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

31st 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

1.1. Deep Learning and Convolution Neural Networks

Neural networks, or Deep Learning as it is now referred, is a field of artificial intelligence that has attracted a lot of attention recently [4]. Over the last few years, they have become state of the art in a number of international contests in pattern recognition, particularly image recognition, speech recognition, and natural language processing[7].

Most of the difficulties 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 generalisation properties. Traditionally this problem is tackled 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 content. By contrast, neural networks aim to learn the relevant features as part of the learning process itself, by training a hierarchical network of simple processing units called neurons or nodes on a large set of data in order to extract a good representation model of the data. Learning in this context means finding a set of weights which makes the neural network exhibit the desired behaviour (e.g. classify hand written-digits correctly).

The first neural networks with multiple, if few, layers date back to the 60s [reference] and have been around for decades. For feedforward neural networks, where the information can only one way through the network, an efficient gradient based method that is widely used called the backpropagation method was developed then and applied to neural networks in 1981 [reference]. However a number of practical difficulties made it difficult to train architectures with more a small number of layers. In particular, one common issue was the so-called neuron saturation problem [reference] where I need to explain what that means.A number of practical solutions where offered [reference], from the choice of parameter initialisation to the influence of the various possible activation functions on the learning behaviour of the network [reference]. However in 2006, a seminal paper [reference] by Hinton made training 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.

Another major recent development was the introduction of convolutional neural networks (CNNs) by LeCun [reference]. CNNs are a specialised type of Neural Network for certain grid-like structures with features that are strongly localised. In particular, they have

been remarkably effective on classification tasks on images, videos, and audio recordings, providing state of the art performance on object and speech recognition tasks. They were introduced in 1998 in the context of hand-written digit recognition on the MNIST dataset. Ever since they have won many official international pattern recognition competitions, in particular the Imagenet challenge. A full historical account can be found in [reference].

1.2. Deep Learning in Medical Imaging

Medical Imaging is a set of techniques to create visual representations of the interior of the body using technologies such as X-rays or ultra-sounds. It's aim is to provide a non-invasive way of making diagnosis by revealing internal structures. Interpreting these images have traditionally been done by trained clinicians such as radiologist or histologists, involving a time consuming and expensive process. Current efforts are being made to automate this process [reference] in order to improve the accuracy, speed and cost of diagnoses.

Following the string of recent success of Deep Learning approaches in the various contexts, there has been a recent effort to implement these models in the context of Medical Imaging. These efforts include for instance the anatomical segmentation of the entire brain [reference], the segmentation of the lungs for cancer detection [reference], biological neuron membrane to map 3D brain structure and connectivity [reference], 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 Master's thesis aims to segments a

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 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 [1], [2] and [5].

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 [8] where at every epoch, a percentage of hidden units are randomly deactivated during training has been shown to give better generalisation performance.

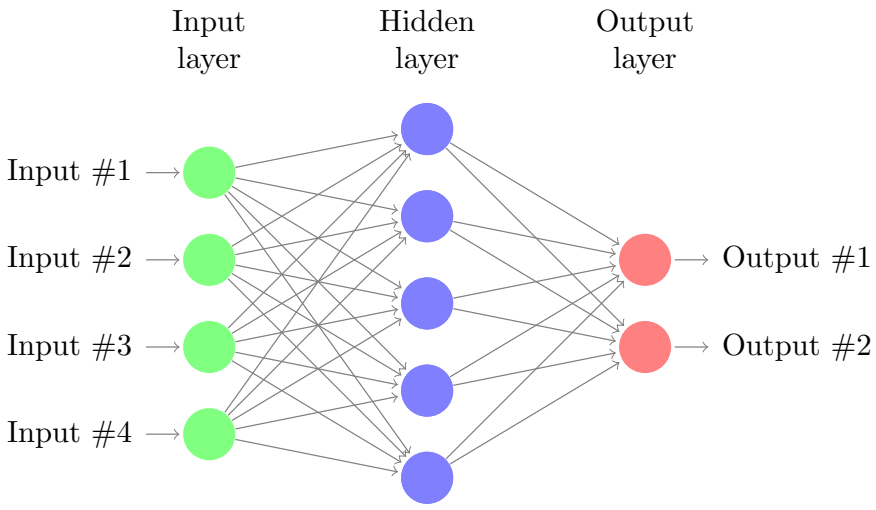

Figure 2.1.: A 1 layer-feedforward neural network with 4 input nodes, 1 hidden layer with hidden nodes and 2 output node.

