Imperial College
London

Imperial College London
Department of Mathematics

Segmentation of CT scans into Atrium/non Atrium

Thomas Vogel

CID:

Supervised by Prof Giovanni Montana

28th August 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

This is a LaTeX template to be used for project theses by the students of the Imperial College MSc in Statistics. Please have a look at the `*.tex` source and the code comments therein. You may use this template with only minor changes, or make any major style changes you desire (e.g. redesigning the title page, adding headers,...), or you may use a different template, or you may write your own approach from scratch, or you may just use some bits and pieces from the template's LaTeX code. It's up to you. Note that resources on how to use LaTeX are available on Blackboard under 'R&LaTeX intro'.

This template will quote relevant sections from the MSc in Statistics student handbook throughout; e.g.

> " The abstract should be a brief statement of the aims and outcomes of the project, to summarise/advise even for a casual reader!"

Acknowledgements

Thank you supervisor/friends/family/pet.

“ Include an acknowledgement.”

Table of Contents

1 Introduction

Describe some high level background about Machine Learning, deep learning, CNNs, Medical Imaging, literature review style.

1.1 Deep Learning and Convolution Neural Networks

The high dimensionality of the data increases the learning complexity of the algorithms grow exponentially with linear increase in the dimensionality of the data. So the job of the data engineer is to find a representation of the data, also called features, which captures the information contained within it while reducing the learning complexity of the algorithms used. There are two approaches to dealing with this. One is called feature engineering where the features are humanly generated. This has arguably become the bulk of the data scientist work [reference]. A second is to learn the features as part of the learning process. This is the Deep Learning approach. The craze surrounding it has to do with the hope that a sparse representation of the data can be learned with little to no human interaction.

The field of Deep learning arose from an inspiration about how the brain works in terms of the flow of information through layers of more and more abstract feature detections. The point is to train a hierarchical network of simple processing units called neurons or nodes on a large set of data to learn a good representation model of the data and then add a simple standard classification layer on top to do the classification. Learning in this context means finding a set of weights which makes the Neural Network exhibit desired behaviour (e.g. classify hand written-digits correctly). The complexity of the learning problem influences the depth of neural network required.

The first neural networks with multiple, if few, layers date back to the 60s [reference] and have been around for decades. An efficient gradient based method for supervised learning problems that is widely used called the backpropagation method was developed then and applied to NN in 1981 [reference]. However there were many difficulties with training these NNs which were prohibitive and it was only starting in 2006 and a seminal paper [reference] by Hinton which made NN with many layers trainable in practice using a unsupervised pre-training approach. This has lead to a resurgence in the interest in NNs and Deep Belief Networks.

Another major development was the introduction of Convolutional Neural Networks by LeCun [reference]. CNNs are a specialised type of Neural Network for certain grid-like structures that reduces the number of parameters via weight sharing. In particular,

they have been remarkably effective on classification tasks on images, videos, and audio recordings providing state of the art performance on object recognition and speech tasks. They were introduced in 1998 in the context of MNIST data, a national competition whose object was the classification of hand-written digits. Ever since they have won many official international pattern recognition competitions and are the state of the art in many applications. A full historical account can be found in [reference].

1.2 Deep Learning in Medical Imaging

1.2.1 What is medical imaging, how has it been done so far

Medical imaging is the technique and process of creating visual representations of the interior of a body for clinical analysis and medical intervention. Medical imaging seeks to reveal internal structures hidden by the skin and bones, as well as to diagnose and treat disease. They have traditionally been done by trained clinicians such as radiologist or histologists and usually involves a time consuming and expensive process that cannot be performed at a large scale. Efforts have been made to automate this process called Computer-Aided Detection systems which aim to improve the performance of radiologists as well as their efficacy in speed and cost.

1.2.2 Deep learning in medical imaging

Following the success of Deep Learning approaches to the various contests, there has been a recent effort to implement these models in the context of Medical Imaging, for instance, in the automation of image segmentation of various medical images such as CT, MRI, X-ray or Electron Microscopy images. These include approaches for the segmentation of the lungs [reference], biological neuron membrane [reference] to map 3D brain structure and connectivity, tibial cartilage [reference], detection of bone tissue in X-ray images [reference] and counting cell mitosis in histology images [reference] for cancer screening and assessment, amongst others.

The segmentation process represents a first step necessary for any automatic method of extracting information from an image.

The machine learning approach is to train a model to classify each voxel into its corresponding anatomical region given a training set consisting of labelled medical images.

Automating this task would enable systematic segmentation of medical images on the fly as soon as these images are acquired. They could also be useful in the detection of anatomical abnormalities such in tumour detections.

1.3 Goal of the Thesis

This thesis takes direct inspiration from the paper on brain segmentation, where the entire brain was segmented into 140 different anatomical regions. It will propose a deep convolution neural network for the automated segmentation of chest CT scans into atrium and non-atrium parts using a tri-planar multi-scale approach.

The second chapters discusses some background on neural networks and CNNs. The second chapter then describes the approach undertaken and presents a number of experimental results from model selection, the varying of dataset size and various sampling strategies. We will finish with some conclusions and a discussion of various other ways to improve upon our results.

