Université de Franche-Comté
Master Informatique Avancée et Applications
Project d'Initiation à la Recherche

Convolutional Neural Networks for Heart MRIs Segmentation

Pierre Feilles *

encadrement: Raphaël Couturier †, Michel Salomon ‡
(Equipe AND: Algorithmique Numérique Distribuée)

September 8, 2018

Abstract

Convolutional Neural Networks (CNN) have proved very efficient in performing image classification tasks but also, after being adequately adapted, image segmentation tasks. In this paper, we describe how we modified and trained one state-of-the-art segmentation CNN architecture, DeepLab, to efficiently segment heart MRI images.

*pierre.feilles@edu.univ-fcomte.fr
†raphael.couturier@univ-fcomte.fr
‡michel.salomon@univ-fcomte.fr

1 Introduction

Machine learning has gained momentum for the past twenty years to the point that it is a fundamental programming paradigm and it will become ubiquitous in applications. More specifically, concerning computer vision tasks, deep learning and convolutional neural networks (CNNs) have been improved dramatically for the past ten years. Even for complex computer vision tasks, like semantic segmentation, CNNs are often today the state-of-the-art solution.

On the other hand, medical imaging has also dramatically been improved and quantities of valuable data are available. This data often entails overwhelming, tedious tasks of analysis for the expert physicians, who could benefit from an automatic image segmentation program, at least as a preprocessing phase of their diagnostic procedures. Moreover, in computer vision in general, it is now common for deep learning models to outperform humans.

However, when it comes specifically to the segmentation of medical images, the models produced so far are still lagging behind. This is due to the specifities of this kind of data sets (for example: few labelled images, *bias field distortion* altering MRI images).

Our goal is to adapt a state-of-the-art semantic segmentation CNN to segment images in a data set containing heart MRIs images, labelled by a cardiologist. The area labelled is an area that needs to be analyzed when patients experienced heart failure. We want to train a deep learning model to automatically segment this area.

We first broadly describe what *machine learning* is (2), and how *Convolutional Neural Networks* (CNNs) are a special kind of machine learning architecture, and what their main purpose is (2.4).

Then, in a general presentation of image segmentation (3), we describe how CNNs could be adapted to perform this task (3.3) and how we chose such an architecture (3.3.6), DeepLab, to be trained to semantically segment our data set of heart MRI images, the aim of our work (4).

Finaly, we detail how we implemented this DeepLab architecture (5) in pytorch, and we summarize the results we obtained (6).

2 Machine Learning

2.1 Overview

2.1.1 Machine Learning

Machine learning at first appears just as a mathematical *optimization* problem, tuning the parameters of a function so that it predicts an output with the least amount of error given an input. "Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions" [10].

The difference between mere optimization and machine learning comes from the data being handled. To speak in machine learning terms, optimization has all the data available and minimizes a cost function on this well defined dataset. "What separates machine learning from optimization is that we want the generalization error, also called the test error, to be low as well. The generalization error is defined as the expected value of the error on a new input. Here the expectation is taken across different possible inputs, drawn from the distribution of inputs we expect the system to encounter in practice" [10].

In machine learning, we can divide the data available to train a machine learning architecture into three sets. The *training set* which is the set used during training to adjust the weights of the network architecture; the *validation set*, different from the training set but drawn from the data according to the same probability distribution, which is used to control overfitting, tune hyperparameters while training and detect when training can be stopped; and the *test set*, also drawn from the data according to the same probability distribution, on which the performance of the trained network is measured.

The model is trained on a training set but also has to perform well on a validation set. Performing too well on the training set and not well on the validation set is called *overfitting* (this happens typically when the model is a polynomial function the degree of which is too high and which is thus capable of interpolating a large set of points). In other words, there is too much *variance* i.e. the model is very sensitive to minor variations in the training set. On the other hand, if the model is too simple (e.g. low degree polynomial function), the model will not give good predictions either as it does not contain enough information about the relationships at work within the data. There is a large *bias* in its predictions.

To design a better machine learning model, we have to make different assumptions on it and/or use techniques designed to minimize the generalization error, like *regularization*.

Stochastic Gradient Descent

The training itself is generally performed using a *Stochastic Gradient Descent* (SGD) algorithm. At each training step, we want to readjust the parameters of the model function so that the cost function decreases. This is performed by gradient descent: in the neighborhood of $\mathbf{w}$, a function f is decreasing most in the $-\nabla f(\mathbf{w})$ direction. Hence the adjustement of the weight vector at each step: $\mathbf{w} - \eta \nabla f(\mathbf{w})$. In its simplest form, we use as f the sum of the errors on

all the observations: $f(w) = \sum_{i=1}^{n} f_i(w)$ and w designates the set of weights of the machine learning architecture. The hyperparameter η, called the *learning rate* has to be adjusted.

It is stochastic when instead of using the whole set of training exemples, the gradient is approximated by the gradient calculated on only one observation, drawn from a probability distribution on all the observations. We iterate until a good enough minimum is obtained: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f_i(\mathbf{w})$. It is also possible, instead of using a single observation as an approximation, to use a small batch of observation drawn from the same probability distribution (*minibatch stochastic gradient descent*).

It should be noted that this algorithm only finds a local minimum, but in practice nevertheless produces good results.

2.1.2 Deep Learning

Although they have been used in a wide variety of applications, "conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input"[17].

Deep learning techniques are a solution to this issue. They allow the construction of complex non-linear functions the weights of which will be adjusted during training in order to learn not only the task they are designed to perform but also the internal representations needed to perform this task. These feature representations do not have to be hand-crafted anymore, they are part of the learning process. Typically these learned features are organized in a hierarchy of layers, the ouput of a layer being the input of the following layer. As the number of layers increases, the machine learning model is becoming "deeper", hence the term *deep* learning.

Deep learning is quickly evolving and new efficient architectures are designed very regularly [10]. We will describe a standard architecture, the *fully connected multi-layer neural network*, and mention two directions for variations on this architecture: *CNNs: Convolutional Neural Networks*, further explained in 2.4, and *RNNs: Recurrent Neural Networks*.

Only *supervised learning* is described here. Indeed, the data set we will use for our segmentation task is composed of labelled data and no preprocessing derived from unsupervised or semi-supervised machine learning techniques will be used.

2.1.3 Neural Networks

The main direction in the development of deep learning models is the *neural networks* architecture. Each layer of the deep learning model is a stack of "neurons". Although there is ground for a genuine analogy to the way actual biological neurons work (chapter 9.10 of [10]), artificial neurons in deep learning

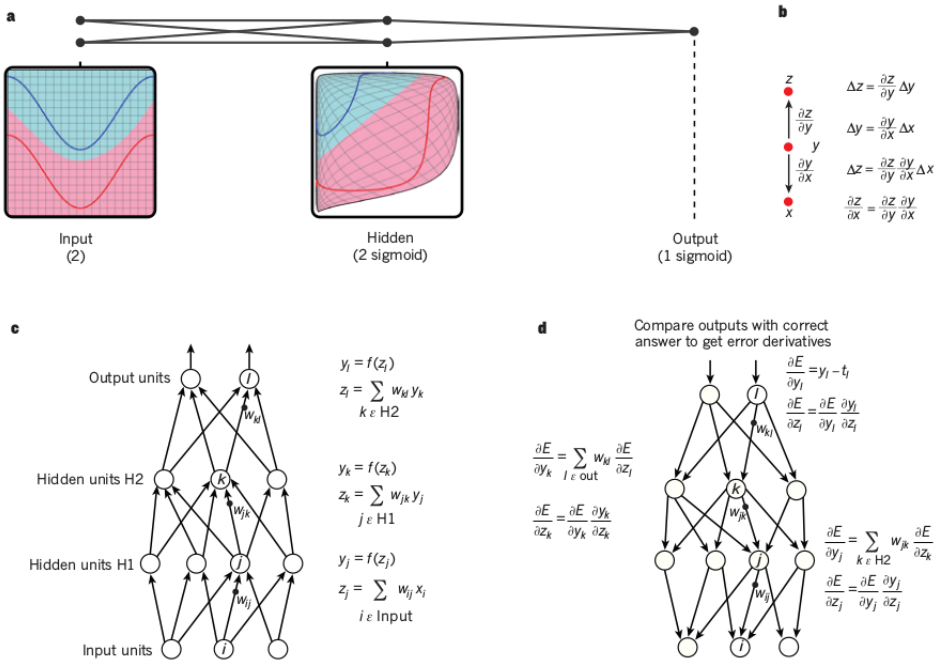


Figure 1: Multi-layer Neuronal Network, figure from Lecun et al. [17]

neural networks are much simpler computing units: they first compute a linear function of all the output of neurons in the preceding layer and pass the result to a non-linear function like the Rectified Linear Unit (ReLU).

Although these neural networks were initially deemed unfit to perform efficiently advanced machine learning tasks and particularly computer vision tasks, they were revived by Lecun [16] and other researchers (see [17] for a panoramic view of this field). Then, the combination of multiple factors (more and more massive datasets, efficient Stochastic Gradient Descent, and generalization of GPUs) made them the state-of-the-art solution for many machine learning tasks (speech recognition, image classification etc.).

2.2 Multi-layer Neural Networks

On figure 1, the layers of neurons are horizontally stacked (a) or vertically stacked (c,d). Part (c) of figure 1 represents the *forward propagation* which is the computation, through the series of layers, of the output given the input. A neural network is a *function*. The computation at the neuron level is the composition of a linear function and of a non-linear one (like sigmoïd or ReLU, Rectified Linear Unit):

$$y_i = f\left(\sum w_{ij} x_i\right). \tag{2.2.1}$$

Each layer will compute a set of *features* thanks to the features from the previous layer, except the first layer which extracts features from the input. If the network is well designed and well trained, these features will eventually be

meaningfull enough to allow the output to be a good prediction for the task at hand.

The synaptic *weights* of the different neurons defining these feature extractions will be automatically calculated through training (and so will not have to be *handcrafted* to then be fed to a classical machine learning classifier). Several teams (see [17]) in the 1970s and 1980s found out that *stochastic gradient descent (SGD)* could be applied to a multi-layer architecture (part (d) of figure 1) thanks to a *backpropagation algorithm*. The idea is that we are still minimizing a cost function with respect to the weights, and in order to do this, we backpropagate the adjustements of all the weights through the network, using the chain-rule for the derivation of composed functions.

Several computational techniques have been created in order to improve training speed and quality of prediction by the networks [10]. For example:

- *Minibatch gradient descent*: on each step of the backpropagation algorithm instead of adjusting the weights with one example (SGD), we sample a minibatch of examples drawn uniformily from the training set. Then we sum or average the gradient over the minibatch. Here the size of the minibatch is a new hyperparameter.

- *Regularization*: the idea is to add a penalty (*regulizer*) to the cost function. For example, *weight decay* will add $\mathbf{w}^\top \mathbf{w}$ to the cost function and thus penalize weights which become too high, thus preventing overfitting without modifying the functions defining the model. "Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [10]. Many other regularization techniques are available (chapter 7 of [10]), for example, *dropout* (chapter 7.12 of [10]).

