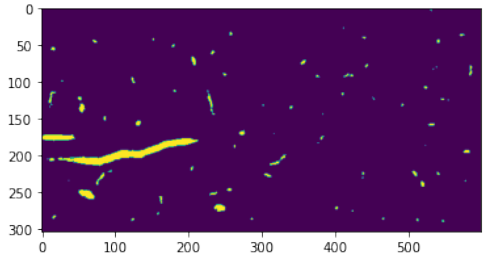
(e) Predicted mask obtained with 3D DVN



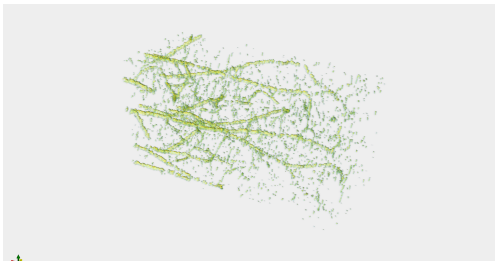
(f) Slice of the predicted mask obtained with 3D DVN



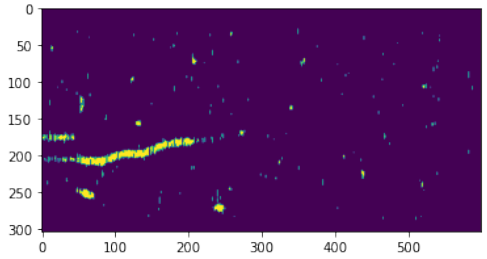
(g) Predicted mask obtained with 3D U-Net



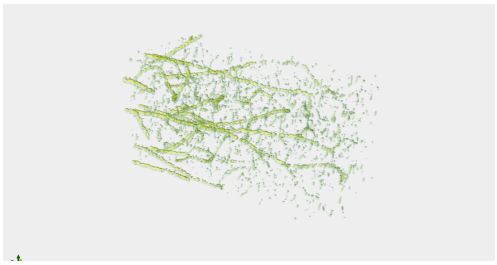
(h) Slice of the predicted mask obtained with 3D U-Net



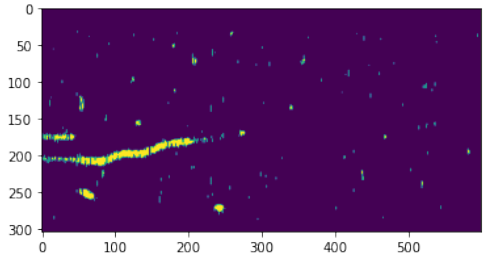
(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN



(k) Predicted mask obtained with 2D U-Net

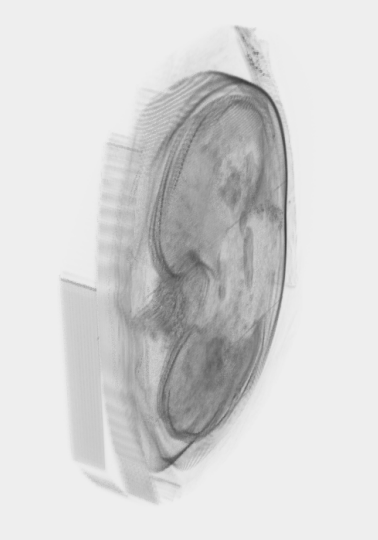


(l) Slice of the predicted mask obtained with 2D U-Net

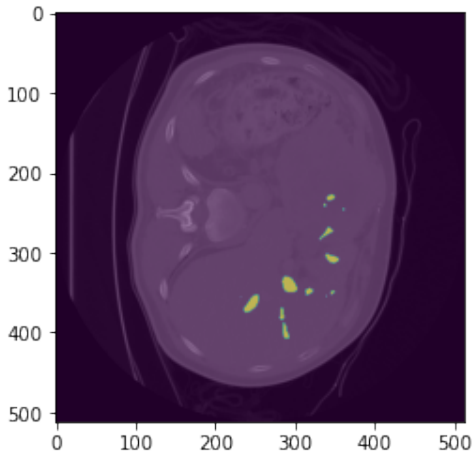
Figure A.0: Input image corresponding to the lower dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks

Appendix B

Some results for trial and error experiments for real dataset

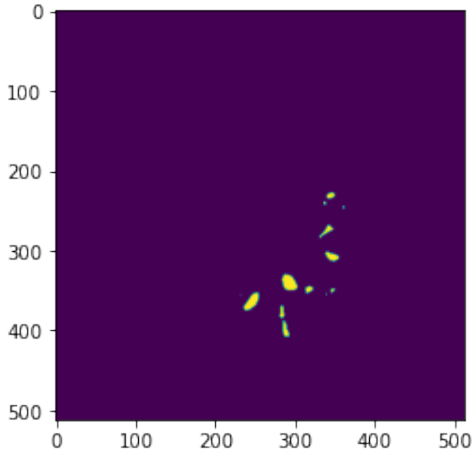


(a) Input image



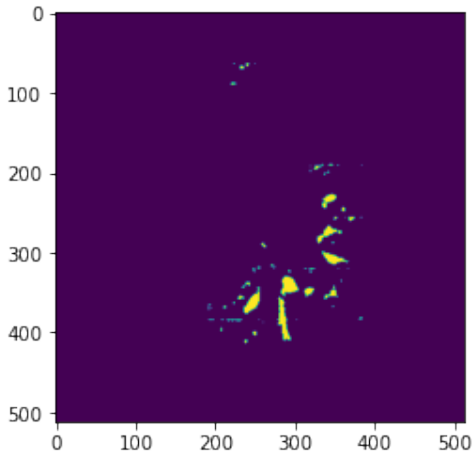
(b) Slice of the input image with its true mask

(c) True mask



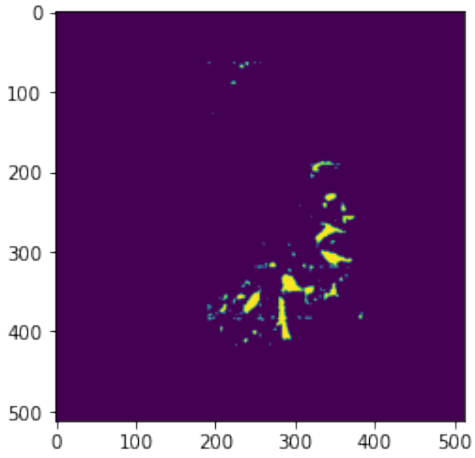
(d) Slice of the true mask

(e) Predicted mask obtained with 3D DVN

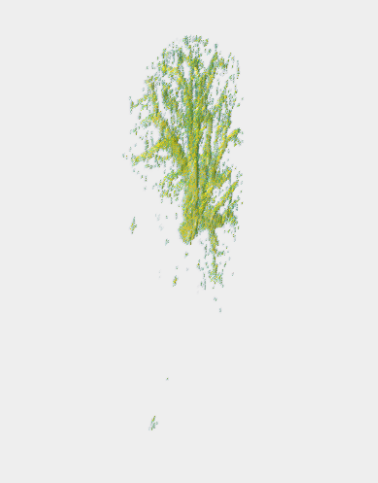


(f) Slice of the predicted mask obtained with 3D DVN

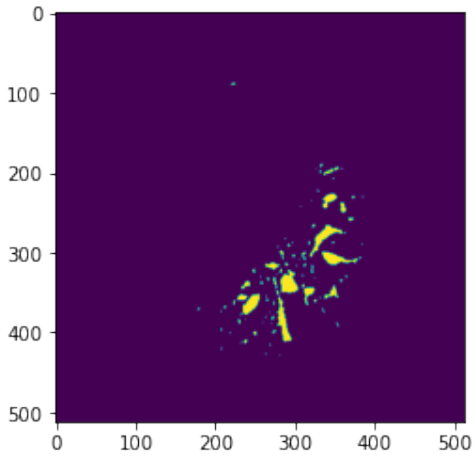
(g) Predicted mask obtained with 3D U-Net



