Imperial College London

Imperial College London
Department of Mathematics

Segmentation of CT scans into Atrium/non Atrium

Thomas Vogel

CID: 01016217

Supervised by Prof Giovanni Montana

3rd September 2015

Submitted in partial fulfilment of the requirements for the MSc in Statistics of Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: Date:

Abstract

The object of this thesis is to implement a convolutional neural network (CNN) to automatically identify the atrium of the heart in Computerised Tomography (CT) scans of the chest. A brief overview of Deep Learning is given in general, and in the context of Medical Imaging in particular as well as some background material on CNNs. We then describe the implementation of our CNN using a multi-scale tri-planar approach followed by some experimental results from a hyper-parameter search for a good architecture and set of learning parameters. The final set of results describes an altering to the sampling procedure yielding a significant improvement to the classification rate. The final model has an mean classification accuracy of 0.985 across 7 test CT scans, but makes significant errors in the neighbouring regions around the atrium where it is expected and, more worryingly, at times also away from it.

Acknowledgements

I would like to thank Professor Giovanni Montana for supplying the computational resources needed for this project as well as Rudra Poudel, his research associate, for providing technical help when I needed it. This thesis was written during a summer riddled with personal difficulties including a severe sciatica which severely threatened its completion. Special acknowledgement goes to my mother, who nursed me for 6 weeks in July and August while I was bed bound and unable to walk, and to fellow student Irina Timoshenko for helping me overcome a number of deep psychological issues.

Table of Contents

1. Introduction

1.1. Deep Learning and Convolution Neural Networks

Deep Learning is a branch of Machine Learning referring to neural networks with many hidden layers. It has attracted a lot of attention lately [15] due to a string of recent successes, which has made it the state of the art in a number of international contests in pattern recognition, particularly in image recognition, speech recognition, and natural language processing [19].

The main difficulty in applying Machine Learning techniques to real world problems comes from the high dimensionality of the datasets encountered, severely increasing the learning complexity of the algorithms required to handle them and thereby reducing their ability to generalise well. Traditionally, practitioners tackle this problem by finding ways to reduce the number of dimensions in what is known as "feature extraction", a deliberate hand-engineered process of transforming the large number of variables into a smaller number of features with the same level of information. By contrast, neural networks aim to learn the relevant features as part of the learning process itself using a hierarchical network of simple processing units called neurons, or nodes, in order to extract a good representation of the data [2]. Learning in this context means finding a set of parameters known as weights which makes the neural network exhibit a particular desired behaviour.

Neural networks are not new, but until recently, deep multi-layered architectures couldn't be trained effectively using standard gradient descent methods based on the backpropagation algorithm from random initialisation (WHY?). In 2006, a seminal paper by Hinton [10] made the training of deep neural networks practical using an unsupervised pre-training approach. These networks called Deep Belief Networks, have lead to a resurgence in the interest in neural networks. Since then, a number of other practical mechanisms have been devised to train deep architectures without unsupervised pre-training [13], from more effective initialisation procedures [8] to the use of Rectified Linear activation function [9] or the introduction of dropout as a regularisation mechanism [20].

Another major recent development was the introduction of convolutional neural networks (CNNs) by LeCun [14]. CNNs are a specialised type of neural network for certain grid-like structures with localised features. In particular, they have been remarkably effective for classification tasks on images, videos, and audio recordings, providing state of the art performance on object and speech recognition tasks [11], [21], [12].

1.2. Deep Learning in Medical Imaging

The field of Medical Imaging encompasses a set of techniques to create visual representations of the interior of the body using technologies such as X-rays or ultra-sounds. Its aim is to provide non-invasive ways of revealing internal structures to help make diagnoses. Interpreting these images have traditionally been done by trained clinicians such as radiologist or histologists, involving a time consuming and expensive process. Machine Learning is now being used to help clinicians with this process [16] in order to improve the accuracy, speed and cost of diagnoses.

Following the string of recent success of Deep Learning approaches in various imaging contexts, there has been recent efforts to implement these classifiers in the context of Medical Imaging. They include for instance the anatomical segmentation of the entire brain [7], biological neuron membrane to map 3D brain structure and connectivity [5], and counting cell mitosis in histology images [6] for cancer screening and assessment, amongst others.

1.3. Goal of the Thesis

This Master's thesis aims to implement a CNN to identify the atrium of the heart by classifying each voxel as either being part of it or outside of it using a tri-planar multi-scale approach. The second chapter discusses some background material on neural networks and CNNs. The third chapter then describes the approach undertaken and presents a number of experimental results. We will finish with some conclusions and a discussion of various ways to improve upon our results.

2. Background Material: on Convolutional Neural Networks

We start with a brief review of feedforward neural networks before elaborating on convolutional neural networks (CNNs). A number of good books available expand on the following content in much greater detail, notably [3], [4] and [17].

2.1. Feed Forward Neural Networks

A feedforward neural network is a standard model in the machine literature for classification problems. It consists of a number of layers composed of simple processing units, also called neurons or nodes, in which each unit in a given layer is connected to a number of units in the previous layers, each connection characterised by a weight encoding the knowledge of the network. Layers in which every unit is only connected to every unit in the previous layer are called fully-connected layers. Each unit represents an activation value generated by passing a linear combination of the values of the incoming connections weighted by their weights to an activation function, such as a Rectified Linear, a Tanh or a Sigmoid function. The information encoded in the data enters at the input layer, gets processed as it passes through the network, layer by layer, until it arrives at the output layer. The choice of the activation function at the output layer is determined by the nature of the data and the assumed distribution of the target variables. For classification problems, the softmax function or, in our case, its log, give the output a probabilistic interpretation. This model can be represented in the form of a network diagram as shown in Figure 2.1.

Training a neural network is done by minimising an error measure of its performance over a training set using gradient-based optimisation routines. The error gradients with respect to the parameters that are needed for the minimisation procedure, can be efficiently evaluated via the standard backpropagation algorithm. To prevent overfitting, a number of regularisation methods are available, including the traditional ones such as L1 and L2 regularisation. Recently a method called dropout [20] where at every epoch, a percentage of hidden units are randomly deactivated during training has been shown to give better generalisation performance.

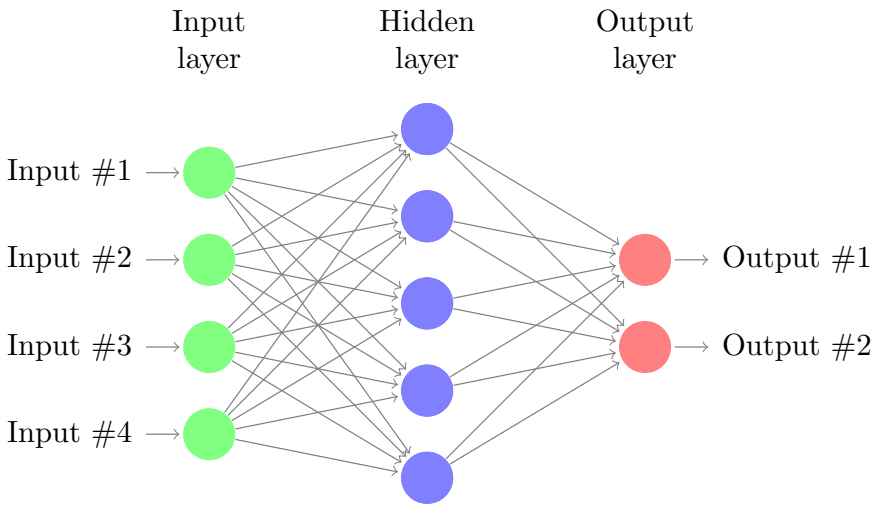

Figure 2.1.: A 1 layer-feedforward neural network with 4 input nodes, 1 hidden layer with 5 hidden nodes and 2 output node.

2.2. Convolutional Neural Networks

A CNN is a specialised kind of feedforward neural network where the first few layers of the architecture are so-called convolutional layers. These consist of three stages: a convolution stage, a detection stage and an optional subsampling stage. We will be discussing CNNs in the context of multi-channel images of size $m \times m$, where each pixel value represents an input node.