- *Batch normalization* (chapter 8.7.1 of [10]): This is a method designed to solve an issue concerning more specifically very deep networks. Without batch normalization, during training we update each parameter according to the calculated gradients assuming the other layers do not change, which is not the case since all layers are updated simultaneously. When the network is deep, this will make the choice of an appropriate learning rate difficult. Batch normalization reparametrizes the model to make some units always standardized by definition.

- *Momentum*: When learning with SGD is too slow, the momentum algorithm computes an exponentially decaying moving average of past gradients, and uses this average to update the weights [10].

2.3 Recurrent Neural Networks

Recurrent Neural Network are typically well suited to handle data in the form of time series like in speech recognition tasks. Figure 2 from Lecun et al [17] is an example of this kind of network architecture. It appears in certain semantic segmentation architectures but we are not using it for our project.

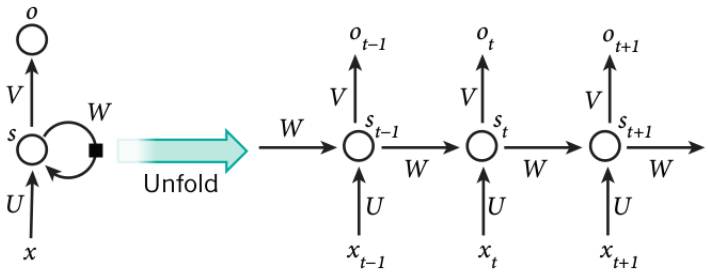


Figure 2: Recurrent Neural Network [17]

2.4 Convolutional Neural Networks (CNNs)

This is the neural network architecture we are using in this work. CNNs are typically used (but not only) for image processing tasks. Figure 3 shows an example of a CNN designed for a classification task. The main idea is that convolutional kernels can be used to extract useful features on images of any size. Fully connected networks work on fixed sized input, are too computationnally expensive for these tasks, and are not well suited when properties like spatial invariance are useful for the prediction task. So in CNNs, the first layers are convolutional layers which will provide a feature vector that can be fed to a regular fully connected sets of layers.

We first describe what convolution originally is in mathematics and how it is adapted to neural networks. This part is essentially a quick summary of chapter 9 of *Deep Learning* by Goodfellow et al. [10]. We then describe the main types of CNNs.

2.4.1 Convolution

Let's start by the convolution operation applied to a one-dimensional continuous signal:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da, \tag{2.4.1}$$

where in (2.4.1) s is the *output* or *feature map*, x is the *input*, and w is the *kernel* (example from [10]). We choose the kernel as a probability density function:

$$\int w(a)da = 1, \tag{2.4.2}$$

hence the convergence of the integral in (2.4.1), and also the meaning in physics if t is time: $s(t)$ is a form of running average according to the probability density function w. (We impose $w(r) = 0$ if $r < 0$ otherwise we would be using future values of the signal to predict its current value).

Here is the *discrete convolution* of a (one-dimensional) discrete signal:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.4.3}$$

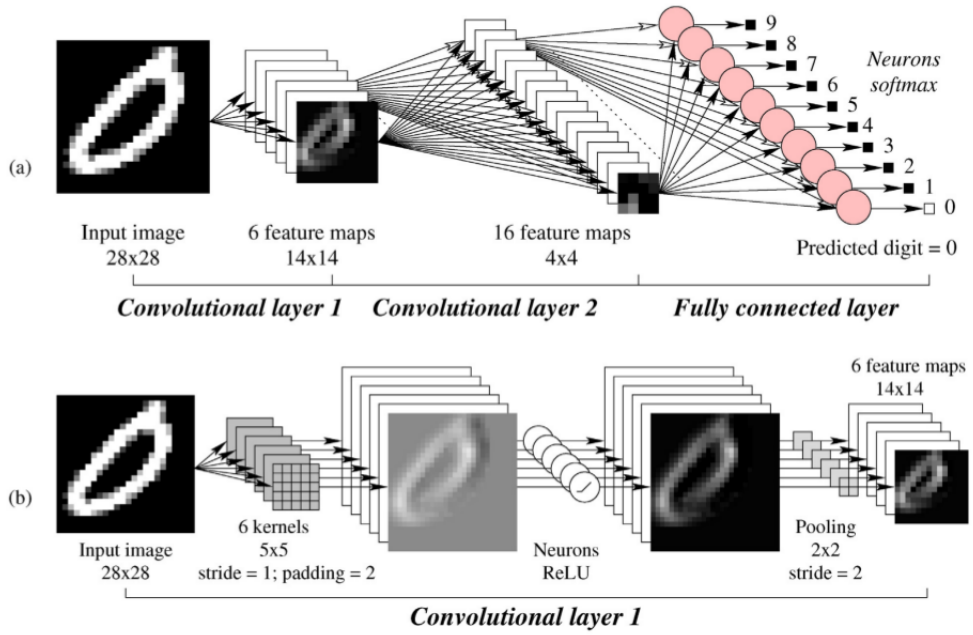


Figure 3: CNN for the MNIST problem [27]

Typically for us, the input signals will be two-dimensional images, and the convolution will be performed over the two axis. So the convolution of a two-dimensional input image I with a two-dimensional kernel K has this form :

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.4.4}$$

A related function is the *cross-correlation*:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n), \tag{2.4.5}$$

which is an operation similar to convolution, the only difference being that the kernel K is not "flipped". In machine learning, the cross-correlation itself is often called convolution. It does not really matter as the learning will just learn the right kernel. Of course with cross-correlation, we do not have the commutative property of genuine convolution, but it does not matter either since convolution will just be a first step and the other functions we will use will not be linear, so the commutativity would have been lost anyway. And even with convolution alone, if it is performed along multiple image channels, commutativity is not guaranteed.

2.4.2 Convolution in CNNs

From a purely algebraic point of view, convolution itself is a linear operation between two layers and could be represented by a matrix. It would be inefficient to implement the operation this way, and we loose much of the intuition of what this operation is about, but this algebraic representation is useful for at least

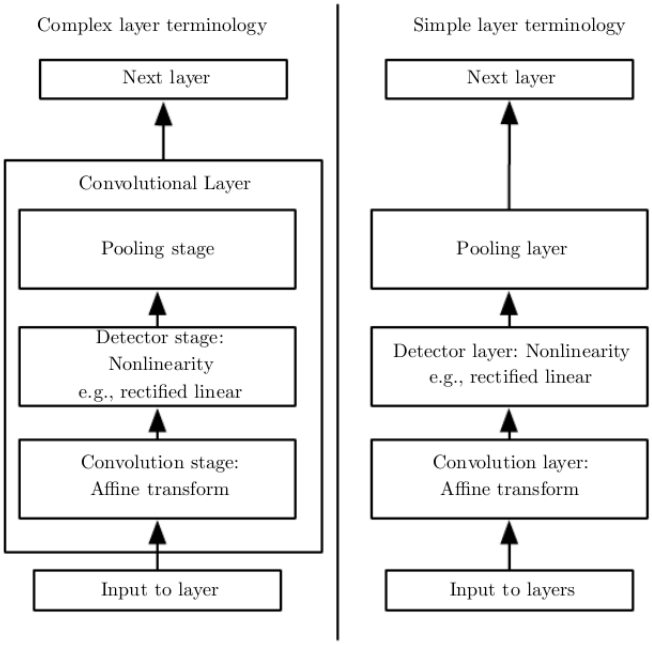


Figure 4: Convolution Layer Terminology [10]

two reasons. First it helps realizing that the convolutional part of a network is not theoretically different from the more traditionnal part that often follows it, i.e. the "fully connected layers" part. Secondly, it helps understanding that convolution is in fact an *infinitely strong prior* imposed on the network and as such it can cause underfitting.

In neural networks terminology, "convolution" often refers in fact to several operations, not just convolution per se. As is summed up in figure 4, there are two types of terminology when it comes to describing neural networks. A "complex layer terminology" in which each layer is an actual layer, i.e. a layer which processes a usual feature vector input and produces an output thanks to its kernel tensor. Or we can decompose more the operations and use a "simple layer terminology" in which not each layer has parameters. Here we will use the complex layer terminology.

In figure 3 is an example of a CNN. As we can see, the complex layer terminology is used. If we look at convolutional layer 1, the convolution operation in the strict mathematical sense is the convolution operation of the input image (28x28) with one kernel. Stride is 1 and padding is 2 so the convoluted image obtained is 28x28. Six kernels are applied in parallel so we get six convoluted images. Then after ReLu functions and 2x2 pooling, we obtain six 14x14 "feature maps". Each of these transformed images will have extracted useful *features* or patterns and will be used as input to the next layer.

Convolutions (in the strict math sense) are powerful at extracting significant features from images, and in a CNN architecture, as in most deep neural net-

works, all the parameters are learned which means that the convolution kernels are learned.

3 Image Segmentation

The semantic segmentation of an image consists in assigning a label to each pixel of the image. There is a wide variety of applications; for us, the goal is to draw contours on heart MRI images such that these contours identify a zone of interest to be checked in order to dectect a pathology.

3.1 Quantitative evaluation of performance

Here are the main metrics for semantic segmentation [9]. Let us denote $k+1$ the number of classes and p_{ij} the number of pixels of class i inferred to be of class j. Thus p_{ii} for example is the number of true positives for class i.

3.1.1 Pixel Accuracy

It is the ratio of properly classified pixels over the total number of pixels:

$$PA = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k} \sum_{j=0}^{k} p_{ij}}. \tag{3.1.1}$$

3.1.2 Mean Pixel Accuracy

Here we compute for each class the pixel accuracy, and we take the average of theses numbers:

$$MPA = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij}}. \tag{3.1.2}$$

3.1.3 Mean Intersection over Union (MIoU)

Usually, the intersection over union (IoU) is the ratio between the intersection and the union of two sets. Here the two sets are the ground truth (GT) and the prediction (Pred). Let $|X|$ denote the cardinal of a set X. For one given class i, $|GT \cap Pred| = p_{ii}$ and $|GT \cup Pred| = p_{ii} + \sum_{j \neq i} p_{ij} + \sum_{j \neq i} p_{ji}$ because $\sum_{j \neq i} p_{ij}$ is the number of false positives and $\sum_{j \neq i} p_{ji}$ is the number of false negatives.

$$IoU = \frac{p_{ii}}{p_{ii} + \sum_{\substack{j=0 \\ j \neq i}}^{k} p_{ij} + \sum_{\substack{j=0 \\ j \neq i}}^{k} p_{ji}} \tag{3.1.3}$$

and then the average over the classes is:

$$MIoU = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{p_{ii} + \sum_{\substack{j=0 \\ j \neq i}}^{k} p_{ij} + \sum_{\substack{j=0 \\ j \neq i}}^{k} p_{ji}}. \tag{3.1.4}$$

3.1.4 Frequency Weighted Intersection over Union (FWIoU)

In this case, each class is weighted according to its frequency.

$$FWIoU = \frac{1}{\sum_{i=0}^{k}\sum_{j=0}^{k} p_{ij}} \sum_{i=0}^{k} \frac{\sum_{j=0}^{k} p_{ij}p_{ii}}{p_{ii} + \sum_{\substack{j=0 \\ j\neq i}}^{k} p_{ij} + \sum_{\substack{j=0 \\ j\neq i}}^{k} p_{ji}}. \tag{3.1.5}$$

3.1.5 Dice's coefficient

For two sets in general, Dice's coefficient is defined by:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}. \tag{3.1.6}$$

In our case, for a given class:

$$DSC = \frac{2|Pred \cap GT|}{|Pred| + |GT|} \tag{3.1.7}$$

(GT: Ground Truth image, and Pred: prediction image) or also:

$$DSC = \frac{2|TP|}{2|TP| + |FP| + |FN|} \tag{3.1.8}$$

(TP: True Positives, FP: False Positives, FN: False Negatives).