(h) Slice of the predicted mask obtained with 3D U-Net

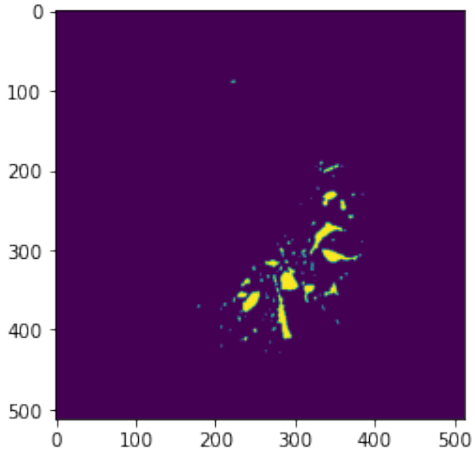


(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN

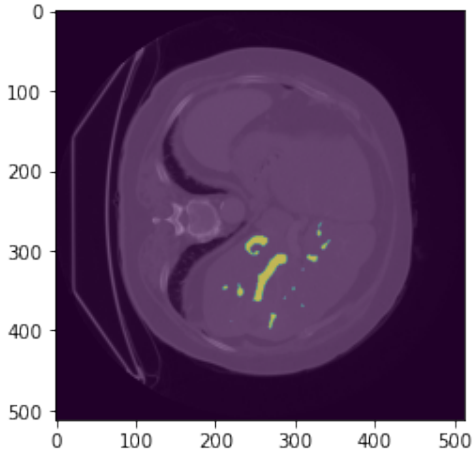
(k) Predicted mask obtained with 2D U-Net



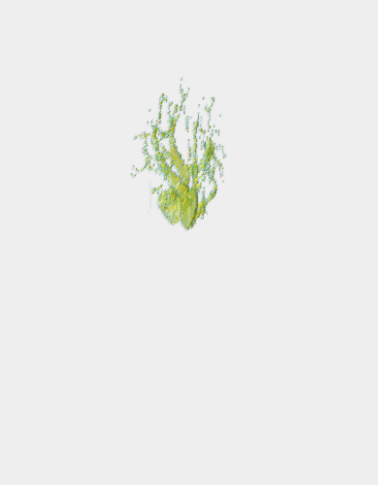
(l) Slice of the predicted mask obtained with 2D U-Net

Figure B.-1: Input image corresponding to the best dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks

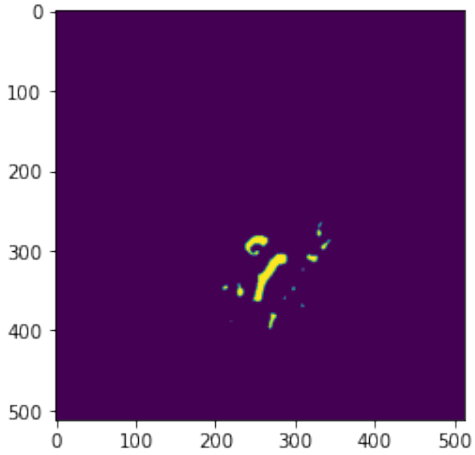
(a) Input image



(b) Slice of the input image with its true mask

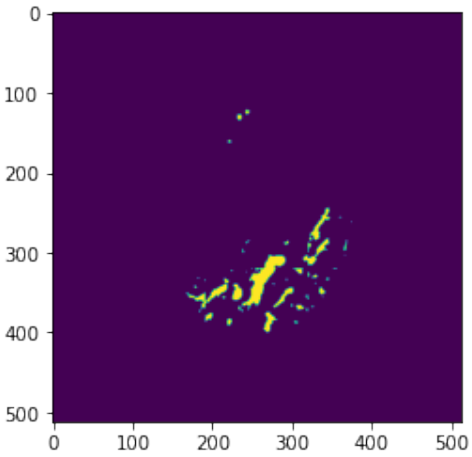


(c) True mask

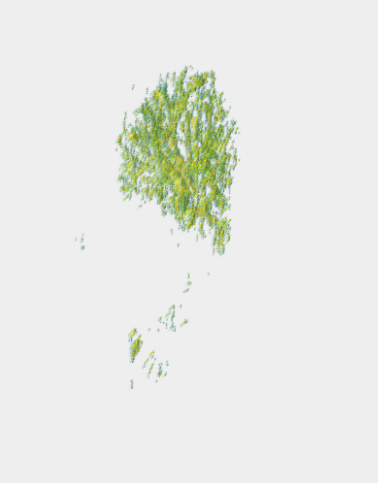


(d) Slice of the true mask

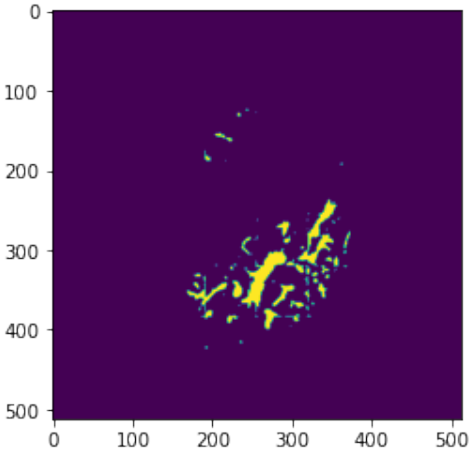
(e) Predicted mask obtained with 3D DVN



(f) Slice of the predicted mask obtained with 3D DVN

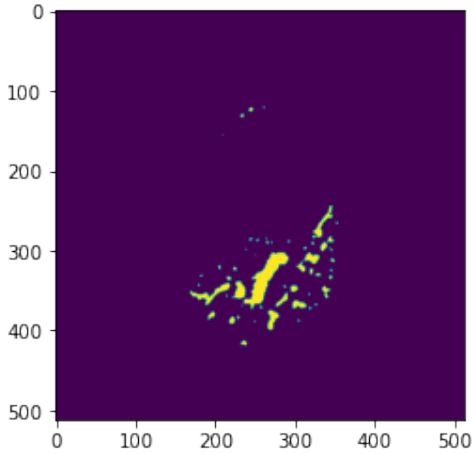


(g) Predicted mask obtained with 3D U-Net

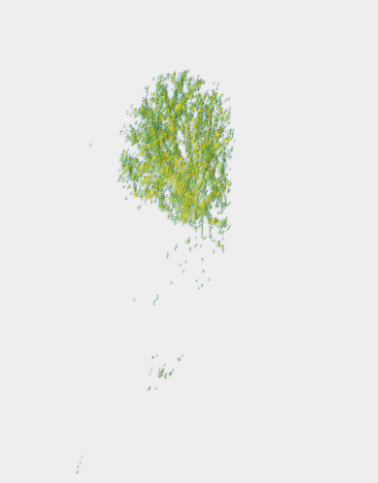


(h) Slice of the predicted mask obtained with 3D U-Net

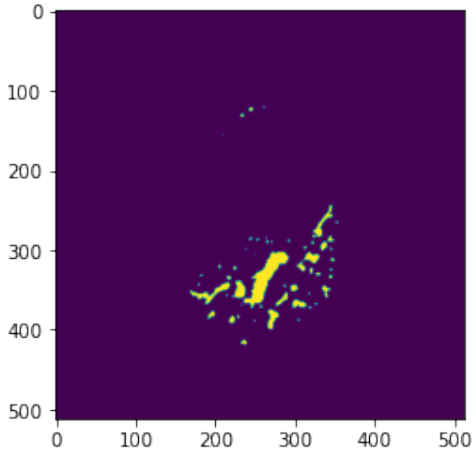
(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN



(k) Predicted mask obtained with 2D U-Net

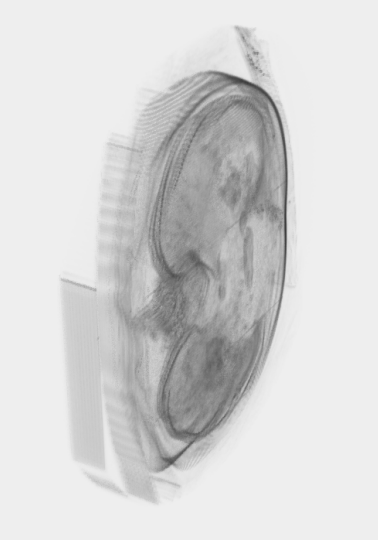


(l) Slice of the predicted mask obtained with 2D U-Net

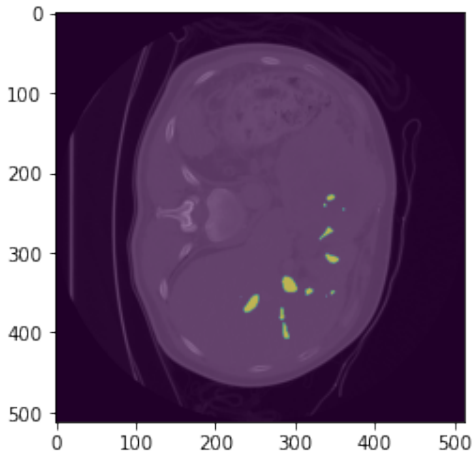
Figure B.-2: Input image corresponding to the lower dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks

Appendix C

Some results for nnU-Net experiments for real dataset

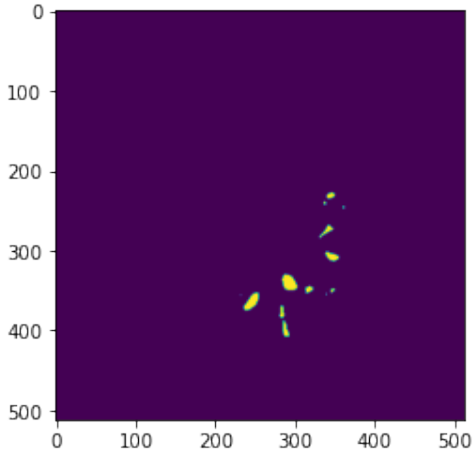


(a) Input image



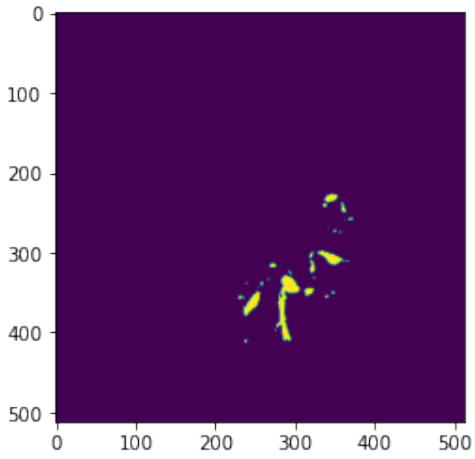
(b) Slice of the input image with its true mas

(c) True mask



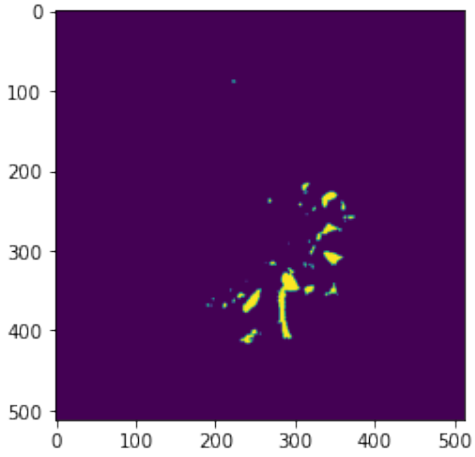
(d) Slice of the true mask

(e) Predicted mask obtained with 3D DVN



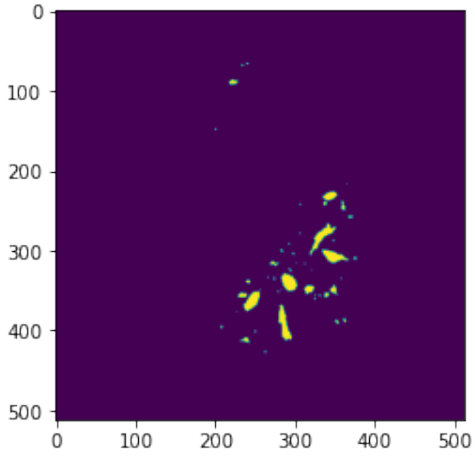
(f) Slice of the predicted mask obtained with 3D DVN

(g) Predicted mask obtained with 3D U-Net



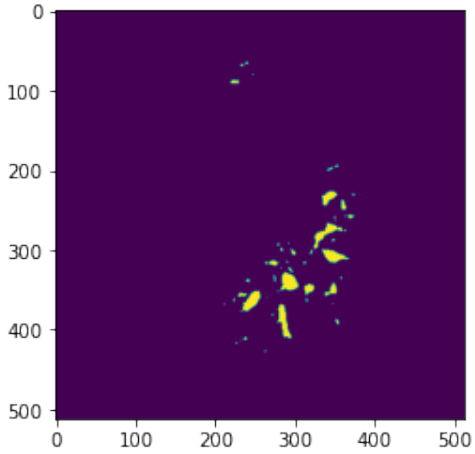
(h) Slice of the predicted mask obtained with 3D U-Net

(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN

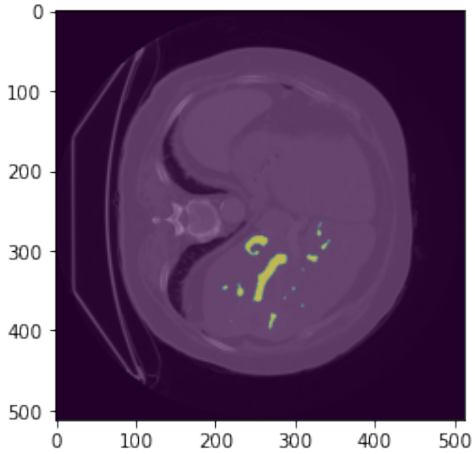
(k) Predicted mask obtained with 2D U-Net