2 Background Material: on convolutional neural networks

We first start with a brief review of feed forward Neural Networks before elaborating on Convolutional Neural Networks (CNNs), a specialised kind of neural network which will be relied on for our architecture for our classification problem. CNNs are a special type of Feed-Forward Neural Networks that are particularly effective for input types that have a grid-like structure, such as images, videos or time-series, where the correlation between inputs is highly localised. CNNs have recently been responsible for some of the state-of-the-art results in image recognition [references], speech recognition [references], and natural language processing [reference]. We will discuss CNNs in the context of images.

2.1 Feed Forward Neural Networks

A feedforward Neural networks is a standard model in the machine literature for classification problems. It consists in a number of layers of simple processing units, also called neurons or nodes. Every unit in a layer is connected with all the units in the previous layer, with each connection characterised by a weight encoding the knowledge of the network. The value at each unit is generated by taking a linear combination of the values of the incoming connections weighted by their weights and the result being passed through an activation function. Data enters at the inputs and passes through the network, layer by layer, until it arrives at the output layer. The choice of the activation function is determined by the nature of the data and the assumed distribution of 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 whatever. The process can then be interpreted as a forward propagation of information through the network.

Training the neural networks is done by minimising an error measure of its performance over a training set using gradient-based optimisation routines. Evaluating the error gradients with respect to the parameters can be efficiently done via the Backpropagation algorithm [reference]. Regularisation such as dropout [reference] where at every epoch, only a preset percentage of randomly chosen set of hidden units are active, is generally added to the optimisation procedure to prevent overfitting.

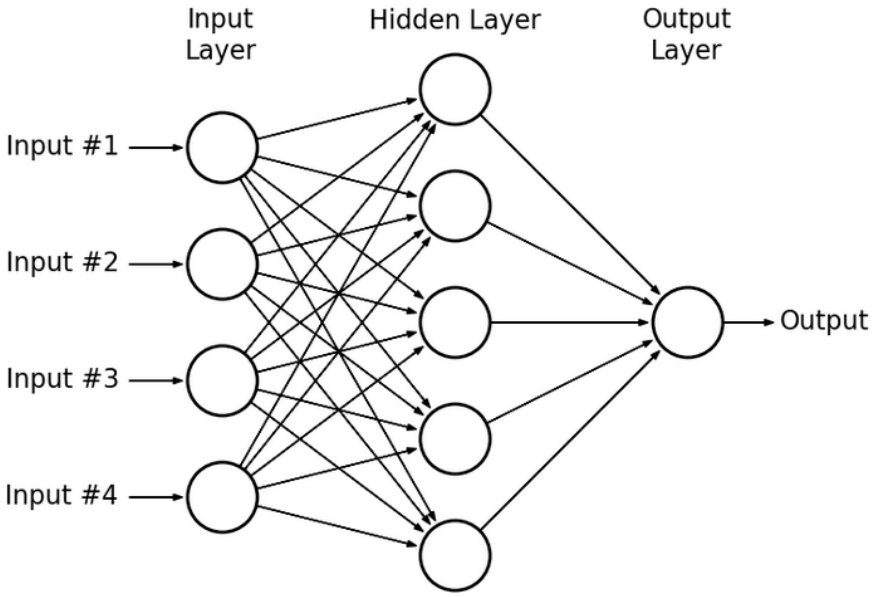

Figure 2.1: A 2 layer feedforward neural network with 4 input nodes, 5 hidden nodes and 1 output node.

2.2 Convolutional Layers

A CNN is a specialised kind of feedforward neural network where the first few layers of the architecture are convolutional layers. These typically consist of three stages: a convolution stage, a detection stage and a subsampling stage.

2.2.1 Convolution

The convolution stage is responsible for implementing k convolution operation over each channel of the input images. This is accomplished by convolving each channel of the input layer with k kernels of size $s \times s$ in parallel. Unlike fully connected layers, in a convolution, each output node is locally connected to a small contiguous subset of corresponding input nodes determined by the kernel size. Additionally the convolution operation implies weight sharing which greatly reduces the number of trainable parameters. In the case of $m \times m$ input images, the set of output nodes resulting from one kernel is called a feature map of size $m - s + 1$, each detecting the presence of a particular feature in the input image encoded by the kernel. Having multiple kernels run through the input layer in parallel produces a set of feature maps, each responsible for detecting a particular type of feature which might be present in the image.

The benefits of this approach are three folds :

- sparse interaction: every output node is connected to a local subset of inputs.