3.2 Prior to Neural Networks

Prior to neural network, feature recognition in images had to be *handcrafted* [17]. Traditional machine learning architectures had trouble handling input too high-dimensional (like an image, for which each pixel is another dimension). Before processing natural data, it had to be transformed so that relevant features could be extracted and fed to the machine learning architecture. This feature extraction was a domain requiring a high level of expertise and often the machine learning architectures were only able to execute classification tasks, not semantic segmentation.

So in order to perform semantic segmentation, in addition to these handcrafted kernels perfoming feature extraction, were techniques working on superpixels, or handling interactions between label assignment and more generally developing probabilistic graphical models like Conditional Random Fields (CRFs) [15] [22]. Probabilistic graphical models like CRFs (see chapter 3.3.3) can still be useful as a post-processing step; Chen et al used it in [4]. Although in this case, a futher iteration of their work performed better and did not have to use CRFs [5].

3.3 CNNs for Image Segmentation

3.3.1 Limitations of classical CNNs

Classical CNNs were not initially designed for segmentation tasks. They are well adapted to process images but were designed to classify these images, i.e. label each image as a whole. Indeed, they are powerful because they learn well spatial

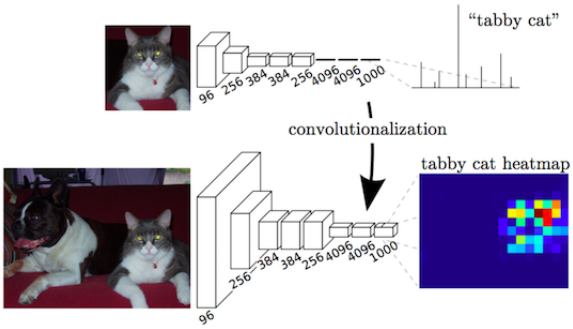


Figure 5: FCN: Fully Convolutional Network, [19]

invariance, but several issues arise when the task involves pixel-wise labeling [4] [9]. First, these CNNs typically reduce feature resolution through the pooling steps of their layers. These poolings are nececessary to reduce the number of parameters and training time, but they are a concern for spatial accuracy. Also the existence of objects at multiple scales is harder to handle when we move from classification to semantic segmentation. Semantic segmentation needs to strike the right balance between local and global information. And even without pooling steps in their layers, in CNNs the receptive field of units grow linearly with the number of layers. So the units that have the entire initial input as a receptive field do not appear until sufficiently deep layers in the network.

There are several solutions that have been used, individually or combined, in order to have a network that builds features that represent global information: refinement as a postprocessing step with Conditional Random Fields (CRFs), dilated or atrous convolutions, multi-scale aggregation, or even defer the context modeling to another kind of deep networks such as RNNs. [9]

In a paper published in 2017 Garcia-Garcia et al. proposed *A Review on Deep Learning Techniques Applied to Semantic Segmentation* [9]. Another review was proposed in a blog post by Chilamkurthy [6], *A 2017 Guide to Semantic Segmentation with Deep Learning*; We summarize hereafter the elements of these two reviews relevant to our task.

3.3.2 FCN: Fully Convolutional Network

Let us first examine the so called *Fully Convolutional Network* [19]: the idea is to replace in existing CNNs the fully connected part of the neural network by convolutional layers. Then the output can be a complete spatial map instead of a classification score, thanks to upsampling (these upsampling layers are called *deconvolution layers*). In order to avoid getting too coarse results (caused by the pooling layers), the final prediction layer can be combined with lower layers with finer strides (i.e. *shortcut* or *skip* connections are introduced). See Figure 5.

Although a major contribution, it has a few drawbacks [9], in particular "its inherent spatial invariance does not take into account useful global context information". That is: the very advantage of convolution (idenfifying similar

patterns anywhere in an image) becomes a problem when we need the information we need for an efficient segmentation is somehow carried by the global context.

To describe variations of this initial solution, we may use the **encoder / decoder terminology** to describe the modification of a classification CNN: the *encoder* part is the convolutional part of the classification CNN and the decoder part is the layers used to replace the fully connected part. So different choices for the decoder will provide different solutions for semantic segmentation. The encoder, through its pooling operations, reduces the spatial dimension; to reach dense pixel-wise prediction, the spatial dimension has to be restored for example as in [19] through upsampling layers and skip connections. SegNet is an example of a different architecture for the decoder [1] designed to improve FCN.

3.3.3 CRFs : Conditional Random Fields

A way to improve pixel-wise predictions is to use CRFs as a post-processing step [9]. CRFs are a kind of probabilistic graphical model [22] [15]. They have been used widely in image processing before deep learning. A probabilistic graphical model links the input X (*observation variables*, for example an image or a set of features derived from that image) to an output Y (*output variables*, for example a pixel-wise prediction fo a pixel belonging to a class). The probabilist model will specify how X and Y interact with each other. CRFs take correlations between observation variables into account in their model, and when the model is simple enough (e.g. fully connected pair-wise CRFs), conditional probabilities of Ys given Xs can be computed. "By using this model, both short and long-range interactions are taken into account, rendering the system able to recover detailed structures in the segmentation that were lost due to the spatial invariance of the CNN. Despite the fact that usually fully connected models are inefficient, this model can be efficiently approximated via probabilistic inference" [9]. Indeed, some parameters of the model can be learned, the CRF becoming a sort of machine learning fully connected layer [22].

3.3.4 Atrous (dilated) Convolutions

As we have seen, when we want to adapt classification CNNs to segmentation tasks, the main issue is constituted by the pooling stages, which eventually diminish spatial accuracy by reducing resolution. Another issue is the size of the receptive field for each node, which can only grow linearly with the number of layers, making it harder to grasp the global context.

Dilated convolutions or *atrous convolutions* can support exponential expansion of the receptive field without loss of resolution or coverage [34]. In [34], the authors have designed a *context module* specifically for dense predictions, a rectangular prism of (dilated) convolutional layers, with no pooling or subsampling [34].

Using the same notations as in the article, $F : \mathbb{Z}^2 \to \mathbb{R}$ is a discrete function (e.g. one channel of an image, the support of F is finite), $\Omega_r = [-r, r]^2 \cap \mathbb{Z}^2$ and $k : \Omega_r \to \mathbb{R}$ a discrete filter of size $(2r + 1)^2$, (this filter is what we have

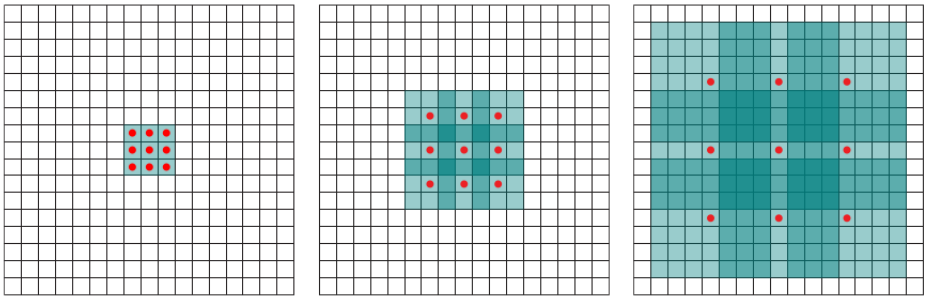


Figure 6: Receptive Fields: (a) F_1 is produced from F_0 by a 1-dilated convolution (b) F_2 is produced from F_1 by a 2-dilated convolution (c) F_3 is produced from F_2 by a 4-dilated convolution [34]

called the *convolution kernel*). The two-dimensional discrete convolution (with flipped kernel) can then be written as:

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}). \tag{3.3.1}$$

Then *dilated convolution* can be defined as:

$$(F *_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}), \tag{3.3.2}$$

where l is the dilatation factor ($l = 1$ corresponds to the usual convolution).

We then have the following property: *dilated convolutions support exponentially expanding receptive fields without losing resolution or coverage* [34]: if we consider a series of $n-1$ 3x3 kernels, k_i, and F_0 an initial image or feature map, that is an initial $\mathbb{Z}^2 \to \mathbb{R}$ function, and we apply the kernels with exponentially increasing dilatation: $F_{i+1} = F_i *_{2^i} k_i$ for $i = 0, 1, ..., n-2$, then the size of the receptive field of each element in F_{i+1} is $(2^{i+2} - 1)^2$, whereas the number of parameters associated with each layer remains the same so, when adding layers, the receptive field grows exponentially while the number of parameters grows linarly (see figure 6).

An animation by Vincent Dumoulin showing this type of dilated convolution is available online [8]. An excerpt of this animation is reproduced figure 7.

This is the neural network architecture we will use (two *DeepLab* variants). It is described in more details in 5.3.

3.3.5 Multi-scale prediction

In multi-scale prediction, the idea is to combine the outputs of several networks, or to train networks in which each layer has several channels that are eventually combined. For example, in [26], the network has two paths, one working with original resolution, the other doubling it. Then the two results are combined. Or networks can be trained independently at different scales and then combined [2].

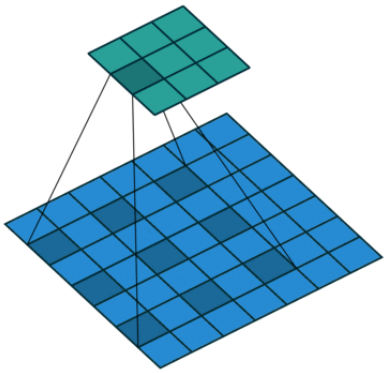


Figure 7: One step of an atrous convolution (for the complete animation online, see [8])

Another example is [34], where in their "context module", the results of several dilated convolutions, with different dilation factors, are combined.

3.3.6 Performance

In [9], Garcia-Garcia et al. used several performance metrics to assess the different segmentation systems: execution time, memory footprint, and accuracy (Pixel Accuracy (PA), Mean Pixel Accuracy (MPA), Mean Intersection over Union (MIoU), Frequency Weighted Intersection over Union (FWIoU)). They found that DeepLab [4] outperforms the other solutions on almost every single RGB images dataset. Similarly, in [6], Chilamkurthy ranks DeepLabv3 [5] first.

We will be using DeepLabv2 and DeepLabv3 (5.3).

3.4 Medical Images: Specific constraints

For this project, we will process heart MRI images. So in this section, we review briefly the main challenges concerning the processing of medical images in general and MRIs in particular.

3.4.1 Medical images

Even if medical image acquisition has improved steadily over the years, automated image interpretation is lagging behind [12].

The main challenge concerning medical images is that, in general, the initial dataset is small and deep learning methods have been very efficient when trained on large datasets. The datasets are small in particular in supervised learning since medical images have to be labelled by medical experts. (And even when it is possible to get good labelling by non-experts, it is still a human task that cannot be automated).