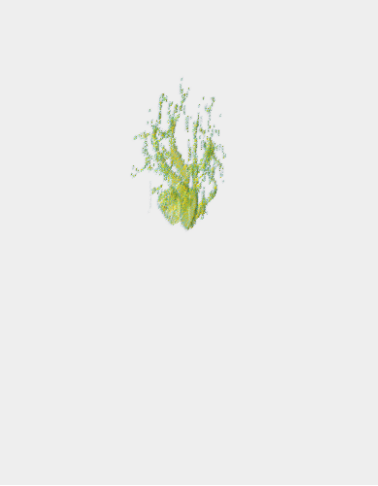
(l) Slice of the predicted mask obtained with 2D U-Net

Figure C.-1: Input image corresponding to the best dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks

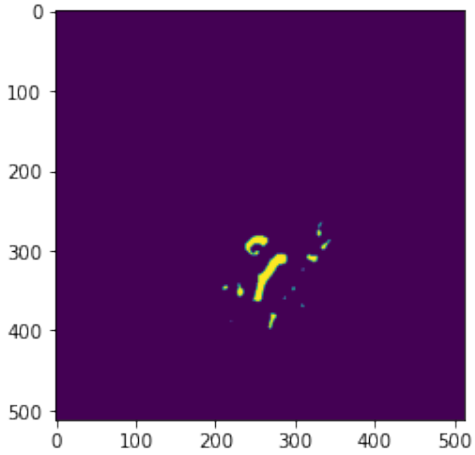
(a) Input image



(b) Slice of the input image with its true mask

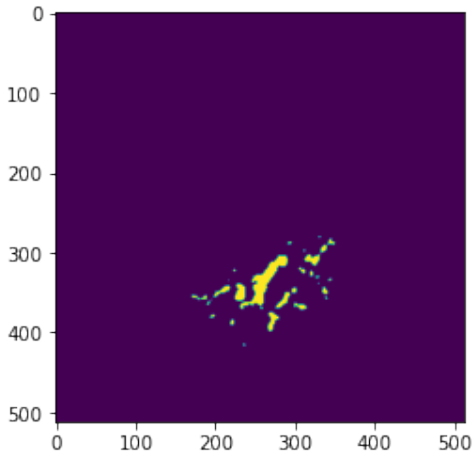


(c) True mask

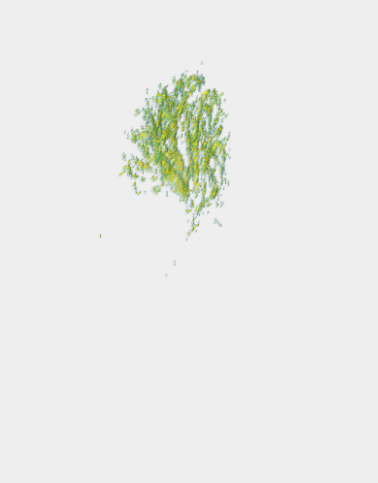


(d) Slice of the true mask

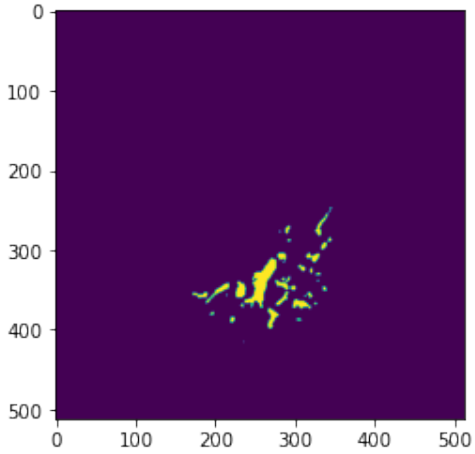
(e) Predicted mask obtained with 3D DVN



(f) Slice of the predicted mask obtained with 3D DVN

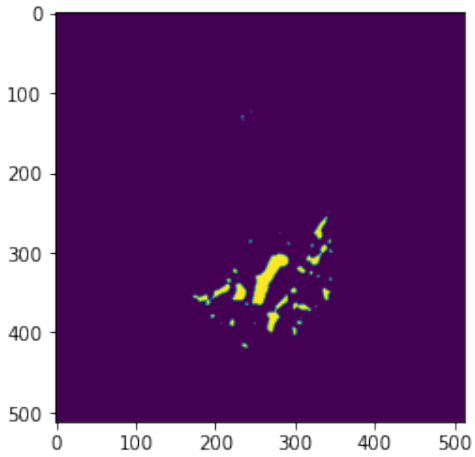


(g) Predicted mask obtained with 3D U-Net



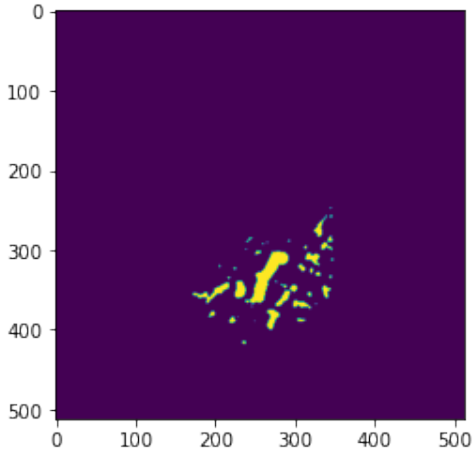
(h) Slice of the predicted mask obtained with 3D U-Net

(i) Predicted mask obtained with 2D DVN



(j) Slice of the predicted mask obtained with 2D DVN

(k) Predicted mask obtained with 2D U-Net



(l) Slice of the predicted mask obtained with 2D U-Net

Figure C.-2: Input image corresponding to the lower dice score for the prediction with the corresponding true mask and the predicted masks obtained with the different networks

Appendix D

Cassiopee server

All the processes are executed on the remote Cassiopee server via SSH protocol. Cassiopee server has four NVIDIA GPUs (4 x GeForce GTX 1080 Ti), with almost 12 GB of RAM. GPUs allow to decrease the computational time by benefiting from the parallelization ability of some tasks.

Bibliography

[1] Danny Varghese. *Comparative Study on Classic Machine learning Algorithms*. https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222. 2018. (Visited on 06/07/2021).

[2] K K D Ramesh et al. *A Review of Medical Image Segmentation Algorithms*. Apr. 2021. DOI: 10.4108/eai.12-4-2021.169184.

[3] Shervin Minaee et al. *Image Segmentation Using Deep Learning: A Survey*. 2020. arXiv: 2001.05566 [cs.CV].

[4] Jaime Durán. *Everything You Need to Know about Gradient Descent Applied to Neural Networks*. https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14. 2019. (Visited on 05/17/2021).

[5] Justin Johnson and Taghi Khoshgoftaar. *Survey on deep learning with class imbalance*. https://www.researchgate.net/figure/Shallow-MLP-vs-deep-MLP-57_fig1_332165523. 2019. (Visited on 05/20/2021).