2.2.1. Convolution

The convolution stage is responsible for implementing k convolution operations over each channel of the input image. This is accomplished by convolving in parallel each channel with k kernels of size $s \times s$. Unlike fully connected layers, in a convolution, each output node is locally connected to a small square subset of corresponding input nodes determined by the kernel size. Additionally, as the same kernel is applied throughout the image, the convolution operation implies weight sharing, thus greatly reducing the number of trainable parameters. Figure 2.2 illustrates the convolution operation graphically. The set of output nodes resulting from one kernel is called a feature map, which is itself a rectangular arrays of nodes of size $(m-s/2+1) \times (m-s/2+1)$. Feature maps detect the presence in the input image of a particular feature encoded by the corresponding kernel. Having multiple kernels run through the input layer in parallel produces a set of feature maps. Together, they are responsible for detecting various types of features which might

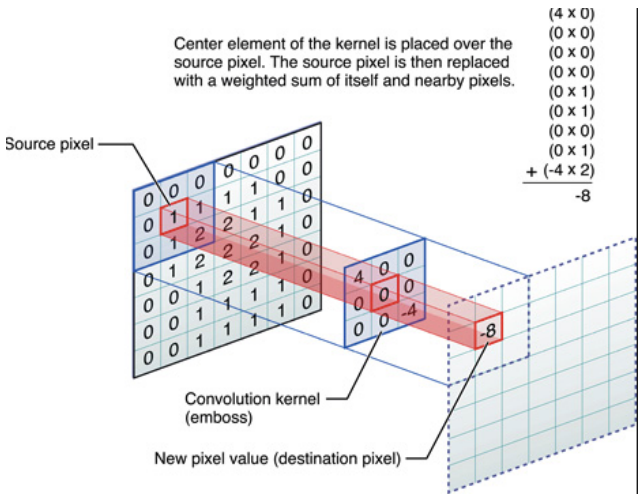

Figure 2.2.: The convolution layer. Illustration of the formation of a feature map taken from [1]

be present in the image.

The benefits of this approach are two folds :

- Computational efficiency: the sparse interaction between the hidden nodes and the weight sharing significantly decrease the number of parameters to train.
- Translational equivariance: shifting the input results in an equivalent shift in the output. This is a property of the convolution operation.

In the detector stage, every node in the feature map is then passed to an nonlinear activation unit exactly as in a standard neural network. These activation units usually consist of either Rectified Linear units (ReLU), Tanh units, or Sigmoid units.

Finally in the pooling stage, each feature map is then subsampled by aggregating the node values using a summary statistic over a rectangular neighbourhood of outputs. Two commonly used ones are the max and average pooling operations which report respectively the maximum and average of a set of inputs. Figure 2.3 shows a diagram of subsampling.

The benefits of subsampling are that it further reduces the number of features by a factor of k for pooling regions spaced k pixels apart, aiding the classification task and improving the computational efficiency of the network. Furthermore subsampling has the added benefit of making the representation become invariant to small translations of the input. Thus translating the input by a small amount results in little to no change in the values

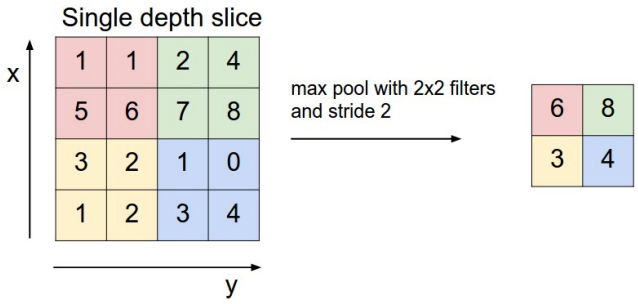

Figure 2.3.: Subsampling in action. Here maxpooling is used to halve the size of an feature map using a 2×2 filter

of the pooled outputs.

2.2.2. Typical Architecture

A typical convolutional neural network architecture consists of an input layer, a number of convolutional layers, a number of fully connected layers, followed by an output layer. The input layer is an $r \times n \times n$ image, with r being the number of channels and n the image height and width. Each channel is passed through a number of convolutional layers in parallel. Then the remaining output nodes from all the resulting feature maps across all channels are "flattened" and passed as input to a number of fully connected layers. The output layer classifies the input into one of a number of different classes. The convolutional layers thus serve as a way to reduce the dimensionality of the image by extracting meaningful local features.

3. Experimental Results

3.1. The Data

The aim of this thesis is to implement, and then fine-tune, a CNN to classify voxels of chest computerised tomography (CT) scan as being either in the part of the heart called the atrium or outside of it. The atrium is the entry point of the blood into the heart. It is composed of two chambers: a right one which recovers blood returning to the heart to complete the cardiovascular cycle through the body, and a left one receiving blood coming back from the lungs after being oxygenated.

The data from which the training and testing datasets are generated comprise 27 CT scans. CT scans are 3 dimensional grey scale images, generally of size 480*480*50, generated via computers combining many X-ray images taken from different angles to produce cross-sectional images of specific body parts. They are stored as DICOM files, DICOM standing for Digital Imaging and Communications in Medicine which is a standard for handling, storing, printing, and transmitting information in medical imaging. Each has been segmented by trained radiologists at St Thomas's hospital, the results of which are stored in Nearly Raw Raster Data (NRRD) files, a standard format for storing raster data. These are 3D arrays of the same dimension as their corresponding CT scan with each entry being either a 1 or 0 indicating whether its corresponding voxel belongs in the atrium or not.

3.2. Generating the Datasets: the Tri-Planar Method

Classifying the voxels requires building a set of input vectors each containing enough local and global information to allow the neural network to learn effectively. One way of providing 3 dimensional information is to use the tri-planar method. This consists in generating 3 perpendicular square patches of a given dimension in the transversal, saggital and coronal planes centred at the voxel of interest as shown in Figure 3.1. This technique has been found to give competitive results compared to using 3D patches while being much more computationally and memory efficient [18]. In addition, we use a multi-scale approach as in [7], where we add 3 more input channels composed of compressed patches that are originally 5 times larger than the above set of patches, but resized to be of the same size as the first 3 input patches to provide global information about the surroundings of the concerned voxel.

Each patch is fed into a different input channel of the CNN to a total of 6 channels.

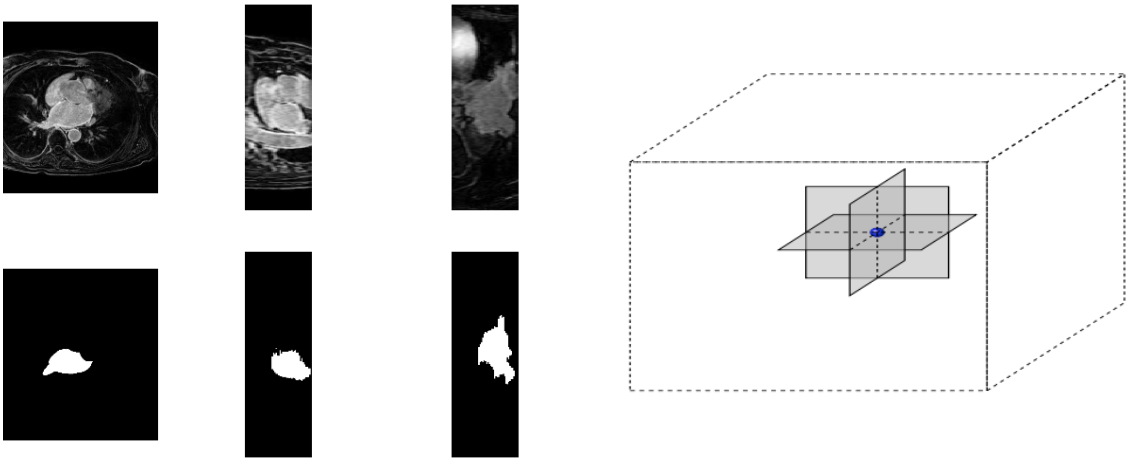

Figure 3.1.: Left: grey scale slices from a CT scan taken in the transversal, saggital and coronal planes. Right: illustration of the triplanar method.

The outputs of those channels are then feed as inputs to a set of connected layers itself connected to a classifying layer.

3.3. Implementation Details

3.3.1. Libraries

Our CNNs were implemented using Torch, an open-source library aiming to provide a Matlab-like environment for scientific computing in Lua, along with a number of dependent libraries (nn, cunn, cudnn, fbcunn) which facilitate the training of neural networks on single and multiple GPUs. In addition, we use Python and a number of its libraries to handle all the logistics ranging from generating the datasets to producing plots of segmentation results.

3.3.2. Computer facilities

The training of the CNNs were conducted alternatively on one of 2 multi-GPU clusters, named montana-nvidia and montana-k80, kindly provided by Prof. Montana. montana-nvidia consists of 24 cores with 129 Gb of memory connected to two NVIDIA Tesla K40m and two Tesla K20Xm while montana-k80 on the other hand has 56 cores with 258 Gb of memory supported by 8 of NVIDIA's Testla K80.

3.4. Model Selection

3.4.1. General Approach for Model Selection

In order to find a good set of hyper-parameters for our CNN, we allocate 20 of the 27 CT scans for generating the training set and the remaining 7 for generating the val-

idation set. The training set is composed of 400000 training examples equally divided among the training CT scans, half being in the atrium and the other half outside it. The validation set is composed of 200000 examples equally divided among the testing CT scans. Furthermore at the end of training, each model fully segments a test CT scan to provide performance statistics. From this segmentation are evaluated the model's sensitivity (the percentage of correctly classified atrium voxels), specificity (the percentage of correctly classified non-atrium voxels), and overall classification rate also known as the Dice coefficient, calculated by evaluating the proportion of correctly classified voxels in the segmented image. As 98% of the CT scan is composed of non-atrium voxels, the Dice coefficient is overwhelmingly influenced by the specificity. We will conduct our model selection by selectively choosing hyper-parameters that optimise the Dice coefficient while having a reasonable sensitivity above 0.8.