In [12], two main directions to design efficient deep learning solutions on small datasets are mentionned: *transfer learning*, that is using a network trained on a general set of images and adapting it to a medical imaging task; the other

direction suggested is to combine deep learning predictions with *handcrafted feature extraction* methods. There is also a whole set of *data augmentation* techniques available [33].

3.4.2 MRI images

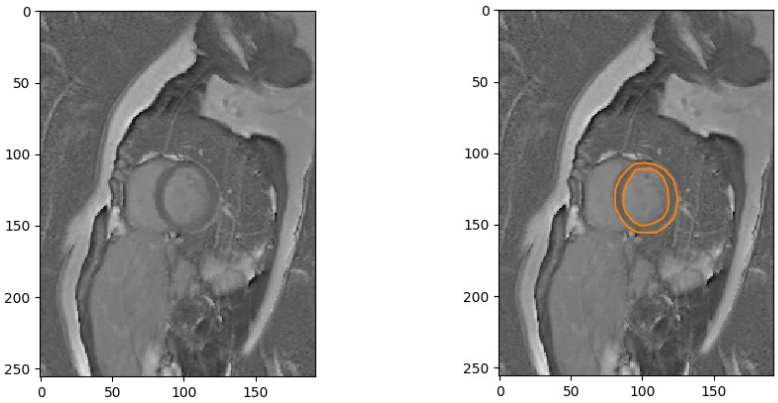
MRI may require a special kind of image preprocessing. Indeed, MRI images are altered by the bias field distortion [25]. Therefore, we will get variations in image intensity not only accross patients but also within the same tissues of a given indivudual. For example, Pereira et al. [25], working on brain MRIs, normalized the images (using a method proposed by Nyúl et al. [23]) and then computed the mean intensity value and standard deviation across all training patches extracted for each sequence and finally normalized the patches on each sequence to have zero mean and unit variance.

We have not used these image preprocessing techniques in our implementation.

4 Aim of this research project

Our goal is to train a neural network to perform a standard segmentation task on a set of labelled heart MRIs. We want the network to draw contours delimiting a zone of interest (this zone is checked after heart failure episodes in order to detect specific damage to the heart muscle).

On figure 8 is an unlabelled MRI image of a heart and the same image manually labelled by a physician: we can see the two contours delimiting the area of interest. We will train our networks to label this area. This is a segmentation task: the network will decide for each pixel if it belongs to the area of interest of not.



(a) unlabelled (b) manually labelled

Figure 8: A Heart MRI

The network will in particular have to be efficient regarding scale. Figure 9 shows a sample of the data : the MRIs concern only one patient, and were performed on the same exam procedure. These are slices of the heart, and we can see how the contours prediction will have to adjust to scale well.

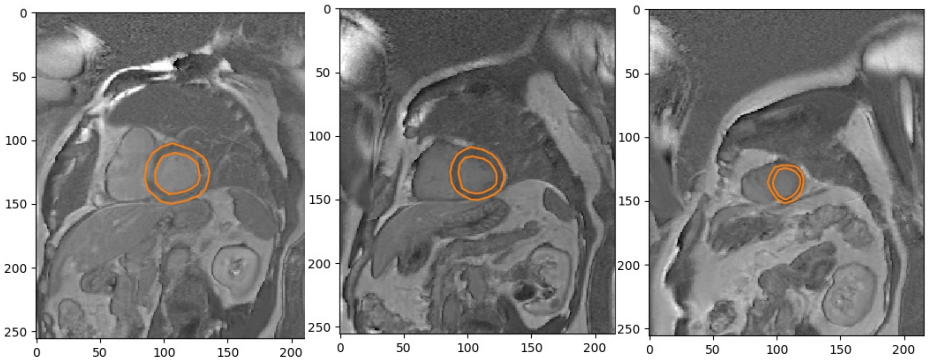


Figure 9: Three slices of the same 3D MRI

5 Implementation

5.1 Data set

The data set we used is composed of the heart MRIs of about a hundred patients. For each patient, there are about 6 images available. See figures 8 and 9. All images are in the DICOM format. They are all manually labelled by a cardiologist who delimited an area of interest within the heart. This is the area we want our program to automatically segment.

5.2 Image Preprocessing

We have mentionned in 3.4.2 that MRI images may require a special kind of preprocessing. Still here, we obtained results by keeping preprocessing to a minimum.

DICOM images and MATLAB labels were converted into PNG images. For the labels, we considered the entire area between the two orange lines (see figure 8) to be the *contour*, and we trained our network to detect this contour.

Our images are one-channel. We just subtracted the mean pixel value calculated over the whole dataset (144.686 in our case). This leaves a few problematic images, but overall it worked well.

All images were standardized to a 513×513 shape. Data augmentation was performed by setting the crop size to 450, large enough so that the pattern we are interested in is present in the image.

The data set was split into two sets, the training set and the test set. It proved important to select *patients* at random, and then pick images at random for this given patient. Indeed, for a given patient, images can be very similar (figure 9) and, as we have experienced, the risk is to overfit by having many images coming from too few patients.

5.3 Neural Network Architecture: DeepLab

As of today, several benchmarks (3.3.6) rank the *DeepLab* neural network architectures first or among the first. So we chose to use them to implement our semantic segmentation model.

They have been developped by Chen et al. first in [3] then in [4] and it was finally refined in [5] by the same team. These DeepLab networks were specifically designed from the beginning to execute semantic segmentation tasks, although they are built upon a state-of-the-art ResNet [14], a network designed to perform classification tasks.

5.3.1 DeepLabv2

In [4], Chen et al. proposed an (atrous) convolutional architecture (named later *DeepLabv2*). They built their network upon ResNet [14] and without changing the number of parameters they obtained a semantic segmentation network:

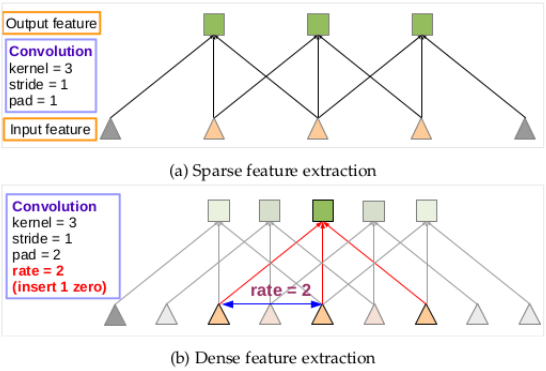


Figure 10: illustration of a 1-dimension atrous convolution [4]

increased resolution, increased field of view and also the handling of multiple scales.

Atrous convolutions

Similarly to what we have seen in 3.3.4, they defined in one dimension convolution as:

$$y[i] = \sum_{k=1}^{K} x[i + rk]w[k], \tag{5.3.1}$$

with w a filter (i.e. kernel) non-mirrored of length K. r is the rate, creating holes ("à trous") in the convolution. It is thus sampling the input signal x (figure 10). Atrous convolution allows to produce outputs at arbitrary high resolutions without modifying the number of parameters.

Increased resolution

This is how DeepLabv2 is built, modifying the classification ResNet network. For example, if the last layer of a network has a pooling layer of stride 2, it is possible to double the output density by setting this last layer stride to one, and replacing all subsequent convolutional layers with atrous convolutional layers having rate $r = 2$. They found it too costly to repeat this transformation on all the layers diminishing resolution (which would have allowed to compute outputs with same resolution as inputs). As a trade-off, they doubled the resolution obtained after atrous convolution thanks to a bilinear interpolation.

Enlarged field-of-view

Also, atrous convolution allows to arbitrarily enlarge the field-of-view of filters at any layer. If the atrous convolution rate is r, $r-1$ zeros are introduced between filter values. Thus the new kernel size of a $k \times k$ filter is $k+(k-1)(r-1)$ without changing the number of parameters.

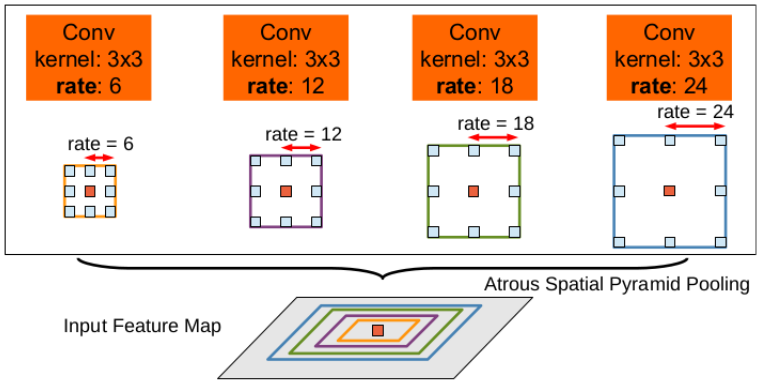


Figure 11: The center pixel (in orange) is the pixel being classified. Here four (atrous) convolution kernels are used in parallel. They have different rates, so each one has a different "field-of-view" [4]

Handling multiple scales

A classical way to train CNN to handle well objects of different scales is to augment the input with rescaled versions of the original images. The authors use here a different approach (originally by He et al. [13]): they use several atrous convolution kernels with different rates (each on a different branch of the network) and then merge the results. They name it *atrous spatial pyramid pooling* (*DeepLab-ASPP*). A representation by Chen et al. is reproduced on figure 11. Here all the features have been extracted at a single scale, the variations in scale being captured thanks to the different field-of-view of each convolution kernel. Eventually, four parallel atrous convolutions with different atrous rates are applied on top of the feature map.

Concerning the postprocessing with fully-connected conditional random Fields (CRFs) as it is described in the article, we have not implemented it. The improvement with CRFs was expected to be minor [21], and it seemed more useful to spend time implementing DeepLabv3. Moreover we obtained some interesting results with DeepLabv2 without CRF postprocessing (cf. 6.1).

5.3.2 DeepLabv3

The definition of atrous convolution in [5] is the same as the one in [4]. That is, written in two dimensions:

$$\mathbf{y}[\mathbf{i}] = \sum_{k} \mathbf{x}[\mathbf{i} + r\mathbf{k}]\mathbf{w}[\mathbf{k}], \tag{5.3.2}$$

as illustrated by figure 12.