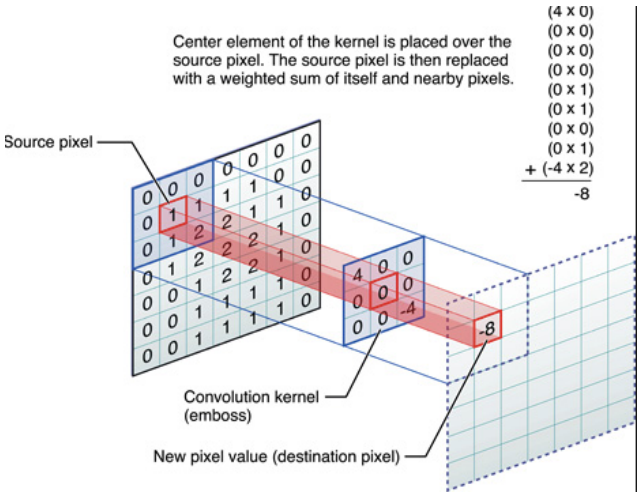

Figure 2.2: The convolution layer. Illustration of the formation of a feature map.

- weight sharing: the same kernel, and thus set of trainable parameters, is used for every output node to produce a given feature map.
- 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 thereby introducing non-linearity into the system. These activation units usually consist of either Rectified Linear units (ReLU), Tanh units, or Sigmoid units.

Each feature map is then subsampled in the pooling stage by aggregating contiguous nodes 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.

The benefits of subsampling are that it 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 makes 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 of the pooled outputs.

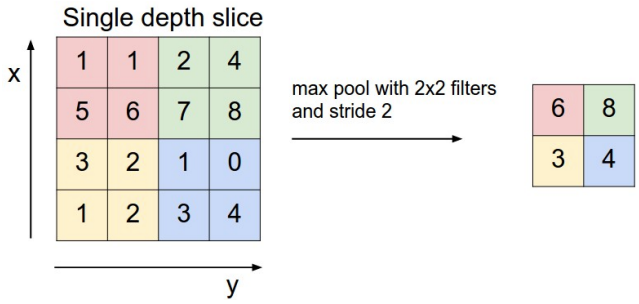

Figure 2.3: Subsampling in action. Here maxpooling is used to halve the size of an feature map using a 2×2 filter

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 convolutional layer 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 followed by an output layer. The convolutional layers thus serve as a way to reduce the dimensionality of the image by extracting meaningful local features.

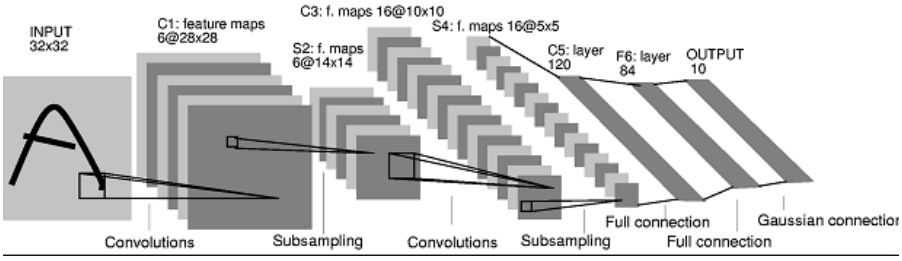

Figure 2.4: A typical convolutional network architecture with 2 convolutional layers, 2 fully connected layer and a Gaussian connection as output layer.

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 it. The atrium is composed of two chambers called the right and left atrium which are the two entry points of the blood into the heart, one returning to the heart to complete the cardiovascular cycle through the body and the other coming back from the lungs after being oxygenated and purified of toxins.

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, each entry being either a 1 or 0 indicating whether the corresponding voxel in the CT scan is in the Atrium or not.

3.2 Generating the datasets: The Tri-Planar Method

Classifying the voxels require 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 whatever. This standard technique has been found to give competitive results compared to using 3D patches while being much more computationally and memory efficient [reference the paper that started it on knee cartilage]. In addition, we use a multiscale approach where we add 3 more input channels composed of compressed patches that are originally 5 times larger than the above set of patches, resized to the same size to provide global information about the surroundings of the concerned voxel.

Each patch is fed into a different input channel of the Convolutional Neural Network with one or more convolutional layers. The outputs of those channels are then feed as

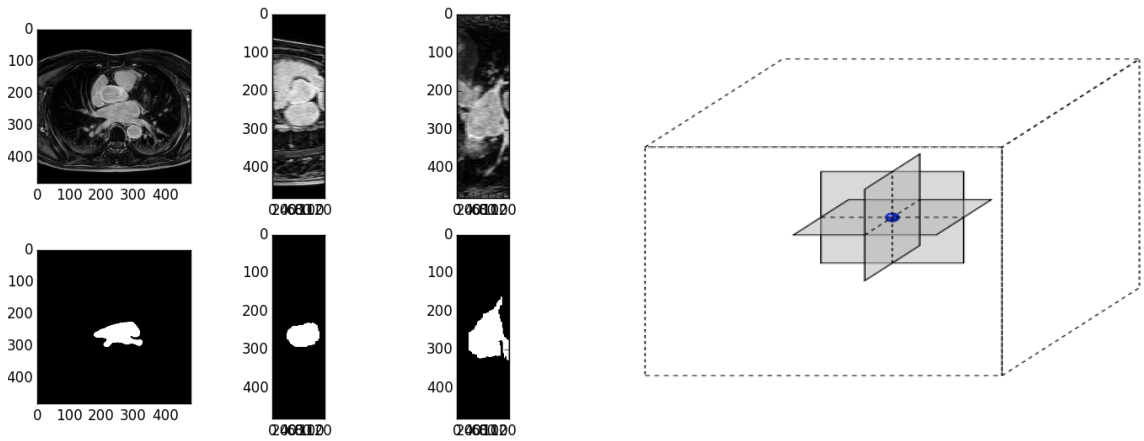

Figure 3.1: Left: grey scale slices from a CT scan taken in the transversal, saggital and coronal planes. Right: illustration of the triplanar

inputs to a set of connected layers itself connected to a classifying layer as output layer.