Additionally, mask images are generated from 3 transversal slices in the segmented CT scan. The masks are formed by overlaying the grey scale CT scan image with colours representing the error status of the classification of a given voxel. They provide a visual representation of the performance of the models. The colour codes are:

- Green: correctly classified atrium voxel.
- Blue: correctly classified non-atrium voxel.
- Red: incorrectly classified atrium voxel.
- Pink: incorrectly classified non-atrium voxel.

We start off our model selection with a CNN comprised of an input layer with 6 channels of patches of size 32*32, 2 convolutional layers with 32 and 64 feature maps respectively and a max pooling filter of size 2×2, 2 fully connected layers with 1000 and 500 hidden units each respectively, and a logsoftmax output layer giving the log probabilities of the voxel belonging to either classes. The training parameters are composed of a learning rate of 0.01, a momentum rate set to 0, and a mini-batch size set to 6000 examples. In addition, the negative log likelihood criterion provides the error measure during training. We will be varying in turn the following hyper-parameters while at each stage keeping the others constant.

- the number of convolutional layers.
- the number of connected layers.
- the number of feature maps in the chosen number of convolutional layers.
- the number of hidden units in the chosen number of connected layers.
- the learning rate.

- the momentum.
- the type of activation function (ReLU, Tanh or Sigmoid).

3.4.2. Varying the Number of Convolutional Layers

To select the number of convolutional layers, we train 4 CNNs with architectures starting with the following convolutional layers:

- Input (6*32*32) => Conv layer (32*28*28) => 2*2 MaxPooling filter (32*14*14)
- Input (6*32*32) => Conv layer (32*28*28) => 2*2 MaxPooling filter (32*14*14) => Conv layer (64*10*10) => 2*2 MaxPooling filter (64*5*5)
- Input (6*32*32) => Conv layer (32*28*28) => Conv layer (32*24*24) => 2*2 MaxPooling filter (32*12*12) => Conv layer (64*8*8) => 2*2 MaxPooling filter (64*4*4)
- Input (6*32*32) => Conv layer (32*28*28) => Conv layer (32*24*24) => 2*2 MaxPooling filter (32*12*12) => Conv layer (64*8*8) => Conv layer (64*4*4) => 2*2 MaxPooling filter (64*2*2)

The values in parentheses indicate the number and dimensions of the feature maps at each layer. The following table gives the results for each of these architectures trained over 100 epochs.

# Conv Layers	Sensitivity	Specificity	Test Dice Coefficient
1	0.913	0.970	0.969
2	0.734	0.989	0.984
3	0.762	0.987	0.983
4	0.881	0.973	0.972

The architecture with the best test Dice coefficient is the one with 2 convolutional layers, closely followed by the one with 3 convolutional layers, with Dice coefficients of 0.984 and 0.983 respectively. These two architectures give low sensitivities of 0.734 and 0.762. The architectures with 1 and 4 convolutional layers yield significantly lower Dice coefficients of 0.969 and 0.972 respectively, despite a much greater sensitivity of 0.913 and 0.881.

Figure 3.2 shows the masks from all 4 models generated after training. The first and fourth columns correspond to segmentations by the architectures with 1 and 4 convolutional layers respectively. These show much larger pink patches, corresponding to higher rates of false positives than the other two columns but with smaller red regions corresponding to lower rates of false negatives, corroborating the story told by the sensitivity and specificity in the table above. There is not much difference between the masks of the second and third layers.

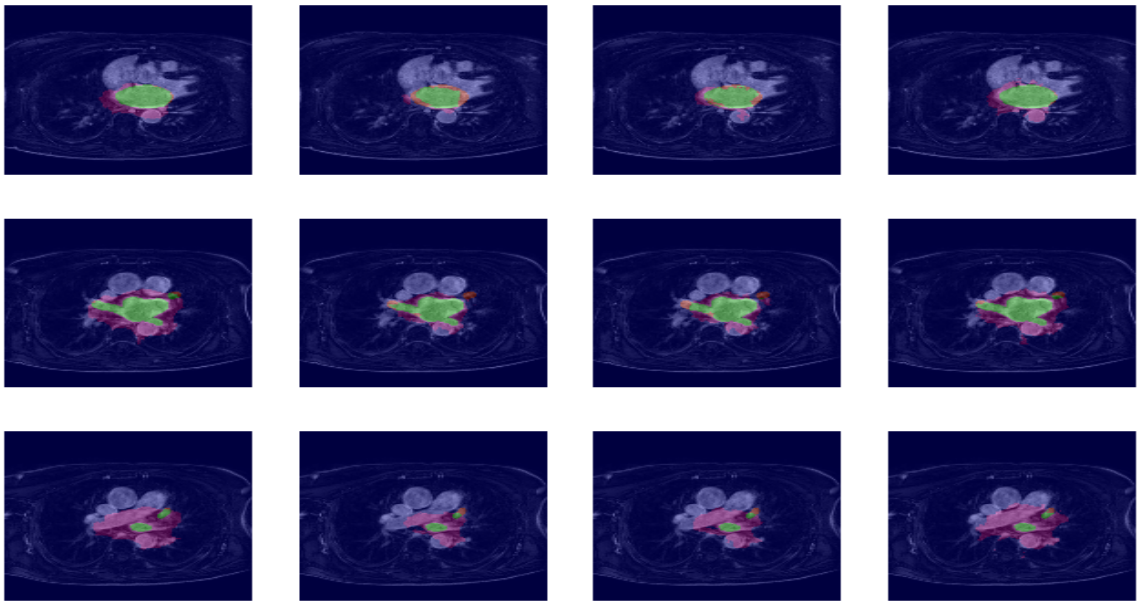

Figure 3.2.: Masks for varying number of convolutional layers. Each row displays the same mask generated by CNNs with from left to right, 1, 2, 3 and 4 convolutional layers.

We choose the architecture with the highest Dice coefficient at this stage and thus select an architecture with 2 convolutional layers.

3.4.3. Varying the Number of Connected Layers

Having settled on an architecture with 2 convolutional layers, we now train 3 CNNs with 1, 2, and 3 fully connected layers each starting with 1000 hidden units and halving the number of hidden units for each additional layer. Hence the CNN with 3 fully connected layers has 1000, 500, and 250 hidden units in each of its successive layers. The results are shown in the following table.

# Connected Layers	Sensitivity	Specificity	Test Dice Coefficient
1	0.887	0.972	0.971
2	0.856	0.974	0.972
3	0.915	0.967	0.966

All three architectures have similar performances on the test CT scan. The architectures with 1 and 2 connected layers in particular have the highest Dice coefficients of the three with marginally different specificities of 0.972 and 0.974 respectively. The discrepancies between their sensitivities are somewhat more pronounced at 0.887 and 0.856 respectively. Figure 3.3 similarly shows very little differences in the colour profile of the masks for all three architectures. Thus it seems that adding extra connected layers doesn't significantly

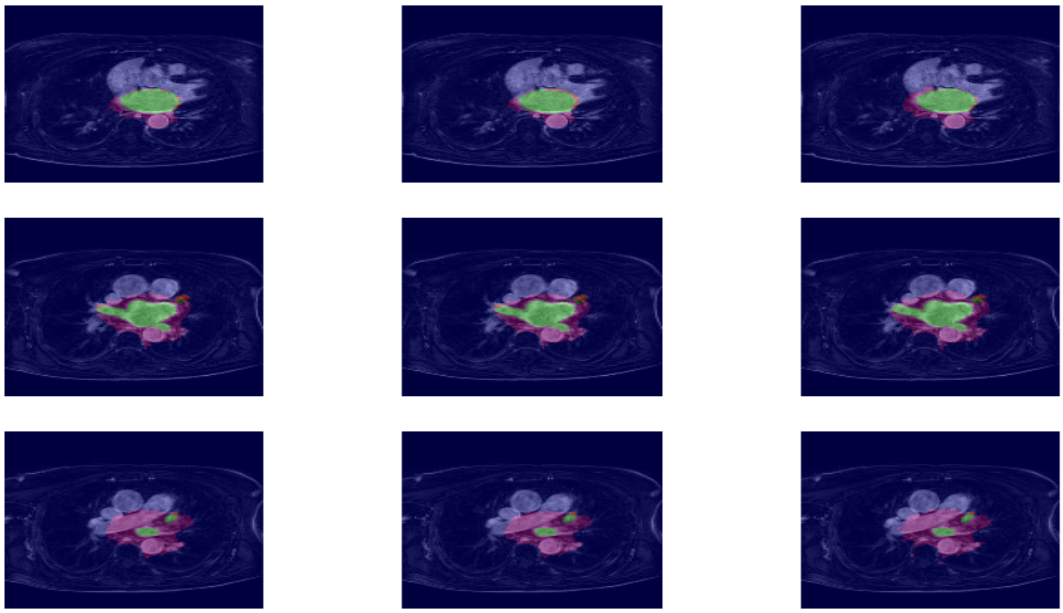

Figure 3.3.: Masks for varying number of connected layers. Each row displays the same mask generated by CNNs with from left to right, 1, 2, and 3 connected layers.

increase the performance of the classifier and hence we opt for having 1 fully connected layer in our final architecture.