[6] Vladimir Lyashenko. *Cross-Validation in Machine Learning: How to Do It Right*. https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right. 2021. (Visited on 05/20/2021).

[7] Tom Schaul, Sixin Zhang, and Yann LeCun. *No More Pesky Learning Rates*. 2013. arXiv: 1206.1106 [stat.ML].

[8] Jason Brownlee. *Loss and Loss Functions for Training Deep Learning Neural Networks*. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/. 2019. (Visited on 05/17/2021).

[9] Sakshi Tiwari. *Activation functions in Neural Networks*. https://www.geeksforgeeks.org/activation-functions-neural-networks/. 2020. (Visited on 05/17/2021).

[10] Jason Brownlee. *How to Choose an Activation Function for Deep Learning*. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/. 2021. (Visited on 05/17/2021).

[11] Thomas Wood. *Softmax Function*. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. (Visited on 06/01/2021).

[12] Arden Dertat. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2. 2017. (Visited on 04/19/2021).

[13] Mars Xiang. *Convolutions: Transposed and Deconvolution*. https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6. 2020. (Visited on 06/01/2021).

[14] Godfrey Booth. *Generative Adversarial Network*. https://slideplayer.com/slide/15353689/. 2019. (Visited on 06/02/2021).

[15] Ben Dickson. *What are convolutional neural networks (CNN)?* https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/. 2020. (Visited on 04/19/2021).

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

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[18] Özgün Çiçek et al. *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*. 2016. arXiv: 1606.06650 [cs.CV].

[19] Ozan Oktay et al. *Attention U-Net: Learning Where to Look for the Pancreas*. 2018. arXiv: 1804.03999 [cs.CV].

[20] Nahian Siddique et al. *U-Net and its variants for medical image segmentation: theory and applications*. 2020. arXiv: 2011.01118 [eess.IV].

[21] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[22] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *2016 Fourth International Conference on 3D Vision (3DV)*. 2016, pp. 565–571. DOI: 10.1109/3DV.2016.79.

[23] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[24] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[25] Ming Liang and Xiaolin Hu. *Recurrent convolutional neural network for object recognition*. 2015. DOI: 10.1109/CVPR.2015.7298958.