3.3 Implementation Details

3.3.1 Libraries

To train Neural Networks, we use Torch, an open-source library maintained by Facebook in Lua aiming to provide a Matlab-like environment for scientific computing, along with a number of dependent libraries (nn, cunn, cudnn, fbcunn) to train the convolutional 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's. 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

For the purpose of model selection, we allocated 20 of the 27 CT scans to the training set and the remaining 7 to a testing 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 testing examples equally divided among the testing CT scans. Furthermore at the end of training, each model fully segments a test CT scan from the testing set to provide performance statistics. In particular, from this segmentation image, we evaluate the model's sensitivity, which is the percentage of correctly classified atrium voxels, specificity, which is the percentage of correctly classified non-atrium voxels, and overall classification rate also known as the Dice coefficient, which is calculated by evaluating the proportion of correctly classified voxels in the segmented image. Model selection will be conducted on the basis of these three test statistics.

Additionally, 3 transversal mask images are generated from the segmented CT scan, where the grey CT scan image is overlaid with colours representing the error status of a given voxel. These masks provide a visual element to 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 with patches of size 32*32, 2 convolutional layers with 32 and 64 feature maps respectively, 2 fully connected layers with 1000 and 500 hidden units each respectively, and a logsoftmax output layer giving log probabilities. 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. 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 type of activation function (ReLU, Tanh or Sigmoid).
- the type of subsampling function (maximum or average pooling).
- the learning rate.
- the momentum.

3.4.2 varying 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)

These convolutional layers are then followed by two fully connected layers with 1000 and 500 hidden layers each. The following table gives the results for each of these architectures trained over 100 epochs.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
1 Conv L	0.959	0.956	0.913	0.970	0.969
2 Conv L	0.958	0.961	0.734	0.989	0.984
3 Conv L	0.956	0.960	0.762	0.987	0.983
4 Conv L	0.959	0.956	0.881	0.973	0.972

The three key statistics are the sensitivity, the specificity and the Dice coefficient over the entire test CT scan. As only 1.8% of its voxels are in the atrium, the specificity has an overbearing effect on the test Dice coefficient and thus takes priority when selecting models. The table above shows a tradeoff between the two across the 4 architectures. While the 1 convolutional layer architecture has the highest sensitivity at 0.913, it also yields the lowest specificity at 0.970, test statistics very close to the 4 convolutional layer architecture. The architectures with 2 and 3 convolutional layers have similar test statistics with relatively low specificities at 0.734 and 0.762 respectively and high specificities at 0.989 and 0.987 respectively.

Figure whatever brings a visual element to this comparison. Each column shows the segmentation of 3 different transversal slices of the test CT scan image. The first and fourth columns corresponding to segmentations by the architectures with 1 and 4 convolutional layers respectively 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

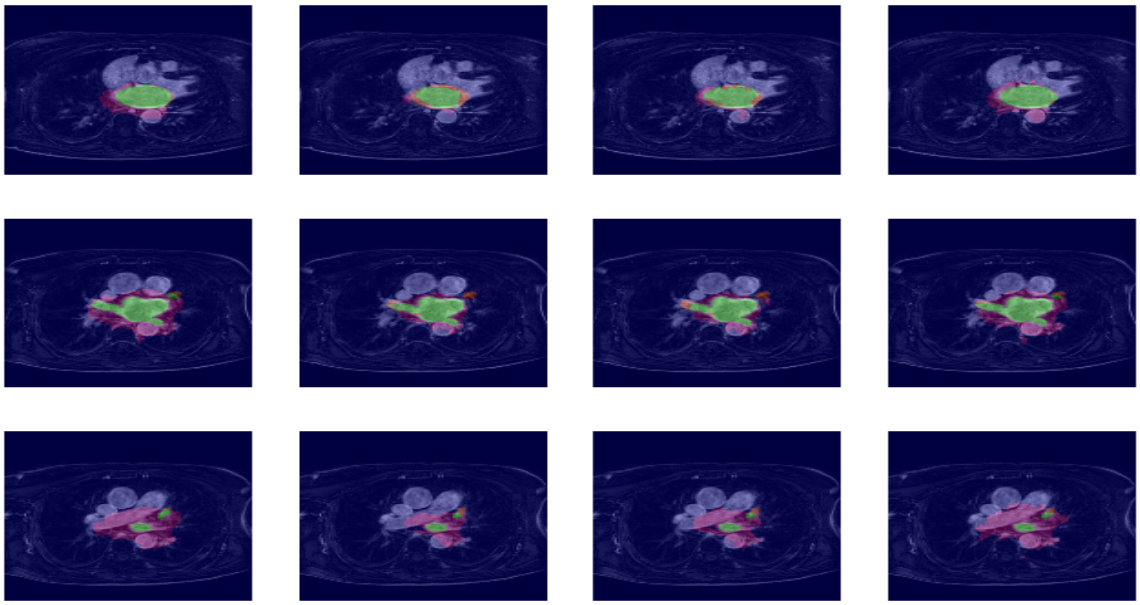

Figure 3.2: Masks for varying number of convolutional layers

layers.