3.4.4. Varying the Number of Feature Maps

We now focus on finding the right number of feature maps for the convolutional layers. In order to keep the computation comparable between the 2 layers, we set the number of feature maps in the second layer to be twice that of the first layer. We try configurations with the first layer having 16, 32, and 64 feature maps. The summary of the results are shown below.

Feature Maps	Sensitivity	Specificity	Test Dice Coefficient
16	0.919	0.967	0.966
32	0.894	0.972	0.970
64	0.904	0.971	0.970

Again the differences between the performance of all three architectures are minimal. The architectures with a first layer having 32 and 64 feature maps have the highest Dice coefficient both at 0.970 and very similar sensitivities around 0.9. The architecture with 16 feature maps yields a slightly lower specificity of 0.967 but a higher sensitivity of 0.919 with an overall lower Dice coefficient of 0.966. Figure 3.4 show slight differences in the first row of images, with a larger pink region for the 16 feature map architecture.

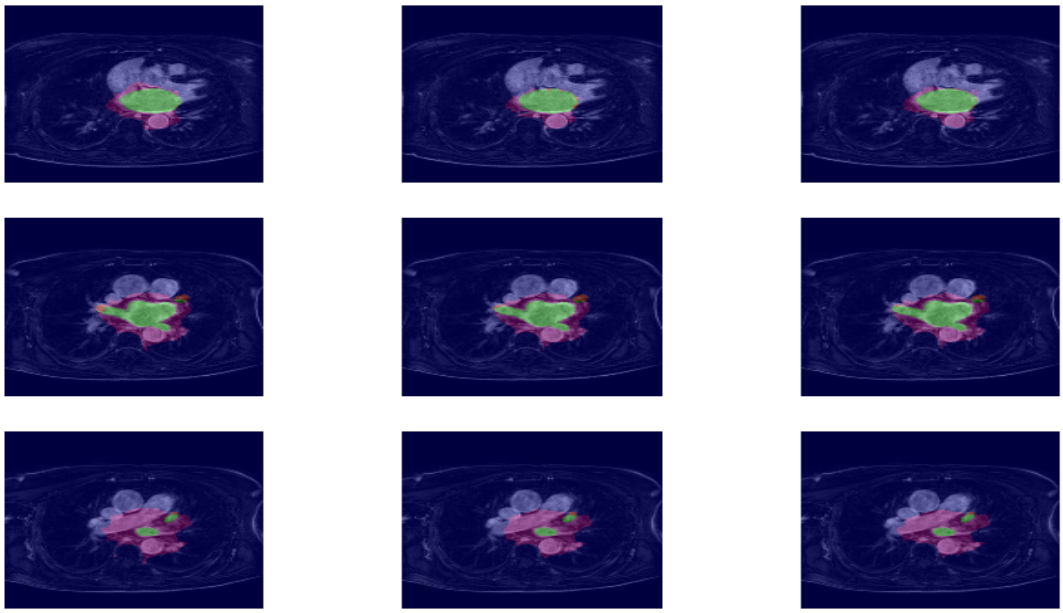

Figure 3.4.: Masks for varying number of feature maps. Each row displays the same mask generated by CNNs with from left to right, 16, 32, and 64 feature maps in their first convolutional layers.

As the highest test Dice coefficient belongs to the architecture with 32 and 64 feature maps, we opt for the simpler model and have 32 feature maps in the first layer of our final architecture.

3.4.5. Varying the Number of Hidden Units

We now vary the number of hidden units in the connected layer, trying 100, 200, 500, 1000 hidden units.

Hidden Units	Sensitivity	Specificity	Test Dice Coefficient
100	0.879	0.972	0.97
200	0.878	0.971	0.97
500	0.894	0.97	0.968
1000	0.892	0.972	0.97

The results across all 4 models are very similar. Indeed the architectures with 100, 200, and 1000 hidden units yield test Dice coefficients of 0.97 and the one with 500 hidden units has a slightly lower one at 0.968. Their sensitivities are also very close, being at 0.879, 0.878, 0.894 and 0.892 respectively. Figure 3.5 shows mask results with little differences between models as well.

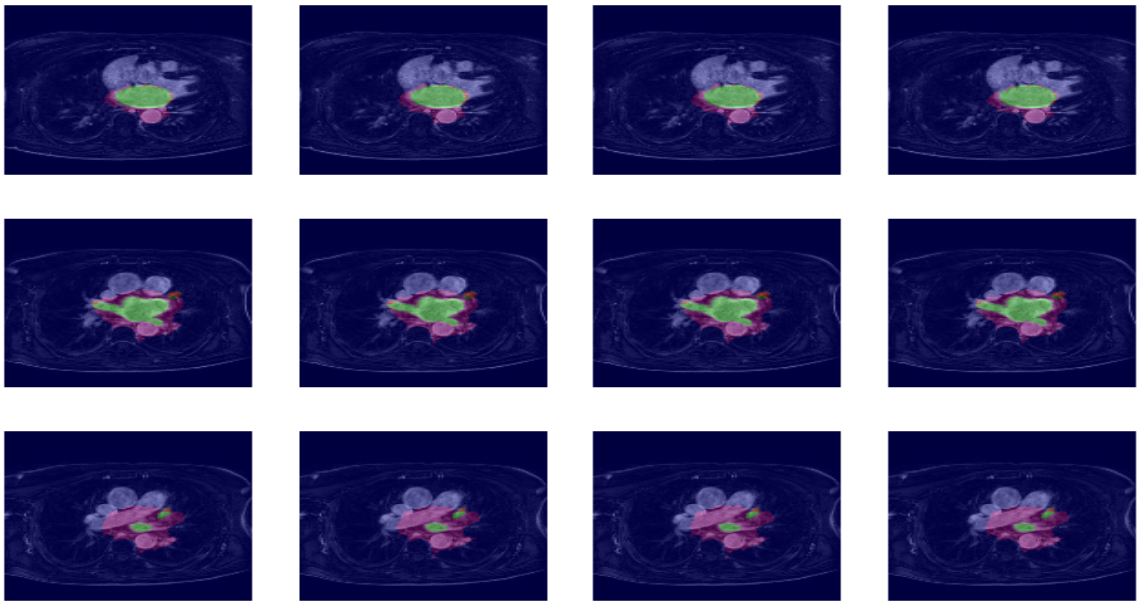

Figure 3.5.: Masks for varying number of hidden units. Each row displays the same mask generated by CNNs with from left to right, 100, 200, 500, and 1000 hidden units in the connected layer.

We choose the simpler model of the 4 and elect to have 100 hidden units in our final architecture.

3.4.6. Varying the Learning Rate

Having established our final architecture, we now turn our attention to choosing the learning rate. We try learning rates of 0.01, 0.05, 0.1, 0.5 and 1. The results are shown in the following table, although we omitted the results for a learning rate of 1 as the CNN didn't train.

Learning Rate	Sensitivity	Specificity	Test Dice Coefficient
0.01	0.885	0.971	0.97
0.05	0.915	0.977	0.976
0.1	0.916	0.98	0.978
0.5	0.932	0.982	0.981

The table shows improved results as the learning rate increases. The test Dice coefficient jumps substantially from 0.97 for a learning rate of 0.01 to 0.981 for a learning rate of 0.5. So does the sensitivity from 0.885 for a learning rate of 0.01 to 0.932 for a learning