In [5] is introduced the following definition: *output_stride*, it is the ratio of the input image spatial resolution to the spatial resolution of the final ouput. With this definition, the transformation of the classification network to deal with a segmentation task can be described this way: if at the end of the convolutional part of a classification network the *output_stride* is for example 32 (i.e. the final feature output is 32 times smaller than the image), one can double the density

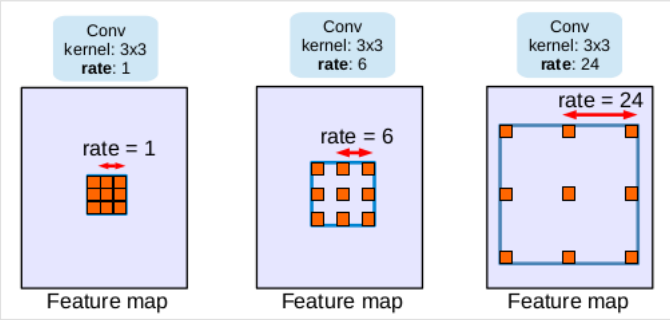
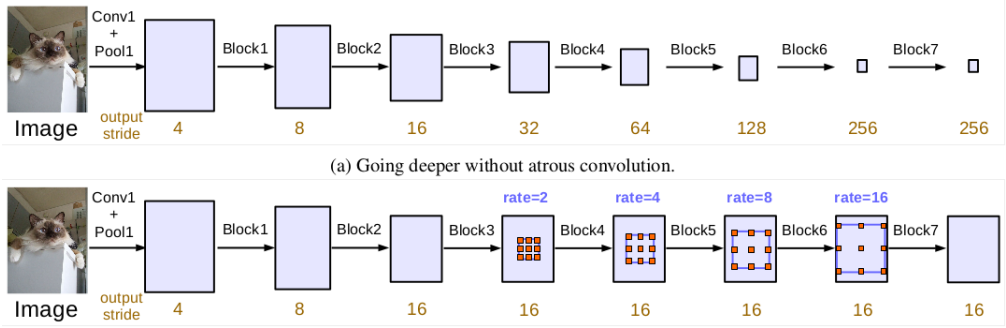


Figure 12: 2-dimension atrous convolutions [5]



(a) Going deeper without atrous convolution.

(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

Figure 13: Cascading atrous convolutions with increasing rates in order to keep the number of weights constant [5]

of the final feature output ($output\_stride = 16$) by setting the stride of the last pooling or convolutional layer that decreases resolution to 1 and replacing all subsequent convolutional layers with atrous convolutional layers having rate r = 2. This will also keep a same number of parameters.

The paper [5] describes the two architectures they have developped, both using atrous convolutions and both being derived from the ResNet network: atrous convolutions laid out in cascade or atrous spatial pyramid pooling. (None of them is using CRFs anymore).

atrous convolutions in cascade

Figure 13 illustrates this architecture. In (b), Block5, Block6, and Block7 are replicas or ResNet's Block4, with modified atrous convolution rate. This prevents the *output_stride* from going over 16.

We have not implemented this architecture.

Atrous Spatial Pyramid Pooling (ASPP)

The ASPP of DeepLabv3 [5], shown in figure 14, is an evolution of the ASPP designed for DeepLabv2 [4].

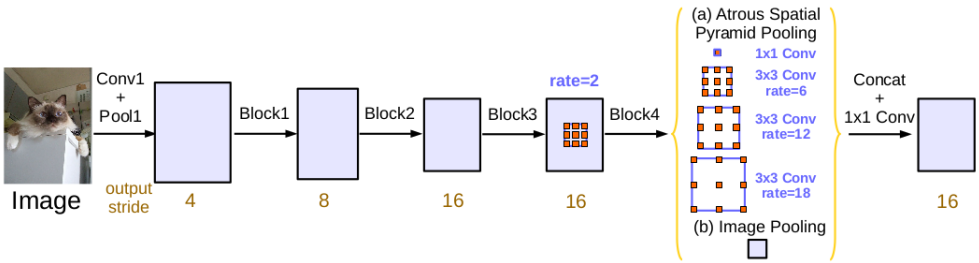


Figure 14: DeepLabv3, [5]

They propose here a new modification of ResNet [14]. The issue they want to fix is the following: as the sampling rate becomes larger, the number of valid kernel weights (i.e., the weights that are applied to the valid feature region, instead of padded zeros) becomes smaller. This can be seen for example on figure 12. If the rate keeps growing, in the extreme case where the rate value is close to the feature map size, the atrous convolution degenerates to a simple 1 x 1 kernel (only the center kernel weight is effective).

They solve this problem with *image level features* [18] [36]. See figure 14:

- part (a) (all convolutions with 256 filters and batch normalization)
  - one 1x1 convolution
  - three 3x3 convolutions with rates = (6, 12, 18) when output stride = 16
- part (b)
  - global average pooling on the last feature map of the model, and feeding the resulting image-level features to a 1 x 1 convolution with 256 filters (and batch normalization)
  - bilinearly upsample the feature to the desired spatial dimension
- concatenation of the resulting features from all the branches
- finally, another 1 x 1 convolution, also with 256 filters and batch normalization is performed.

Image-level Features

The key to capturing global context is *global average pooling*, added in DeepLabv3, which produces *image level features*.

The idea comes in particular from [18] (figure 15) and global average pooling is for example detailed in [7] (figure 16). As we can see on this figure, global average pooling is by itself rather simple. If we write the last feature map as a tensor, each slice of this tensor correspond to the transformation produced by a convolution kernel from the previous layer. In the global average pooling step, each of these slices is averaged into one number. It is simple, but the vector obtained allows carrying information about the global context: this vector is concatenated with the branches of the atrous spatial pyramid pooling (figure 14).

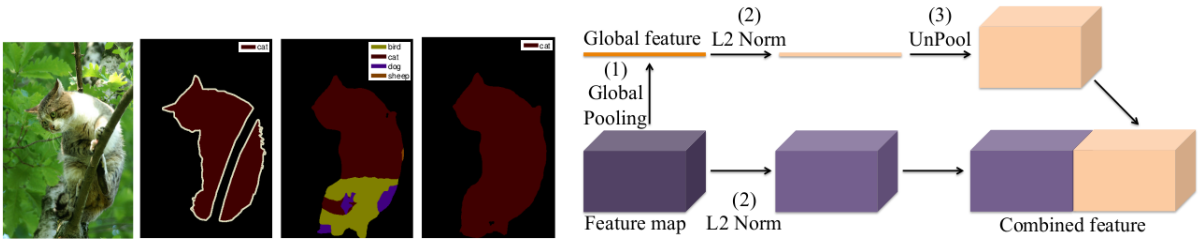


Figure 15: ParseNet uses extra global context to clarify local confusion and smooth segmentation [18]

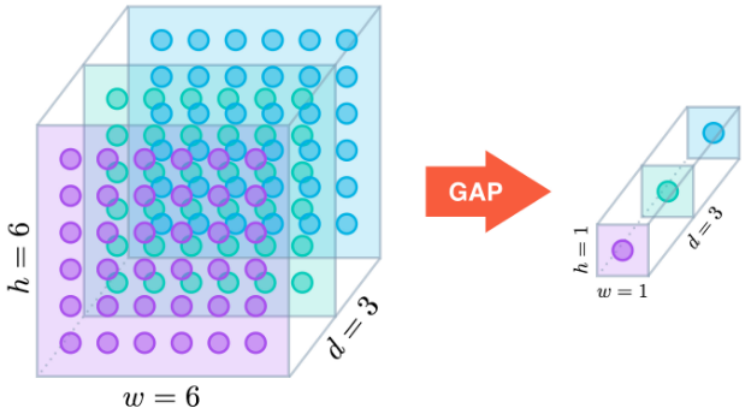


Figure 16: Global Average Pooling, [7]

In particular, this carrying of information about the global context aleviates the problem mentionned at the beginning: that with increasing the atrous rates, we end up with convolutions using only one pixel.

5.4 Pytorch

We used a pytorch (`https://pytorch.org/`) implementation of DeepLab. Our code is derived from the code publicly shared on github by Kazuto Nakashima [21]. In particular, we have modified the code so that it handles one-channel images (our MRI images are one channel grayscale images whereas Nakashima's code is designed to handle the usual RGB images of the Cocostuff dataset). Another of the modifications of the code was performed so that it works well when the number of classes is reduced to two (in our case contour and background). We also completed the DeepLabv3 training part of the implementation.

6 Results

Both DeepLabv2 and DeepLabv3 provided results. We stayed close to the original structure of the networks as described by [4] and [5]. We only divided the data set into a training set and a test set, so either the hyperparemeters remained constant or they were automatically modified.

One key element to have our trainings lead to good models was to use this parameter

```
weight = torch.tensor([.95, .05])
```

in the loss definition :

```
criterion = CrossEntropyLoss2d_cardiotraining(
    weight = torch.tensor([.95, .05])
      # the first class (0) is black, scarce and important
      # the second class (1) is white, all the rest
)
```

Indeed, for us, there are only two classes, *contour* or *background*. As there are much less contour pixels than background pixels (even when we consider, as we did, the entire area between the two orange lines in figure 8 to be contour pixels), we used the following compensation during training: we assigned weights to the two classes during training: 0.95 to contour pixels, and 0.05 to the other pixels.

The training was performed on a Geforce Titan X GPU, with a batch size of 2, and each iteration would last about 4.3 seconds.

In the following analysis of the results, we are not discussing the few images that the model could not even draw contours on, i.e. images for which the model labelled all the pixels as 1 (background) and none as 0 (contour). In a follow-up work, these images (probably way too dark) should either be removed or prepreprocessed so that their histograms are similar to the histograms of the other images.

6.1 DeepLabv2

Following are the best results we obtained by training a DeepLabv2 model on our dataset. (There is no CRF post-processing here*.)

The details of the DeepLabv2 architecture are given section 5.3.1. The atrous rate of the ASPP pyramid were set to: pyramids=[6, 12, 18, 24], as in the original paper. The learning rate was set to $2.5e-4$ and was not automatically adapted throughout learning (the few attempts to do so led to worse results). The batch size was 2 and the number of iterations $100,000$. So the computation lasted for about 120 hours. The performance metrics are in table 1.

*On the performance results he provides [21], Nakashima obtains for example a 1% improvement of the mean IoU on the COCO-Stuff 164k dataset. Insted of adapting the CRF implemented in the code to one-channel images, we prefered working on the DeepLabv3 implementation, likely to provide a much more significant improvement.

Mean IoU	Mean Dice	Mean Acc	FreqW Acc	Overall Acc
0.69	0.78	0.77	0.98	0.99

Table 1: DeepLabv2: best performance metrics we obtained

Since the contour (class 0) has a much smaller area than the background (class 1), we are also interested in these performance metrics considering each class separately (table 2).

IoU (class 0)	IoU (class 1)	Dice (class 0)	Dice (class 1)
0.39	0.99	0.56	0.99

Table 2: DeepLabv2: best performance metrics we obtained, by class (0: contour, 1: background)

We would like to have IoU (class 0) over 0.5, but 0.39 is an interesting result.

Figure 17 is plotting two performance indices, IoU (class 0) and Dice (class 0), through training. And we can see that the last $25,000$ iterations do not increase the performance. We could have used the $75,000$ iterations checkpoint of this model.

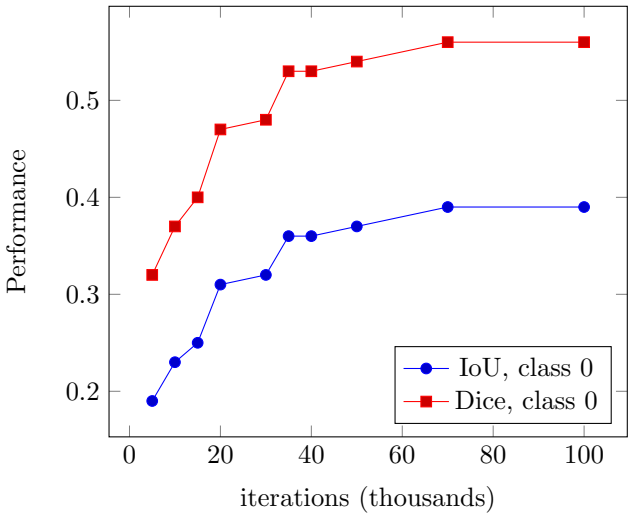


Figure 17: DeepLabv2 performance through training

Here are some significant examples of the results we obtained. First, an example of good result on the training set is on figure 18. We see that the model is fitting well the ground truth, although with some overemphasis. On the train set, most of the results are that good. On figure 19, there is an example of one of the not so good results on the trainset. On the test set, figure 20, is an example of a good fit, figure 21 provides two examples of not so good results.