We choose to go with having a higher specificity at this stage and thus select an architecture with 2 convolutional layers.

3.4.3 varying 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 layer respectively. The results are shown in the following table.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
1 Connected L	0.957	0.959	0.887	0.972	0.971
2 Connected L	0.958	0.961	0.856	0.974	0.972
3 Connected L	0.953	0.951	0.915	0.967	0.966

All three architectures have similar performances on the test CT scan with specificities around 0.97 and sensitivities around 0.9. Figure whatever 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 make much difference to the performance of the classifier and hence we opt for having 1 fully connected layer in our final architecture.

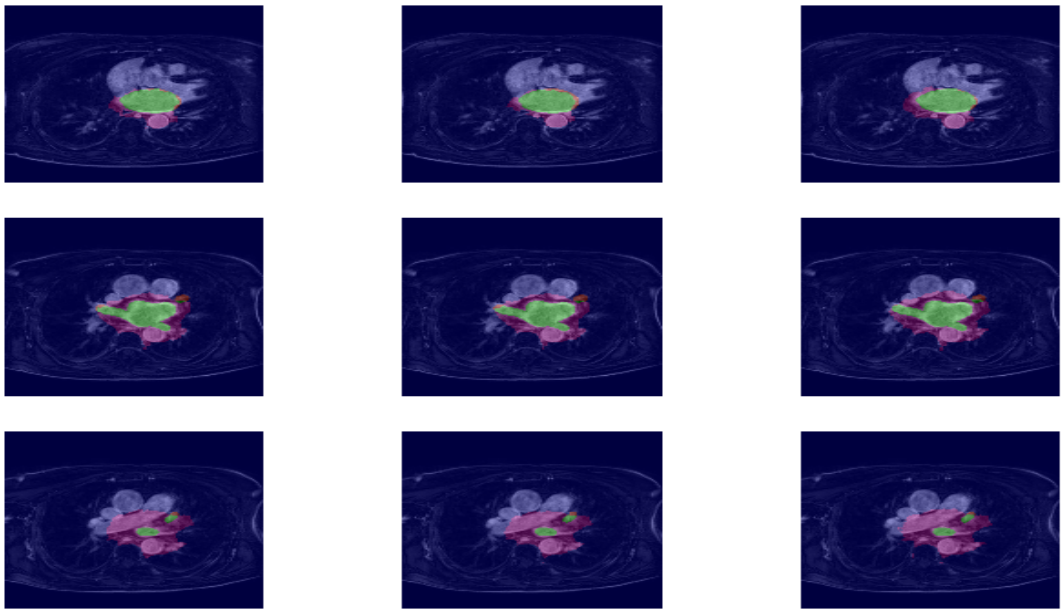

Figure 3.3: Masks for varying number of connected layers

3.4.4 varying 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 tried configurations with the first layer having 16, 32, and 64 feature maps. The summary of the results are shown below.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
With 16 FM	0.956	0.953	0.919	0.967	0.966
With 32 FM	0.957	0.958	0.894	0.972	0.970
With 64 FM	0.959	0.958	0.904	0.971	0.970

Yet again there is not much difference between the performance of all four architectures. The specificities and sensitivities are around 0.97 and 0.9 respectively. We thus settle with having a first layer with 32 feature maps.

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.