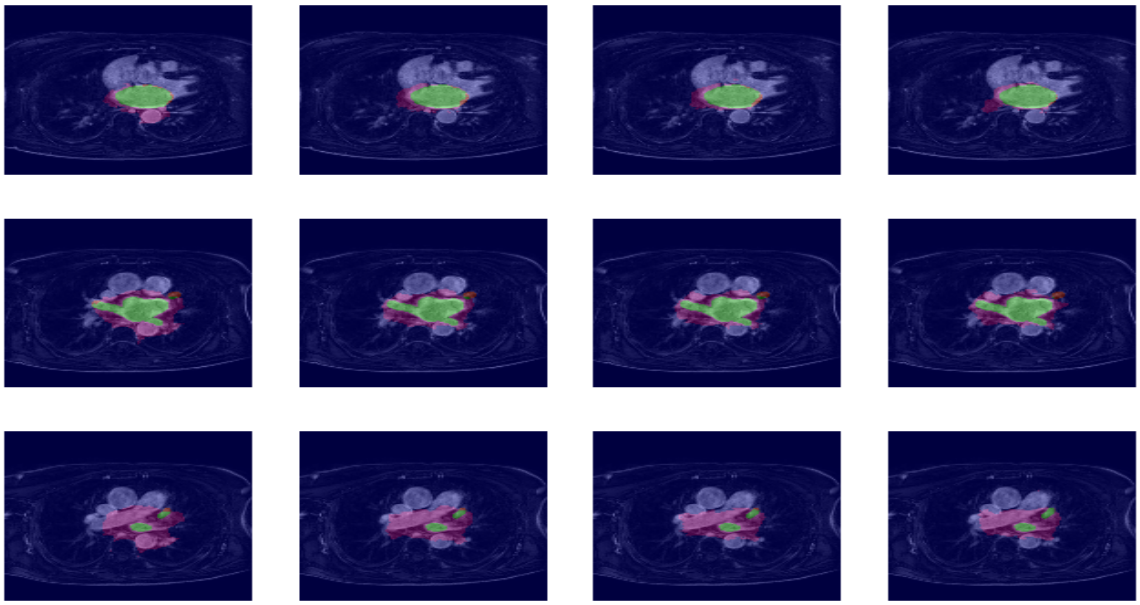

Figure 3.6.: Mask for varying learning rate. Each row displays the same mask generated by CNNs with from left to right, learning rates of 0.01, 0.05, 0.1 and 0.5.

rate of 0.5. The masks shown in Figure 3.6 reflect this improvement in accuracy with a substantial decrease of the size of the pink regions as the learning rate increases. The mark difference in performance illustrate the importance of setting an appropriate learning rate.

We thus choose a learning rate of 0.5 to train our selected architecture.

3.4.7. Varying the Momentum

We now train our CNN with a variety of momentums. We train with momentums set to 0, 0.05, 0.01, 0.5 and 1. The results are shown in the following table, albeit without those for a momentum set to 1 as the network didn't train.

Momentum	Sensitivity	Specificity	Test Dice Coefficient
0	0.906	0.982	0.981
0.05	0.916	0.98	0.979
0.1	0.9	0.984	0.982
0.5	0.83	0.988	0.985

The momentum yielding the best test Dice coefficient is the one set at 0.05 with an accuracy of 0.985, despite having the lowest sensitivity of 0.83. The test results for the other networks are fairly similar with test Dice coefficients of 0.981, 0.979 and 0.982,

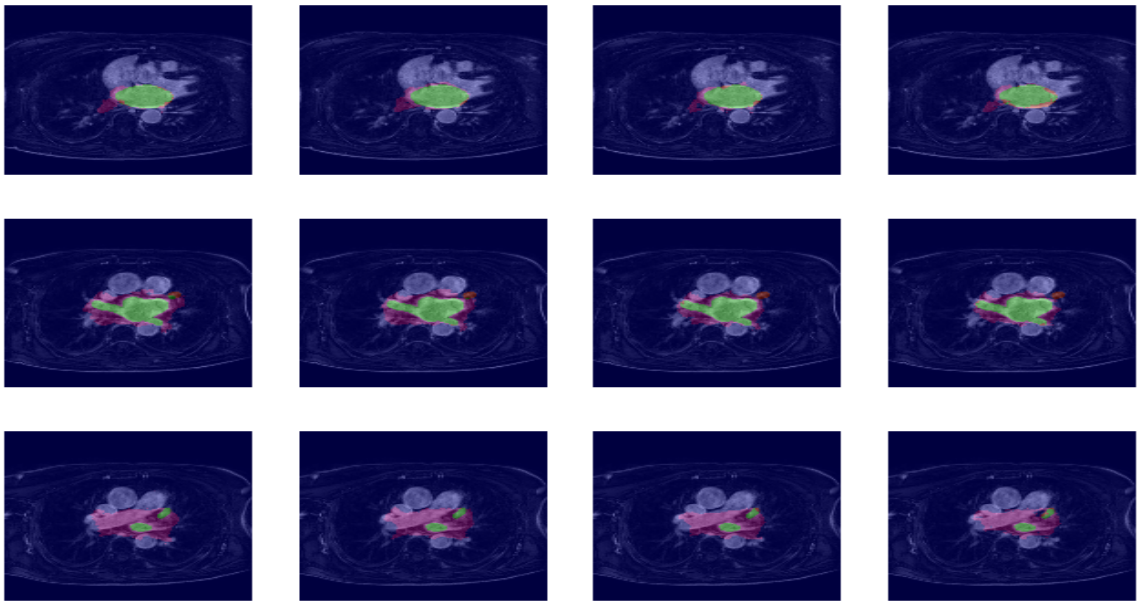

Figure 3.7.: Masks for varying momentum. Each row displays the same mask generated by CNNs with from left to right, momentum set to 0, 0.05, 0.1 and 0.5.

and sensitivities of 0.906, 0.916 and 0.9 for those with momentums of 0, 0.05 and 0.1 respectively. Here there is a tradeoff between the overall accuracy and the sensitivity. Figure 3.7 shows the slight improvement of overall accuracy.

As the one with the highest Dice coefficient, we set our momentum to 0.5.

3.4.8. Varying the Activation Function

As the last step towards choosing our final architecture, we experiment with the 3 main types of possible activation function: ReLU, Tanh, Sigmoid. The results are shown in the table below

Activation Function	Sensitivity	Specificity	Test Dice Coefficient
ReLU	0.87	0.986	0.984
Tanh	0.934	0.98	0.979
Sigmoid	0.941	0.968	0.967

As could be expected, the architecture with ReLU activation functions performed better overall with the highest Dice coefficient of the 3 at 0.984, followed by the ones with Tanh and Sigmoid activation functions at 0.979 and 0.967. However its sensitivity is the lowest of the three at 0.87, with the ones for the Tanh and Sigmoid at 0.934 and 0.941 respectively. The mask results shown in Figure 3.8 are very similar across all three models.

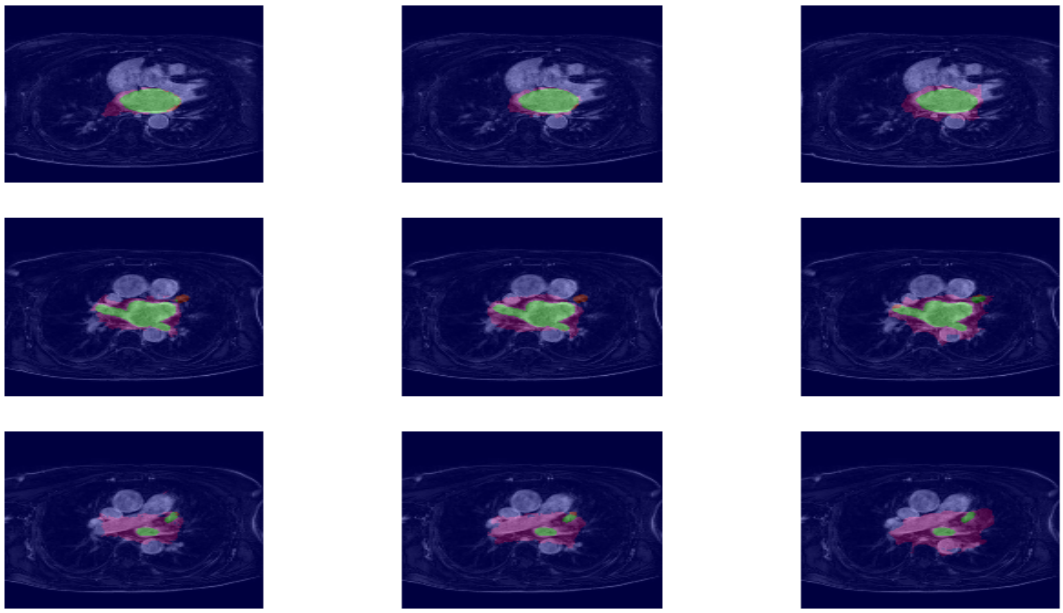

Figure 3.8.: Masks for varying activation function. Each row displays the same mask generated by CNNs with from left to right, ReLU, Tanh and Sigmoid activation functions.

Unsurprisingly, we opt for an architecture with ReLU activation functions.