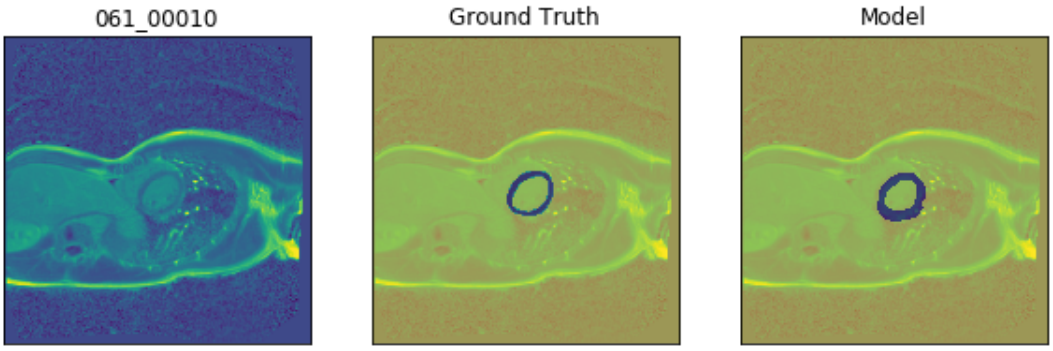


Figure 18: DeeLabv2, training set, example of a good fit

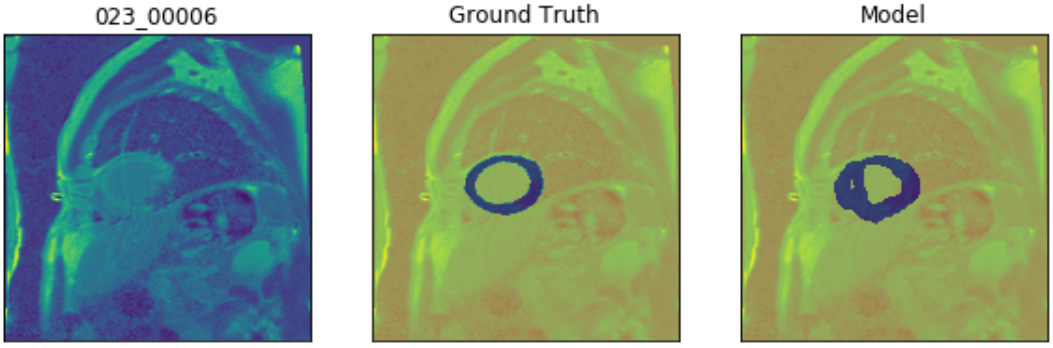


Figure 19: DeeLabv2, training set, example of one of the worst fit

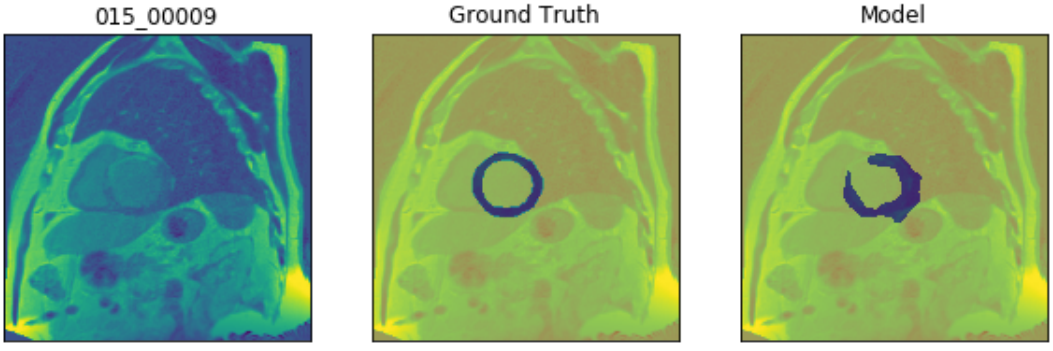


Figure 20: DeepLabv2, test set, example of one of the good fits

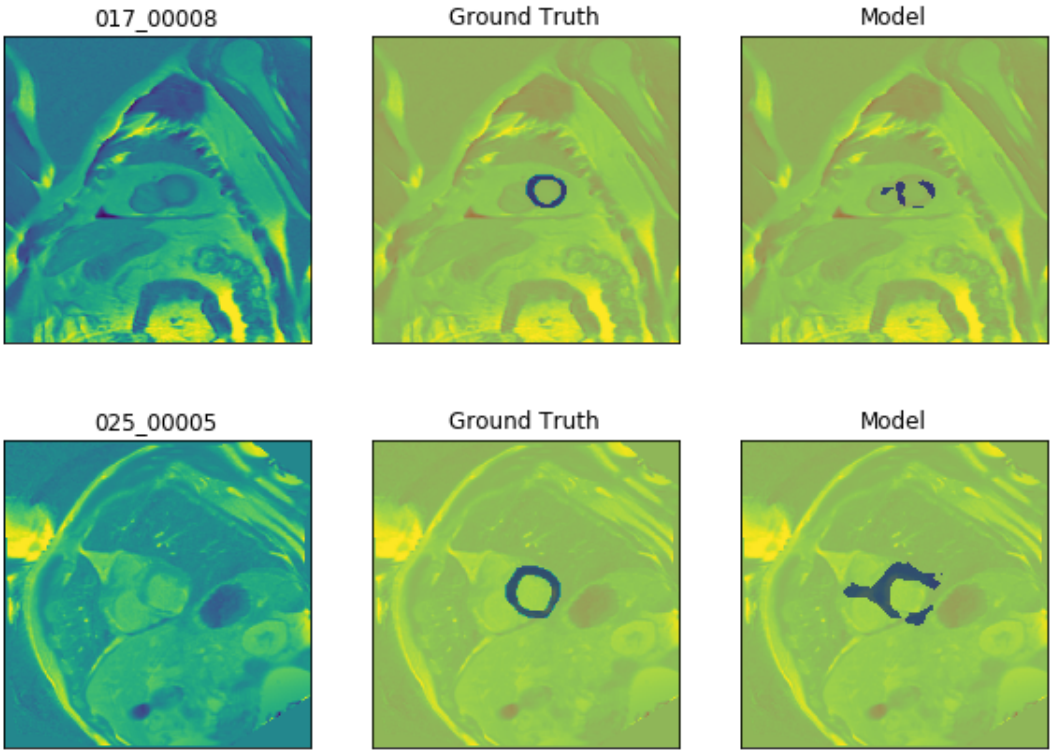


Figure 21: DeepLabv2: examples on the test set, not a good fit

6.2 DeepLabv3

Following are the best results we obtained by training a DeepLabv3 model on our dataset.

The details of the DeepLabv3 architecture are given section 5.3.2. We implement the architecture described in the original paper [5]. In particular, pyramids=[6, 12, 18]. For the first 15,000 iterations, the learning rate was set to $2.5e-4$ and was not automatically adapted throughout learning. Figure 22 plots the performance through this first part.

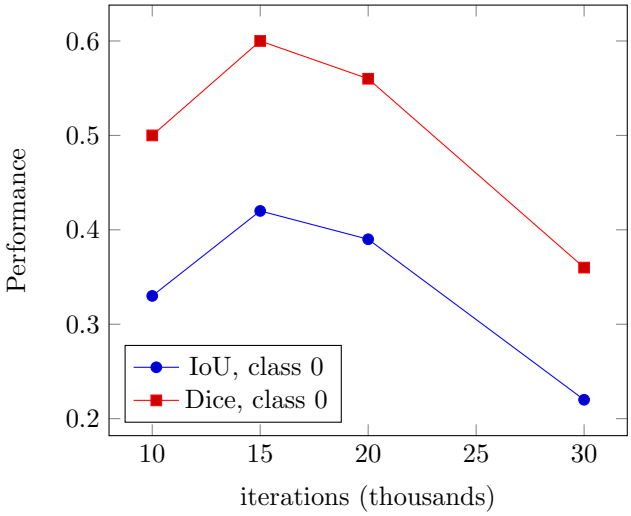


Figure 22: DeepLabv3 performance through training, first part

We used the 15,000 iterations checkpoint as the new starting point since performance declines after 15,000 Iterations. Then, for the following 20,000 iterations, the learning rate was automatically decreased by multiplying it by

$$\left(1 - \frac{iter}{max\_iter}\right)^{power} \tag{6.2.1}$$

with power = 0.9. The performance through training is plotted figure 23. We keep the 10,000 iterations checkpoint, after which performance declines. The batch size was 2. In each part of the training, the computation time for one iteration was 4.3 seconds, so the 25,000 iterations lasted about 30 hours. The performance metrics are in table 3.

Mean IoU	Mean Dice	Mean Acc	FreqW Acc	Overall Acc
0.72	0.81	0.90	0.98	0.99

Table 3: DeepLabv3: best performance metrics we obtained

We are also interested in these performance metrics considering each class separately since the contour (class 0) has a much smaller area than the background (class 1) (table 4).

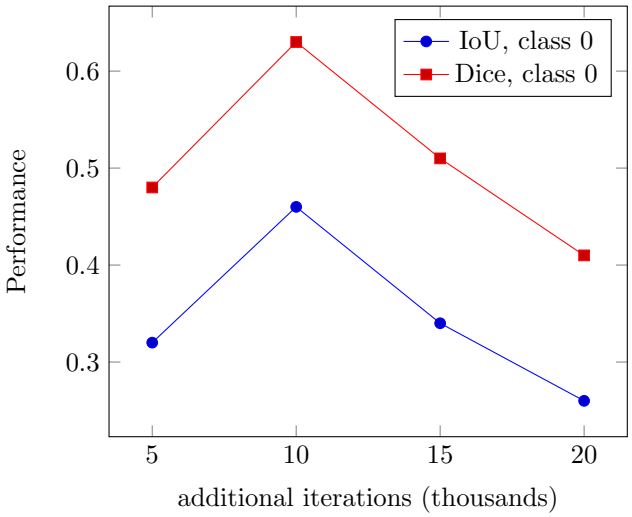


Figure 23: DeepLabv3 performance through training, second part

IoU (class 0)	IoU (class 1)	Dice (class 0)	Dice (class 1)
0.46	0.99	0.63	0.99

Table 4: DeepLabv3: best performance metrics we obtained, by class (0: contour, 1: background)

DeepLabv3 clearly obtains better results than DeepLabv2 (for example IoU (class 0): 0.46 versus 0.39). This is consistent with the results obtained on benchmark data sets in [4], [5], and [21]. We can visualize these results with DeepLabv3 on the same images as those used to evaluate DeepLabv2, figure 24 for the training set, and figure 25 for the test set. To make this comparison easier, figure 27 puts next to each other DeepLabv2 and DeepLabv3 results (test set examples).

These results seem promising. However, there are some images or sets of images not yet adequately handled by DeepLabv3 (figure 26).