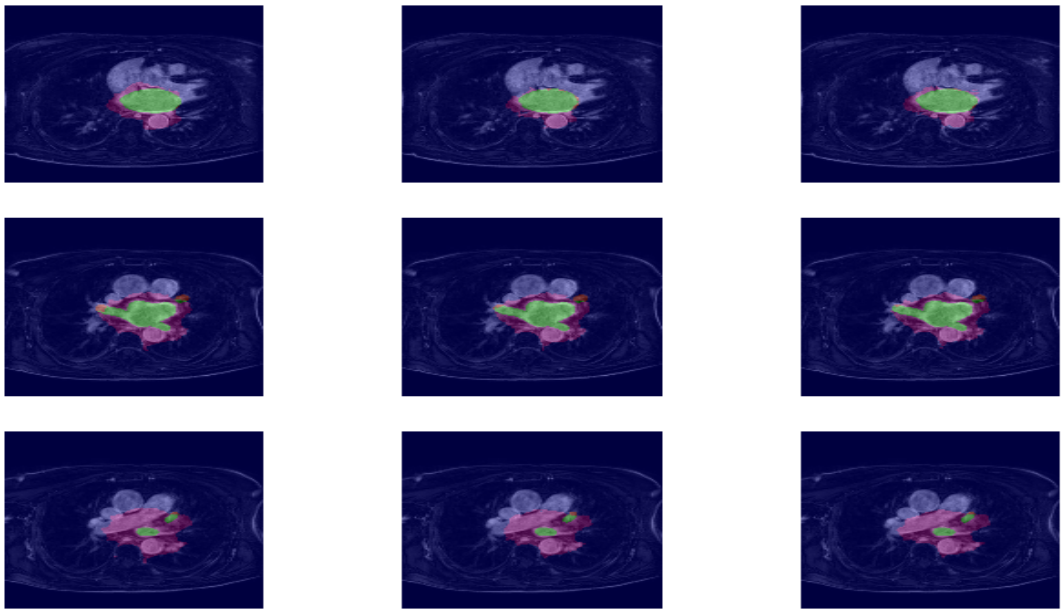

Figure 3.4: Masks for varying number of feature maps

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
100 Hidden Units	0.956	0.957	0.902	0.969	0.967
200 Hidden Units	0.955	0.957	0.913	0.968	0.967
500 Hidden Units	0.956	0.953	0.921	0.967	0.966
1000 Hidden Units	0.955	0.953	0.921	0.967	0.966

Again the statistics across all 4 cases are very similar except for the increasing sensitivity for the 500 and 1000 hidden unit architectures. We thus select to have 500 hidden units in the fully connected layer.

3.4.6 varying the activation function

We experimented with different types of activation function: ReLU, Tanh, Sigmoid. Doing it with ReLU is better...

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
ReLU	0.956	0.953	0.923	0.967	0.966
Tanh	0.956	0.942	0.942	0.955	0.955
Sigmoid	0.673	0.987	0.0	1.0	0.982