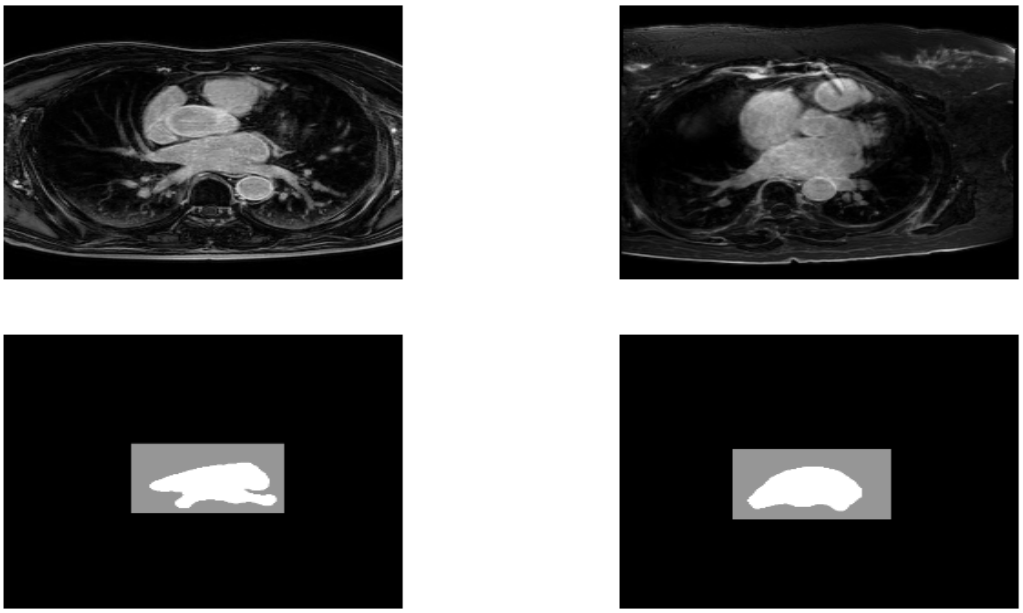

Figure 3.9.: Images illustrating the atrium box. The top row contains transversal slices of 2 different CT scans and the bottom row, their labelling with, in black, non-atrium voxels outside the atrium box, in grey, non-atrium voxels inside the atrium box, and in white, atrium voxels.

3.5. Sampling Method

We now look at a possible improvement of the sampling method used for obtaining the training dataset. As most of the classification errors lie in the border regions of the atrium, one obvious avenue of improvement is to increase the proportion of examples in that region. To that end, we consider a rectangular box containing the atrium, which we call the atrium box.

Such a box is constructed by going through all the coordinates of the voxels in the atrium, pick the minimum and maximum coordinate values in each of the coordinate planes, and possibly add some padding. Figure 3.9 shows 2 transversal slices of different CT scans with their labels, where the non-atrium voxels inside the atrium box are in grey. A simple modification of our sampling procedure would be to sample half of the non-atrium voxels inside the box and half outside it.

We train our selected CNN using datasets generated from 3 sampling procedures: using no atrium box, using a small atrium box constructed by the procedure above with a padding of 5 pixels in the transversal plane and of 1 pixel in the remaining coordinate direction, and finally a large atrium box with a padding of 30 pixels in the transversal plane and of 5 pixels in the remaining coordinate direction. The test results are shown

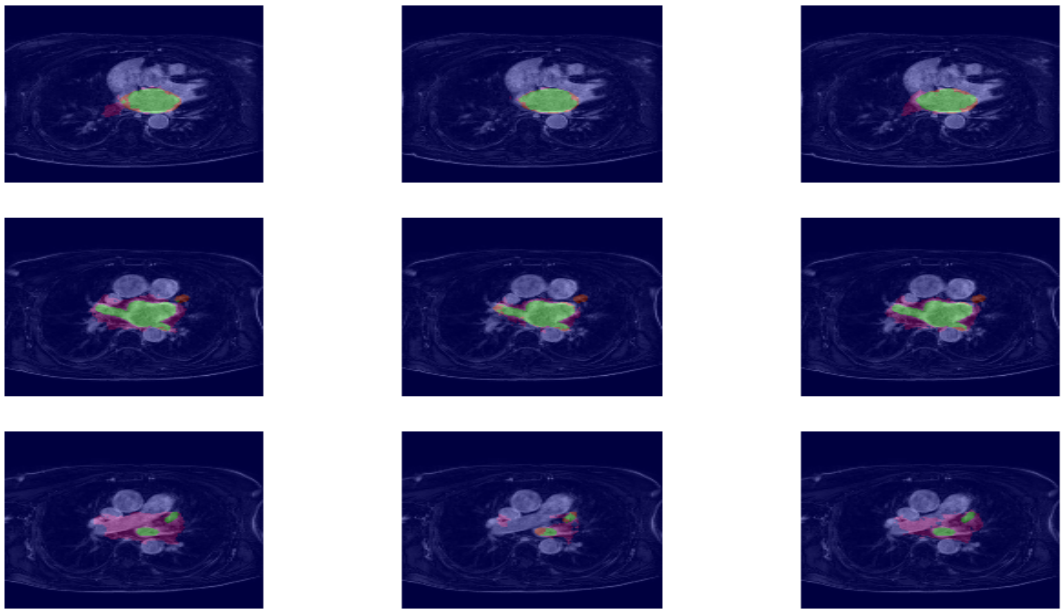

Figure 3.10.: Masks for varying training dataset. Each row displays the same mask generated by CNNs with from left to right, using training datasets generated with no atrium box, a small atrium box, and a large box

in the table below.

Sampling Type	Sensitivity	Specificity	Test Dice Coefficient
No Atrium Box	0.881	0.986	0.984
Small Atrium Box	0.838	0.994	0.991
Large Atrium Box	0.871	0.989	0.987

Sampling using the small atrium box yield a substantial improvement of the Dice coefficient, raising it to 0.991 against 0.984 with a large atrium box and 0.984 with no atrium box. However the sensitivity is negatively affected, dropping from 0.881 by sampling with no atrium box, to 0.871 with a large one and more still to 0.838 with a small one. Overall, training with a dataset generated using an atrium box improves the accuracy substantially, as illustrated by Figure 3.10 showing smaller pink regions for its masks over the other two sets.

3.5.1. Summary of the Selected Model.

To review, we choose an architecture consisting of 2 convolutional layers with convolution filters of size 5×5, max pooling filters of size 2×2, and 32 and 64 feature maps in each successive layer. We add one fully connected layer with 100 hidden units. Additionally we used ReLU activation functions, and learning parameters set to 0.5 for the learning

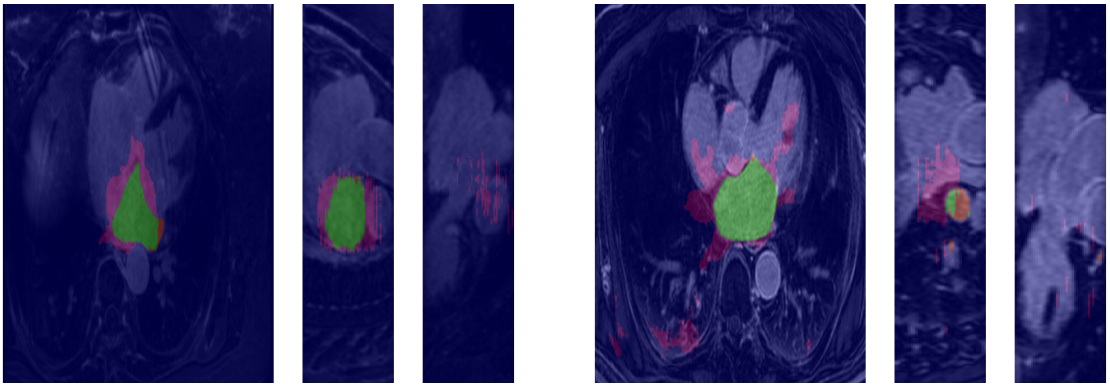

Figure 3.11.: Left: grey scale slices from a CT scan taken in the transversal, saggital and coronal planes. Right: illustration of the triplanar

rate and momentum, and 6000 for the batch size.

To further test our model, we segment the remaining 6 test CT scans and present in the table below some summary statistics describing the performance of our selected CNN.

	Mean	Standard Deviation	Minimum	Maximum
Sensitivity	0.847	0.04	0.777	0.900
Specificity	0.989	0.003	0.986	0.994
Dice Coefficient	0.987	0.003	0.983	0.991

The chosen CNN seems to be performing reasonably well on all the test CT scans, with a mean Dice coefficient of 0.987 and tight standard deviation of 0.003. However, the sensitivity is more variable across the CT scans, with a standard deviation of 0.04 and a low minimum value at 0.777. This seems to indicate a lack of consistency in classifying atrium voxels.

Despite a good overall testing accuracy, the CNN behaves erratical on some of the masks observed. Figure A shows 2 examples of such masks in the transversal, saggital and corronal planes taken at index 200, 200 and 20 on 2 different test CT scans. The masks on the left hand side show expected errors near the border regions of the atrium. The masks on the right hand side show more unexpected errors away from the atrium. More examples of masks taken from each of the test CT scans are shown in the Appendix.