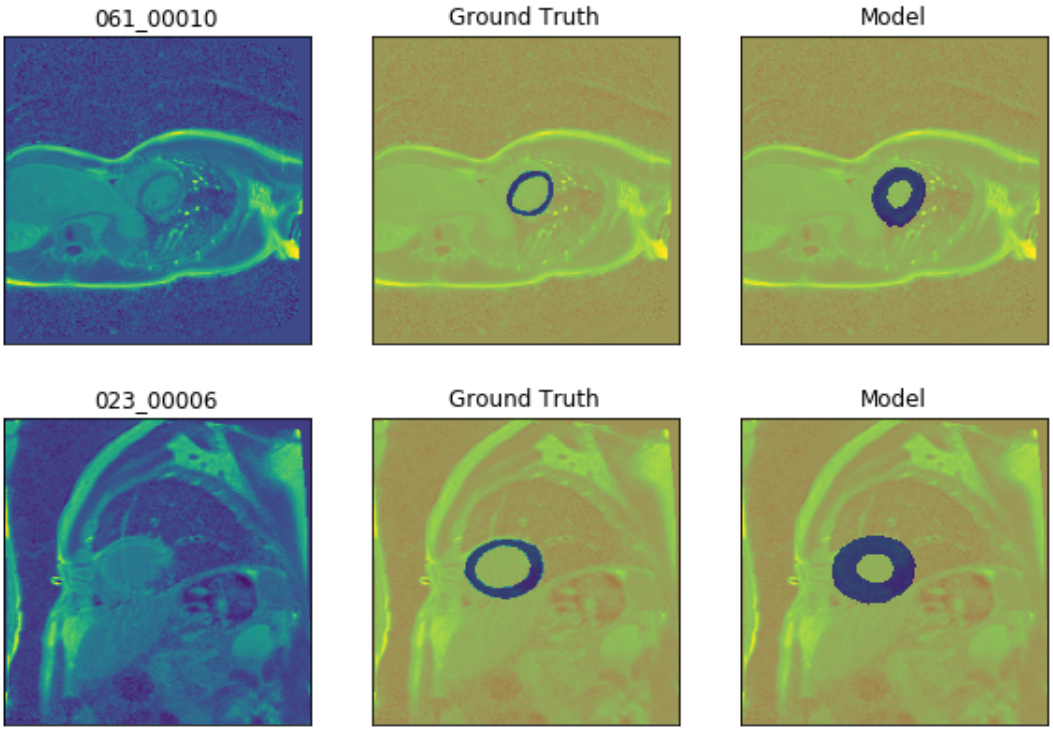


Figure 24: DeepLabv3: examples on the training set (same as those used for DeepLabv2, see figure 18 and figure 19)

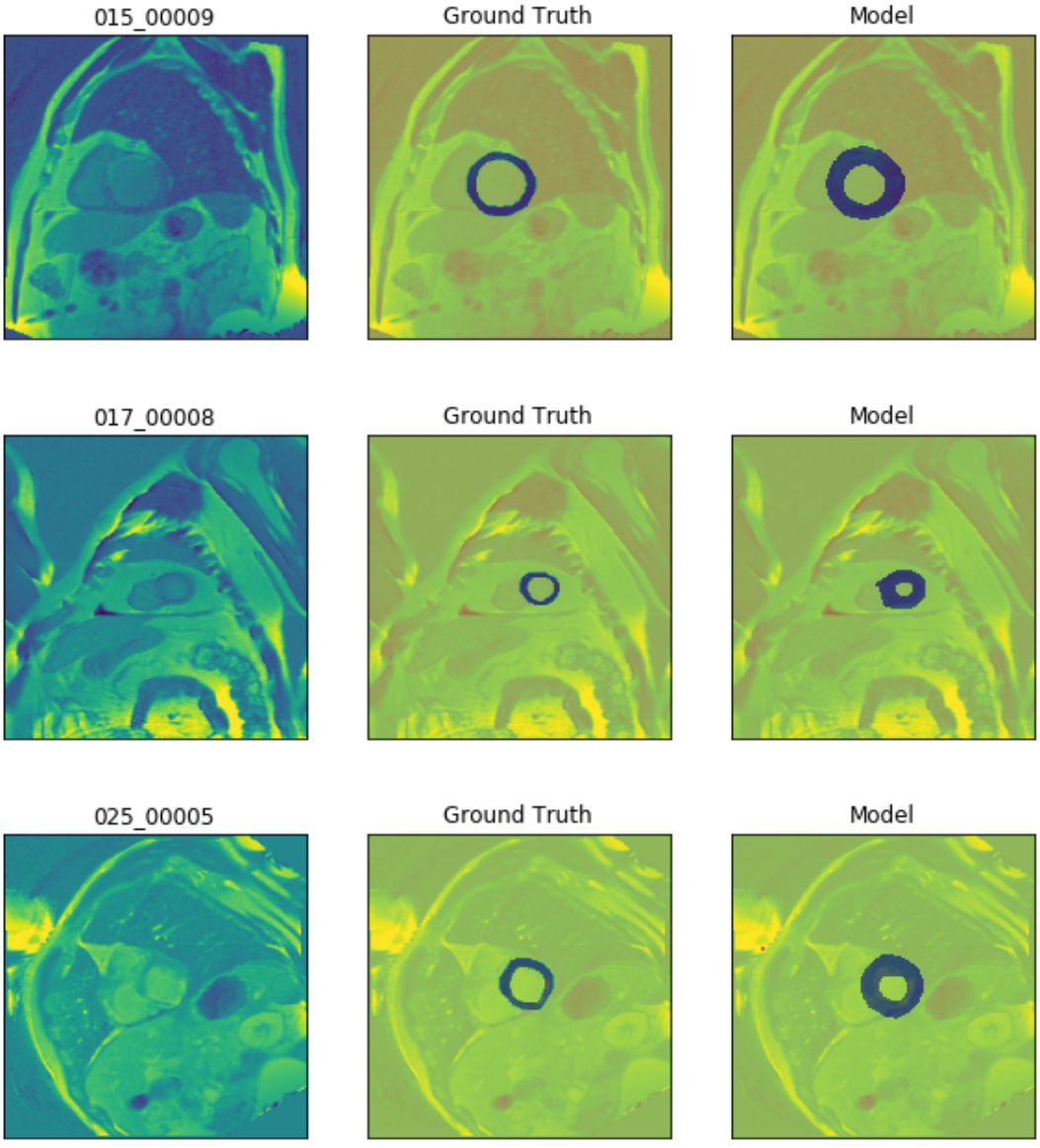


Figure 25: DeepLabv3: examples on the test set (for comparison with DeepLabv2 see figures 20 and 21)

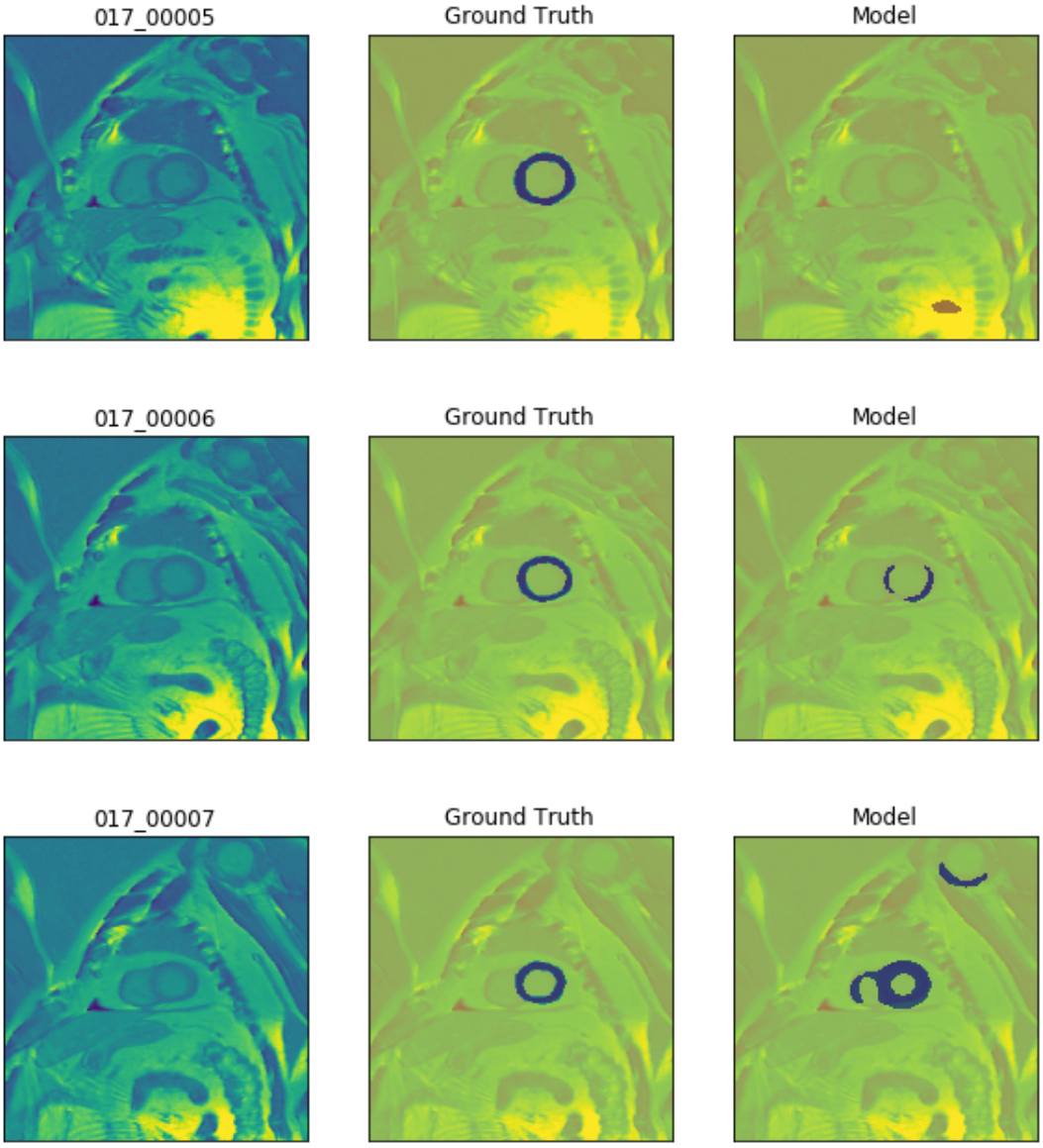
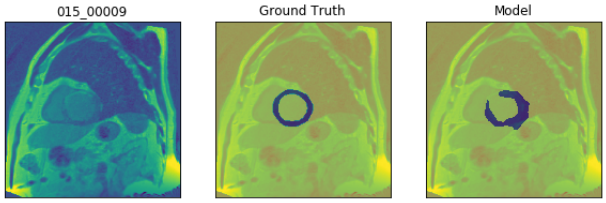
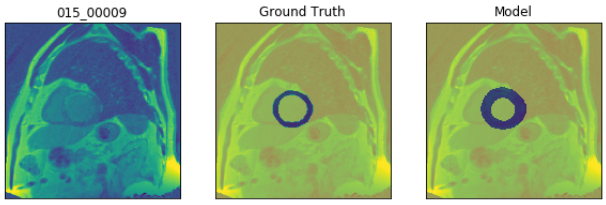