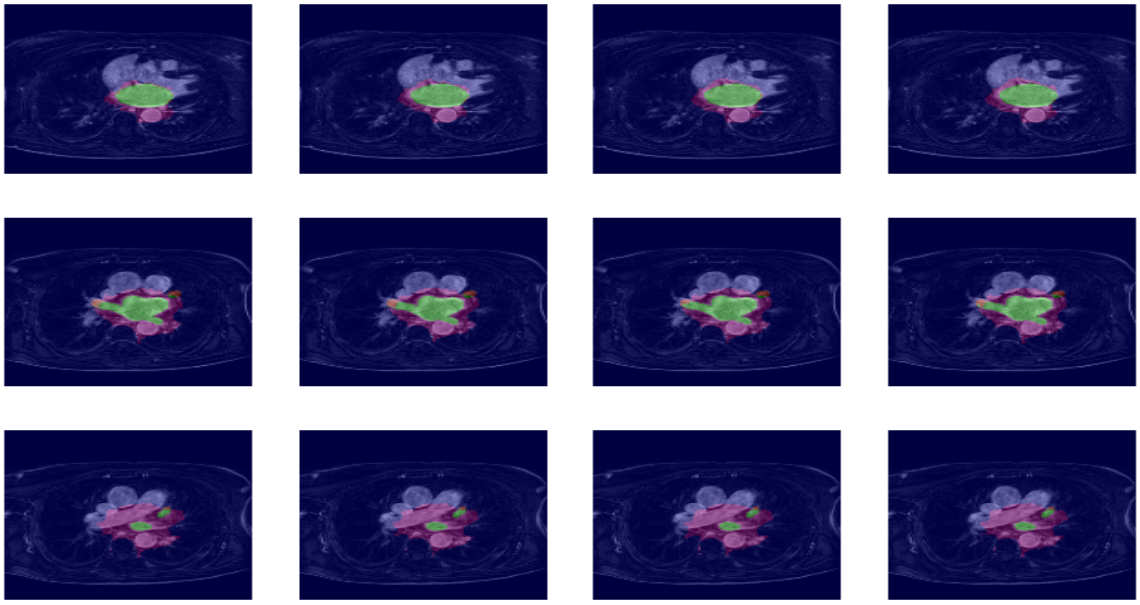

Figure 3.5: Masks for varying number of hidden units

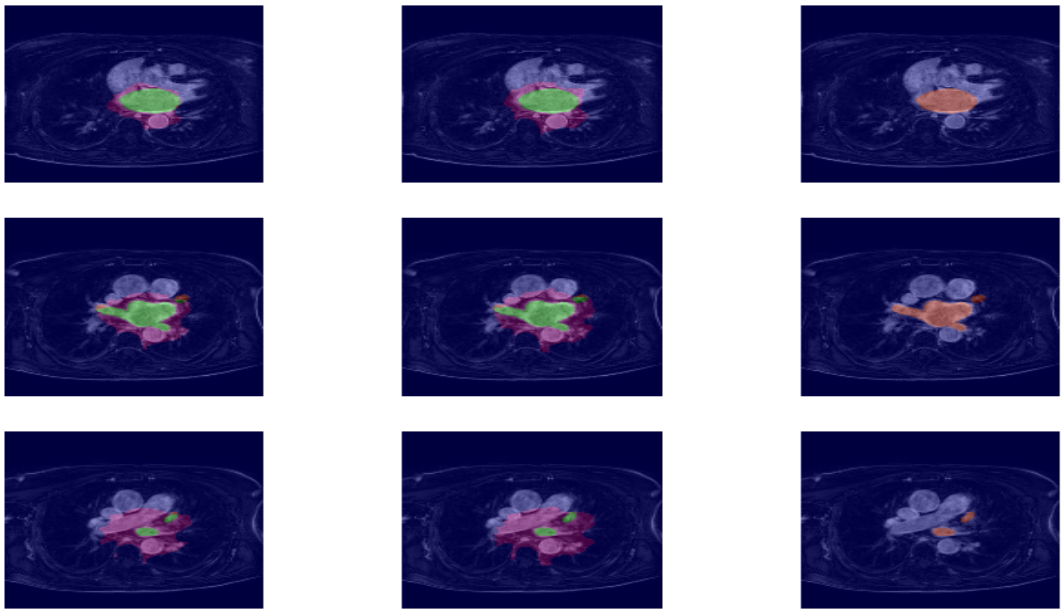

Figure 3.6: Masks for varying activation function

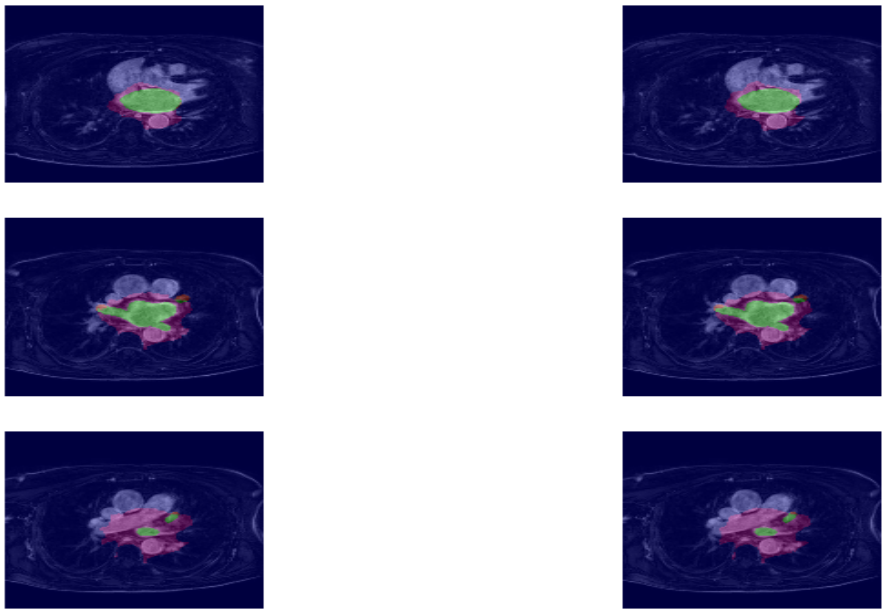

Figure 3.7: Masks for varying pooling function.

3.4.7 varying the type of pooling

We also tried different types of pooling: Max pooling and average pooling. Average Pooling is better.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
Max Pooling	0.956	0.953	0.923	0.967	0.966
Average Pooling	0.953	0.95	0.937	0.964	0.963

3.4.8 varying the learning rate

Tried different learning rates: 0.01, 0.05, 0.1, 0.5. It seems that 0.1 is best. Having a learning rate of 0.5 doesn't train the network.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
0.01	0.953	0.95	0.936	0.964	0.963
0.05	0.968	0.958	0.935	0.972	0.971
0.1	0.973	0.963	0.946	0.976	0.975
0.5	0.667	0.987	0.0	1.0	0.982

3.4.9 varying the momentum

Tried different momentums: 0, 0.01, 0.05, 0.1, 0.5, 1.