4. Conclusions and Further Work

We conclude this thesis with a brief summary. We first discussed the recent advances which have allowed Deep Learning techniques to engender a string successes in pattern recognition, and particularly in the context of images. These successes have led to an increasing interest in applying these techniques to Medical Imaging. In the second chapter, we presented some background material on CNNs, covering convolutional layers, subsampling layers and an overview of the architectural organisation. In Chapter 3, we turn our attention to implementing a CNN ourselves using a tri-planar approach to generate the input set as a way of providing 3D information at a much lower cost than 3D patches along with a number of other practical considerations. We then performed model selection, finding a much better architecture and set of learning parameters than was initially proposed. To that end, we varied the number of convolutional and connected layers, the number of feature maps and hidden units, the type of activation function, the learning rate and the momentum. We then tried a different sampling procedure to increase the frequency of examples near the boundary of the atrium where most of the errors lie, yielding a significant increase in accuracy. Our final model gives a mean Dice coefficient of 0.985 across our 7 test CT scans. Most of the errors were at the border regions with the atrium where we expect them to be. However, despite the high rate of accuracy, the sensitivity was shown to be variable and some masks demonstrated errors far away from the atrium which we would expect the classifier not to commit. As a result, we do not expect this particular classifier to be useful for practicing radiologists in their work.

A number of things could be undertaken to improve the results. A first obvious step would be to increase the number of training examples. Another one would be to further improve with the sampling procedure in order to increase the number of training examples at the border regions. One could, for instance, systematically include all the pixels located at the border region of the atrium from each CT scan allocated to generating the training set. Another avenue for improvement would be to add more inputs channels to provide more information to the network, such as 3D patches.

Bibliography

[1] vimage programming guide, 10 2011.

[2] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[3] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[5] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012.

[6] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot, and Nassir Navab, editors, *Medical Image Computing and Computer-Assisted Intervention 2013*, volume 8150 of *Lecture Notes in Computer Science*, pages 411–418. Springer Berlin Heidelberg, 2013.

[7] Alexandre de Brebisson and Giovanni Montana. Deep neural networks for anatomical brain segmentation. *CoRR*, abs/1502.02445, 2015.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[10] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[11] Koray Kavukcuoglu, Pierre Sermanet, Y lan Boureau, Karol Gregor, Michael Mathieu, and Yann L. Cun. Learning convolutional feature hierarchies for visual recognition. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1090–1098. Curran Associates, Inc., 2010.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[13] Hugo Larochelle, Yoshua Bengio, Jérome Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

[14] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] John Markoff. Scientists see promise in deep-learning programs. *New York Times*, November 2012.

[16] Srinivasan Nagaraj, G Narasinga rao, and K Koteswararao. The role of pattern recognition in computer-aided diagnosis and computer-aided detection in medical imaging: A clinical validation. *International Journal of Computer Applications*, 8(5):18–22, October 2010. Published By Foundation of Computer Science.

[17] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] Adhish Prasoon, Peter Kersten Petersen, Christian Igel, Francois Bernard Lauze, Erik Dam, and Mads Nielsen. *Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network*, pages 246–253. Lecture Notes in Computer Science. Springer, 2013.

[19] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

A. Extra Masks

We include some extra masks from our final model taken from all 7 test CT scans.