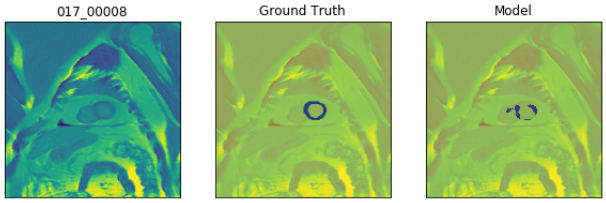
Figure 26: DeepLabv3: problematic examples on the test set



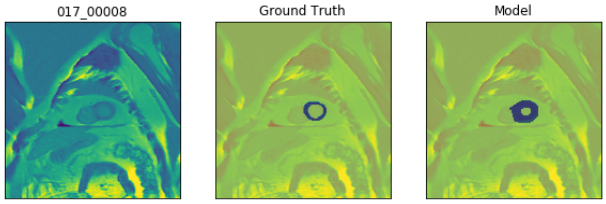
(a) DeepLabv2, image 015_00009



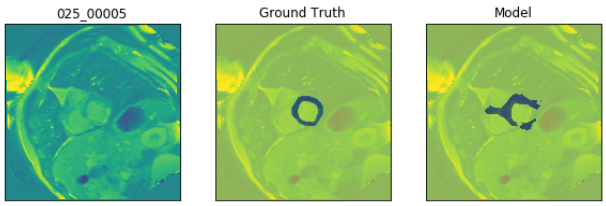
(b) DeepLabv3, image 015_00009



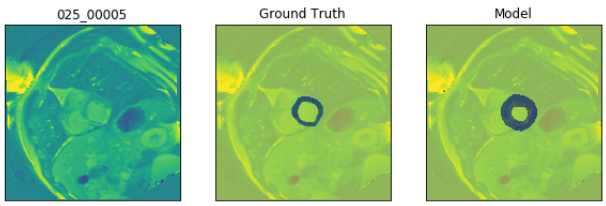
(c) DeepLabv2, image 017_00008



(d) DeepLabv3, image 017_00008



(e) DeepLabv2, image 025_00005



(f) DeepLabv3, image 025_00005

Figure 27: Test set: comparison between DeepLabv2 and DeepLabv3, a few examples

6.3 Comparison and Analysis

The improvement from DeepLabv2 to DeepLabv3 is probably due to the image level features created by global average pooling (5.3.2). Indeed, we can see a comparison of the two situations, with and without global average pooling, on figure 15, by Liu et al [18]. On (c) is the result obtained by a regular FCN (Forward Convolutional Network) and on (d) the result they obtained with ParseNet, which specifically uses global average pooling to capture global context, and which was one of the inspirations for DeepLabv3 [5]. It looks like this improvement is comparable to the improvement from DeepLabv2 to DeepLabv3 (see figure 27, which puts a few DeepLabv2 and DeepLabv3 results (on the test set) next to each other).

7 Future Work

7.1 Image Preprocessing

It would possible to further standardize the images (we have just substracted the overall mean pixel value). They could first be standardized patient by patient and then accross patients. This additional image preprocessing should allow to detect and transform the few problematic (too dark) images.

7.2 Hyperparameters

In order to fine tune hyperparemeters while training, it could be useful to transform the code so that it handles a split in three of the data: training set, validation set, test set. We have used a trainval set / test set split, and either not modified the hyperparemeters or modified them automatically (for example using an automactic learning rate decay). By using a validation set, the hyperparameters can be re-evaluated during training. However, as our dataset is not large, the benefit might be limited.

7.3 Network Architecture

And of course, we used the well-tested solutions provided by [4] and [5]. With time and care, it can be interesting to modify the overall architecture, and adapt it specifically to our project. However, it is probably hard to outperform the solutions described in these two articles, even if they are tested on general benchmark datasets.

8 Conclusion

The results obtained are consistent with those obtained by Chen et al. [4][5]. They show that this neural network architecture designed for semantic segmentation and originally benchmarked on standard datasets works well on a more specific semantic segmentation task. Moreover, according to our first results, it seems that the adaptation of the DeepLab network architecture and its implementation code proposed here can achieve the desired segmentation of heart MRI images.

9 Acknowledgements

I would like to thank Raphaël Couturier and Michel Salomon for guiding me throughout the project. From the beginning, their help proved invaluable. They helped me shape this project such that it remained manageable. Crucially Professor Couturier gave me access to the medical data set, and provided an initial script handling DICOM format images and MATLAB format contours. He also asked key questions after reading an initial version of this manuscript. Professor Salomon gave me key advice on how to choose a machine learning network, and he provided a careful analysis of an initial version of this manuscript.

I am also very grateful to both of them for providing me an with an access to their GPUs in their lab in Belfort.

Thank you very much to Kazuto Nakashima for sharing his pytorch implementation of DeepLab on github. His code, clean and easy to read was the foundation of our implementation.

This master's thesis is a module of the computer science degree proposed by the CTU (Centre de Télé-enseignement Universitaire) of the University of Franche-Comté, France. I would like to thank all, professors and staff, who make it happen, and in particular Fabien Peureux, always efficient and available.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," arXiv preprint arXiv:1511.00561, 2015.

[2] X. Bian, S. N. Lim, and N. Zhou, "Multiscale fully convolutional network with application to industrial inspection," in Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on. IEEE, 2016, pp. 1–8.

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, 2015, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in ICLR, 2015

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, 2016, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs.* `https://arxiv.org/abs/1606.00915v2` (Submitted on 2 Jun 2016 (v1), last revised 12 May 2017 (this version, v2))

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation* , (Submitted on 17 Jun 2017 (v1), last revised 8 Aug 2017 (this version, v2)), `https://arxiv.org/abs/1706.05587v2`

[6] Sasank Chilamkurthy, A 2017 Guide to Semantic Segmentation with Deep Learning, July 5, 2017, `http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review`

[7] Alexis Cook, `https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/`

[8] Vincent Dumoulin, Convolution arithmetic, `https://github.com/vdumoulin/conv_arithmetic`, in particular `https://github.com/vdumoulin/conv_arithmetic/blob/master/gif/dilation.gif`

[9] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, Jose Garcia-Rodriguez, 2017, *A Review on Deep Learning Techniques Applied to Semantic Segmentation*, `https://arxiv.org/abs/1704.06857`

[10] Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016, *Deep Learning*, MIT Press, `http://www.deeplearningbook.org`

[11] TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, `https://arxiv.org/abs/1603.04467v2`

[12] Hayit Greenspan, Bram van Ginneken, Ronald M. Summers, 2016, *Guest Editorial: Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique*, IEEE Transactions on Medical Imaging (Volume: 35, Issue: 5, May 2016), `https://doi.org/10.1109/TMI.2016.2553401`

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition (Submitted on 18 Jun 2014 (v1), last revised 23 Apr 2015 (this version, v4)) https://arxiv.org/abs/1406.4729

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv:1512.03385, 2015.

[15] Daphne Koller, Probabilistic Graphical Models 1: Representation (week 3 : Markov Networks (Undirected Models)), Coursera Lectures, https://www.coursera.org/learn/probabilistic-graphical-models/home/week/3

[16] LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. IEEE Communications Magazine, 27(11), 41–46

[17] Yann LeCun, Yoshua Bengio & Geoffrey Hinton, 2015, *Deep learning* Nature 521, 436–444 (28 May 2015) doi:10.1038/nature1453 https://www.researchgate.net/publication/277411157_Deep_Learning

[18] Wei Liu, Andrew Rabinovich, Alexander C. Berg ParseNet: Looking Wider to See Better (Submitted on 15 Jun 2015 (v1), last revised 19 Nov 2015 (this version, v2)) https://arxiv.org/abs/1506.04579

[19] J. Long, E. Shelhamer, and T. Darrell, 2015, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.

[20] P. Moeskops et al., "Automatic segmentation of MR brain images with a convolutional neural network," IEEE Trans. Med. Imag., vol. 35, no. 5, pp. 1252–1261, May 2016.

[21] Kazuto Nakashima, PyTorch implementation of DeepLab (ResNet) + COCO-Stuff https://github.com/kazuto1011/deeplab-pytorch

[22] Sebastian Nowozin, Christoph H. Lampert, Structured Learning and Prediction in Computer Vision, January 2011, Foundations and Trends in Computer Graphics and Vision 6(3-4):185-365, DOI10.1561/0600000033

[23] L. G. Nyúl, J. K. Udupa, and X. Zhang, "New variants of a method of MRI scale standardization," IEEE Trans. Med. Imag., vol. 19, no. 2, pp. 143–150, Feb. 2000.

[24] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," 2016, https://arxiv.org/abs/1606.02147

[25] S. Pereira, A. Pinto, V. Alves, and C. Silva, 2016, "Brain tumor segmentation using convolutional neural networks in MRI images," IEEE Trans. Med. Imag., vol. 35, no. 5, pp. 1240–1251, May 2016.

[26] A. Raj, D. Maturana, and S. Scherer, "Multi-scale convolutional architecture for semantic segmentation," 2015

[27] M. Salomon, R. Couturier, C. Guyeux, J.-F. Couchot, J.M. Bahi, Steganalysis via a convolutional neural network using large convolution filters for embedding process with same stego key: A deep learning approach for telemedicine. Eur Res Telemed (2017), http://dx.doi.org/10.1016/j.eurtel.2017.06.001

[28] H.-C. Shin et al., "Deep convolutional neural networks for computer- aided detection: CNN architectures, dataset characteristics and transfer learning," IEEE Trans. Med. Imag., vol. 35, no. 5, pp. 1285–1298, May 2016.

[29] N. Tajbakhsh et al., "Convolutional neural networks for medical image analysis: Full training or fine tuning?," IEEE Trans. Med. Imag., vol. 35, no. 5, pp. 1299–1312, May 2016

[30] G. van Tulder and M. de Bruijne, "Combining generative and discriminative representation learning in convolutional restricted Boltzmann machines," IEEE Trans. Med. Imag., vol. 35, no. 5, pp. 1262–1272, May 2016.

[31] PASCAL VOC Challenge performance evaluation and download server, Segmentation Results, VOC2012http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=6

[32] Charles Sutton and Andrew McCallum, 2012, "An Introduction to Conditional Random Fields", Foundations and Trends in Machine Learning: Vol. 4: No. 4, pp 267-373. http://dx.doi.org/10.1561/2200000013

[33] Jason Wang , Luis Perez, The Effectiveness of Data Augmentation in Image Classification using Deep Learning https://arxiv.org/abs/1712.04621

[34] Fisher Yu, Vladlen Koltun, Multi-Scale Context Aggregation by Dilated Convolutions, conference paper at ICLR 2016, https://arxiv.org/abs/1511.07122

[35] M. D. Zeiler and R. Fergus, 2013, "Visualizing and understanding convolutional networks," https://arxiv.org/abs/1311.2901v3, in ECCV, 2014

[36] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia, Pyramid Scene Parsing Network (Submitted on 4 Dec 2016 (v1), last revised 27 Apr 2017) https://arxiv.org/pdf/1612.01105.pdf