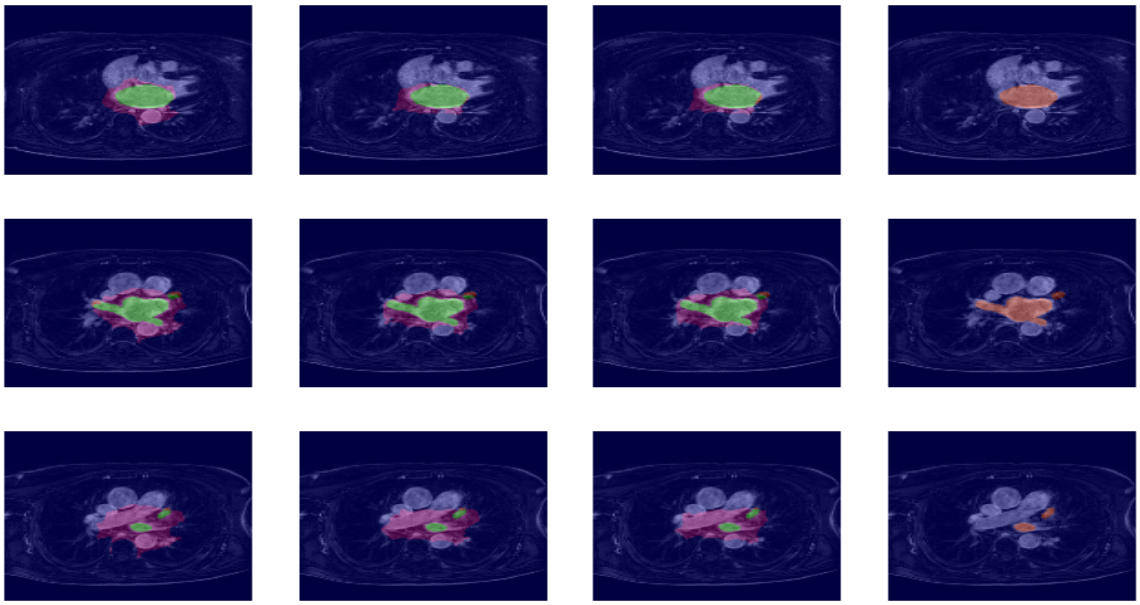

Figure 3.8: Mask for varying learning rate.

	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
0	0.972	0.961	0.951	0.975	0.974
0.05	0.972	0.965	0.933	0.977	0.976
0.1	0.972	0.962	0.945	0.975	0.975
0.5	0.973	0.969	0.906	0.979	0.978

3.4.10 varying datasets type

The first thing we investigate is the sampling method for obtaining the training set. In order to increase the number of non-atrium training examples lying near the boundaries where we expect most of the classification errors to lie, we construct a rectangular area which contains the atrium. The atrium box is constructed by going through all the coordinates of the voxels labeled as being in the atrium, picking the minimum and maximum values in each of the coordinate planes, and possibly adding some padding, this procedure gives us a box containing the atrium.

We train our base CNN on 3 sampling procedures with no atrium box, i.e. all the non-atrium training examples are sampled randomly uniformly, a small atrium box constructed by the procedure above with a padding of 5 pixels in the x and y coordinate directions and of 1 pixel in the z coordinate direction, and finally a large atrium box with a padding of 30 pixels in the x and y coordinates and of 5 pixels in the z coordinate.

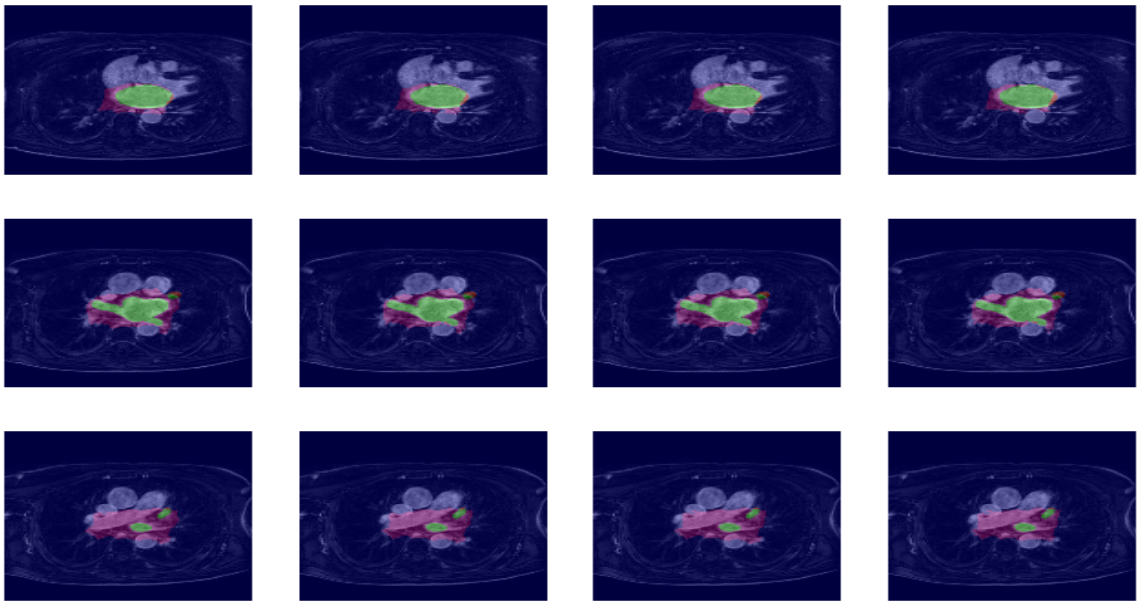

Figure 3.9: Masks for varying momentum

The results of the three training runs are shown in Figure whatever. From the testing dice coefficient plot, we get a better classification rate with sampling using an atrium box than no atrium box and particularly with the smaller atrium box. In the segmentation mask, sampling with no atrium box clearly yields more errors in the proximity of the atrium but none away from it whereas there are some errors far away from the atrium from the models trained with the atrium box sampling procedure. This is a consequence of the sampling procedures in both cases. Sampling with an atrium box would naturally yield better segmentation results near the atrium as the proportion of training examples is much higher in those regions than without an atrium box around it.

3.4.11 varying the data size

Tried a number of dataset sizes: 400000, 1000000, 3000000

4 Conclusions and Further Work