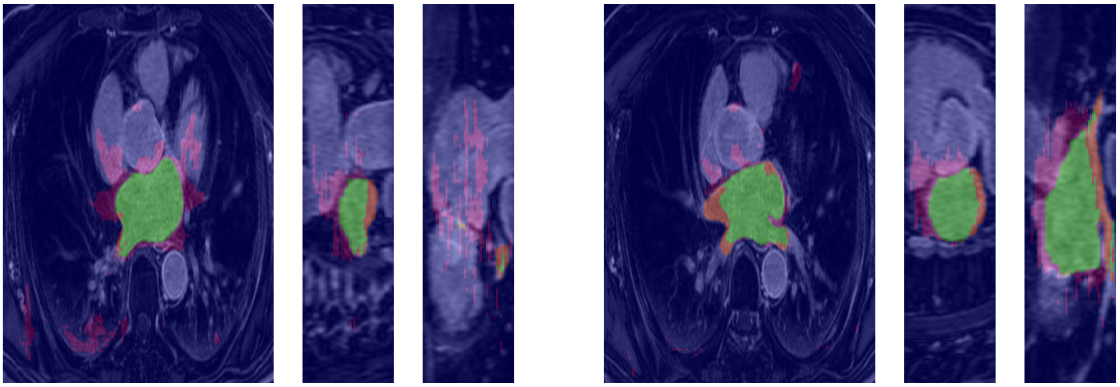

Figure A.1.: Two sets of masks taken from CT scan 14012303

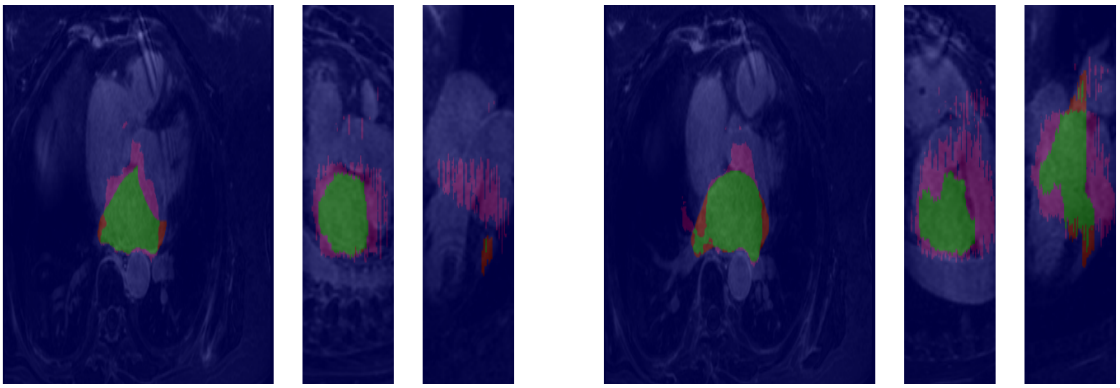

Figure A.2.: Two sets of masks taken from CT scan 14022801

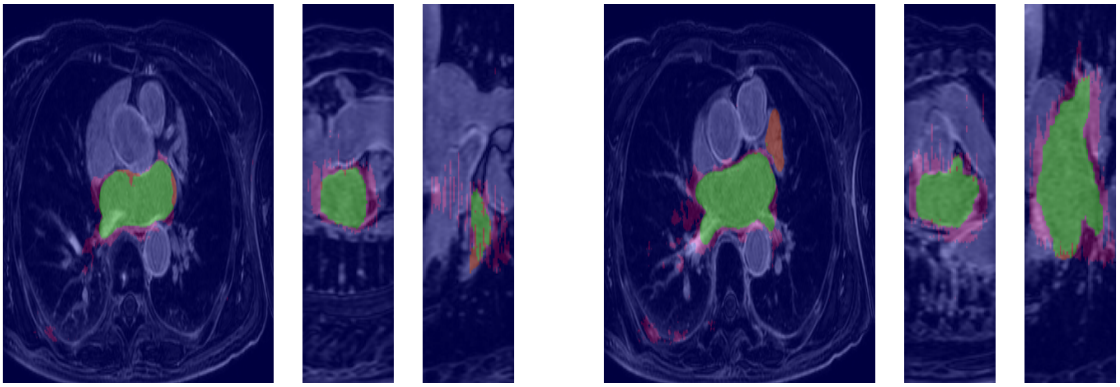

Figure A.3.: Two sets of masks taken from CT scan 14031001

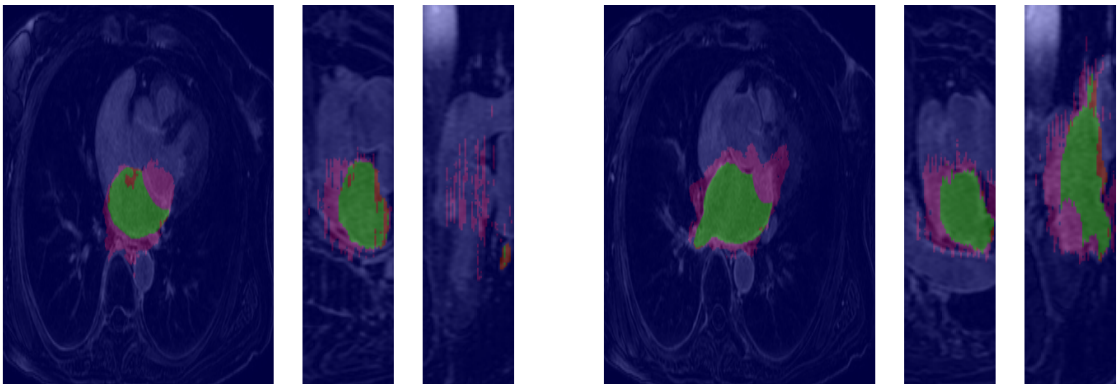

Figure A.4.: Two sets of masks taken from CT scan 14031201

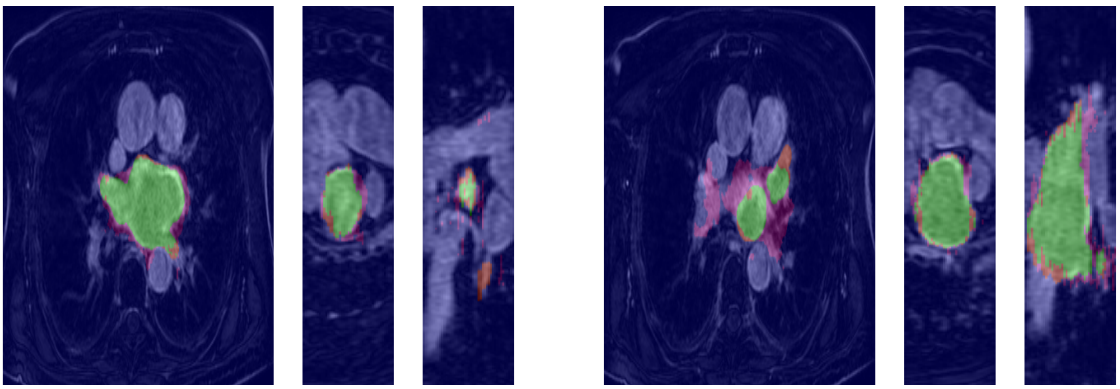

Figure A.5.: Two sets of masks taken from CT scan 14040204

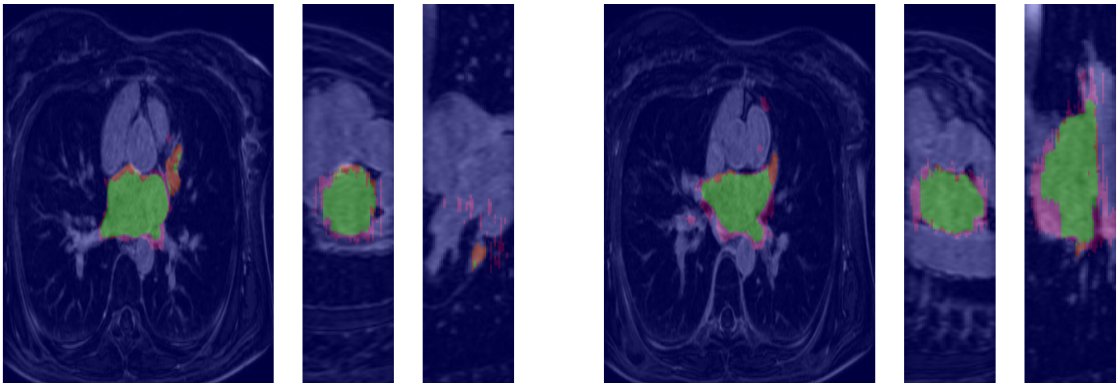

Figure A.6.: Two sets of masks taken from CT scan 14051403

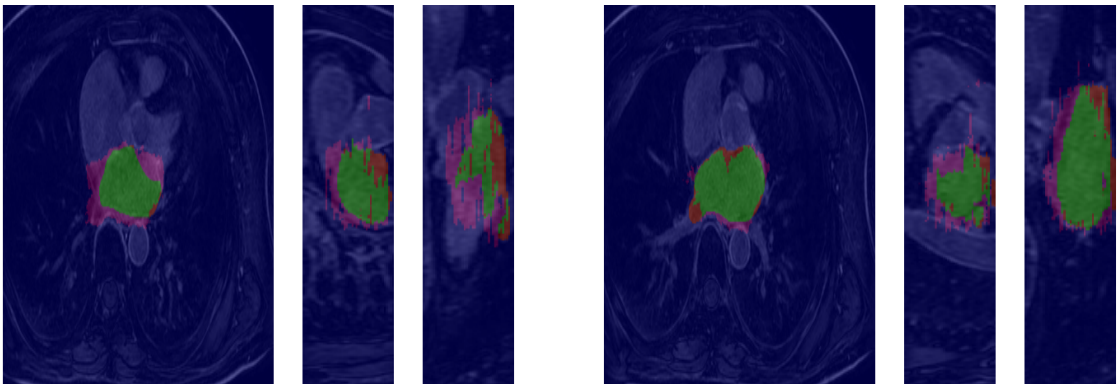

Figure A.7.: Two sets of masks taken from CT scan 14051404

B. Code

Only the two most important part of the code for training neural networks are shown: the model and the training function. In the interest of brevity, we omit the rest which deals with handling logistics, data generation and plotting. However, the entire code base can be found on Github at `https://github.com/eisenhower444/Master_project`.

The following code defines a model using the nn package in Torch.

```
require "nn"
require "cudnn"

----------------------------------------------------------------------
-- Model parameters

-- hidden units, filter sizes:
nfeaturemaps = { 32, 64, 100 }
filtsize     = 5
poolsize     = { 2, 2 }
featuremaps_h = 5
featuremaps_w = 5
noutputs     = 2

----------------------------------------------------------------------
model = nn.Sequential()

-- stage 1 : mean suppresion -> filter bank -> squashing -> max pooling
model:add(cudnn.SpatialConvolution(nfeats, nfeaturemaps[1], filtsize,
    filtsize))
model:add(cudnn.ReLU(true))
model:add(cudnn.SpatialMaxPooling(poolsize[1],poolsize[1],poolsize[1],poolsize[1]))
-- stage 2 : mean suppresion -> filter bank -> squashing -> max pooling
model:add(cudnn.SpatialConvolution(nfeaturemaps[1], nfeaturemaps[2], filtsize,
    filtsize))
model:add(cudnn.ReLU(true))
model:add(cudnn.SpatialMaxPooling(poolsize[2],poolsize[2],poolsize[2],poolsize[2]))
-- stage 2 : standard 1-layer MLP:
model:add(nn.View(nfeaturemaps[2]*featuremaps_h*featuremaps_w))
model:add(nn.Dropout(0.5))
model:add(nn.Linear(nfeaturemaps[2]*featuremaps_h*featuremaps_w,
    nfeaturemaps[3]))
model:add(nn.ReLU())
```

```
model:add(nn.Dropout(0.5))
model:add(nn.Linear(nfeaturemaps[3], noutputs))
model:add(nn.LogSoftMax())

----------------------------------------------------------------------
criterion = nn.ClassNLLCriterion()
```

The following code is the main workhorse for training a neural network on one or many GPUs in Torch.

```
-- Multi-GPU set up
if opt.number_of_GPUs > 1 then
    print('Using data parallel')
    local GPU_network = nn.DataParallel(1):cuda()
    for i = 1, opt.number_of_GPUs do
        local current_GPU = math.fmod(opt.GPU_id + (i-1)-1,
            cutorch.getDeviceCount())+1
        cutorch.setDevice(current_GPU)
        GPU_network:add(model:clone():cuda(), current_GPU)
    end
    cutorch.setDevice(opt.GPU_id)

    model = GPU_network
end

model:cuda()
criterion:cuda()

-- Optimizer
optimator = nn.Optim(model, optimState)

-- Retrieve parameters and gradients:
-- this extracts and flattens all the trainable parameters of the model
if opt.number_of_GPUs > 1 then
    parameters, gradParameters = model:get(1):getParameters()
    cutorch.synchronize()
    model:cuda() -- get it back on the right GPUs
else
    parameters, gradParameters = model:getParameters()
end

function train()

    -- epoch tracker
    epoch = epoch or 1

    -- set model to training mode (for modules that differ in training and
        testing, like Dropout)
    model:training()

    -- shuffle at each epoch
    shuffle = torch.randperm(trainData.size())

    -- do one epoch
    for t = 1,trainingSize,opt.batchSize do
```

```
      -- disp progress
      xlua.progress(math.min(t+opt.batchSize-1,trainingSize), trainingSize)

      -- create mini batch
   if t < (trainingSize - opt.batchSize) then
      batchSize = opt.batchSize
   else
      batchSize = trainingSize - t - math.fmod((trainingSize -
          t),opt.number_of_GPUs)
   end

      inputs = torch.Tensor(batchSize,nfeats,patchsize,patchsize)
      targets = torch.Tensor(batchSize)
      for i = t,math.min(t+opt.batchSize-1,trainingSize) do
          -- load new sample
          inputs[{{i%batchSize + 1},{},{},{}}] =
              trainData.data[shuffle[i]]:clone()
          targets[i%batchSize + 1]          = trainData.labels[shuffle[i]]
      end

      inputs   = inputs:cuda()
      targets  = targets:cuda()

      f, outputs = optimator:optimize(optim.sgd, inputs, targets, criterion)

      if opt.number_of_GPUs > 1 then cutorch.synchronize() end
    end

    -- next epoch
    collectgarbage()
    epoch = epoch + 1
end
```