[26] Zongwei Zhou et al. *Unet++: A nested u-net architecture for medical image segmentation*. in Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support, pp. 3–11. 2018.

[27] Fabian Isensee et al. *nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation*. 2018. arXiv: 1809.10486 [cs.CV].

[28] Nicholas Heller et al. *The state of the art in kidney and kidney tumor segmentation in contrast-enhanced CT imaging: Results of the KiTS19 Challenge.* 2020. arXiv: 1912.01054 [eess.IV].

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

[30] Özgün Çiçek et al. *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation.* 2016. arXiv: 1606.06650 [cs.CV].

[31] *MICCAI Challenge on Circuit Reconstruction from Electron Microscopy Images.* https://cremi.org.

[32] Larissa Heinrich et al. *Synaptic Cleft Segmentation in Non-Isotropic Volume Electron Microscopy of the Complete Drosophila Brain.* 2018. arXiv: 1805.02718 [cs.CV].

[33] Giles Tetteh et al. *DeepVesselNet: Vessel Segmentation, Centerline Prediction, and Bifurcation Detection in 3-D Angiographic Volumes.* 2019. arXiv: 1803.09340 [cs.CV].

[34] Roberto Rigamonti et al. "Learning Separable Filters". In: 2013, pp. 2754–2761. DOI: 10.1109/CVPR.2013.355.

[35] Holger R. Roth et al. "A New 2.5D Representation for Lymph Node Detection Using Random Sets of Deep Convolutional Neural Network Observations". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014.* Ed. by Polina Golland et al. Cham: Springer International Publishing, 2014, pp. 520–527. ISBN: 978-3-319-10404-1.

[36] Siqi Liu et al. "Triple-Crossing 2.5D Convolutional Neural Network for Detecting Neuronal Arbours in 3D Microscopic Images". In: *Machine Learning in Medical Imaging.* Ed. by Qian Wang et al. Cham: Springer International Publishing, 2017, pp. 185–193. ISBN: 978-3-319-67389-9.

[37] Patrick Ferdinand Christ et al. "Automatic Liver and Lesion Segmentation in CT Using Cascaded Fully Convolutional Neural Networks and 3D Conditional Random Fields". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016.* Ed. by Sebastien Ourselin et al. Cham: Springer International Publishing, 2016, pp. 415–423. ISBN: 978-3-319-46723-8.

[38] Anjany Sekuboyina et al. *A Localisation-Segmentation Approach for Multi-label Annotation of Lumbar Vertebrae using Deep Nets.* 2017. arXiv: 1703.04347 [cs.CV].

[39] Matthias Schneider et al. *Tissue metabolism driven arterial tree generation.* Special Issue on the 2011 Conference on Medical Image Computing and Computer Assisted Intervention. 2012. DOI: https://doi.org/10.1016/j.media.2012.04.009. URL: https://www.sciencedirect.com/science/article/pii/S1361841512000576.

[40] Matthias Schneider et al. “TGIF: Topological Gap In-Fill for Vascular Networks”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014*. Ed. by Polina Golland et al. Cham: Springer International Publishing, 2014, pp. 89–96. ISBN: 978-3-319-10470-6.

[41] Benoit M. Dawant et al. *Semi-automatic segmentation of the liver and its evaluation on the MICCAI 2007 grand challenge data set*. 2007.

[42] Amber L. Simpson et al. *A large annotated medical image dataset for the development and evaluation of segmentation algorithms*. 2019. arXiv: 1902.09063 [cs.CV].

[43] M. Nolden et al. *The Medical Imaging Interaction Toolkit: challenges and advances*. 2013.

[44] Xavier Glorot and Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*. 2010. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[45] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[46] Jason Brownlee. *Code Adam Optimization Algorithm From Scratch*. https://machinelearningmastery.com/adam-optimization-from-scratch/. 2021. (Visited on 05/30/2021).

[47] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent. 2016. (Visited on 05/31/2021).

[48] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV].

[49] Shipra Saxena. *Binary Cross Entropy/Log Loss for Binary Classification*. https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/. 2021. (Visited on 05/30/2021).

[50] Saining Xie and Zhuowen Tu. *Holistically-Nested Edge Detection*. 2015. arXiv: 1504.06375 [cs.CV].

[51] Jeremy Jordan. *Evaluating image segmentation models*. https://www.jeremyjordan.me/evaluating-image-segmentation-models/. 2018. (Visited on 04/27/2021).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**