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 2.2.1 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

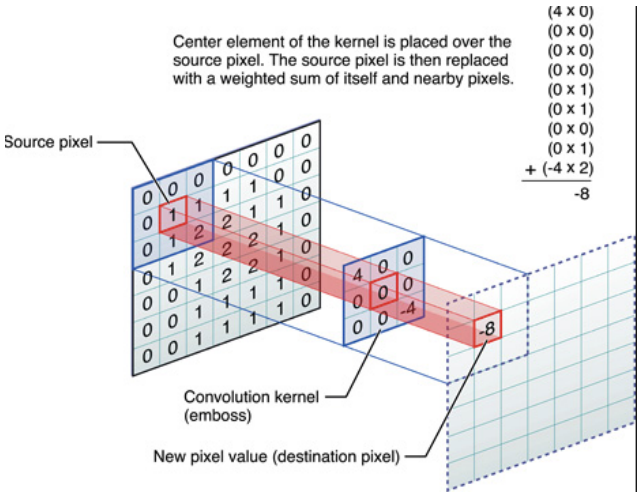

Figure 2.2.: The convolution layer. Illustration of the formation of a feature map.

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 2.2.1 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

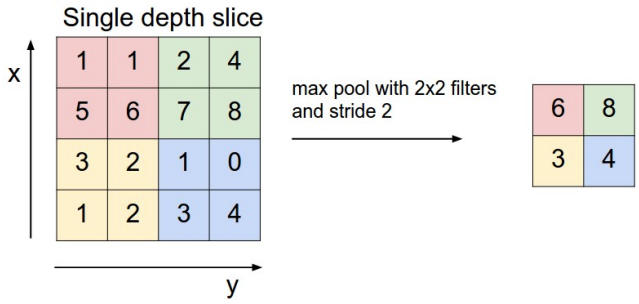

Figure 2.3.: Subsampling in action. Here maxpooling is used to halve the size of an feature map using a 2×2 filter

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 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.

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 two 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 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 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 3.2. This technique has been found to give competitive results compared to using 3D patches while being much more computationally and memory efficient [6]. In addition, we use a multi-scale approach as in [3], 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.

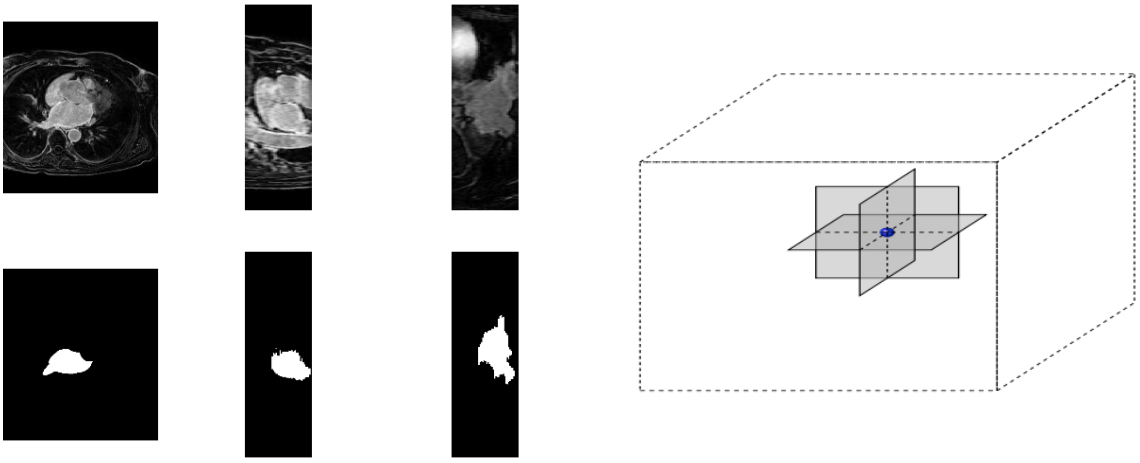

Figure 3.1.: Left: grey scale slices from a CT scan taken in the transversal, saggital and coronal planes. Right: illustration of the triplanar

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

The purpose of this section is to find a good set of hyper-parameters fro our CNN. To that end, we allocate 20 of the 27 CT scans for generating the training set and the

remaining 7 for generating the validation 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 from the testing set to provide performance statistics. In particular, from this segmentation image, we evaluate 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 in the regions of 80 to 90%.

Additionally, mask images are generated from 3 transversal slices in the segmented CT. 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	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
1	0.959	0.956	0.913	0.970	0.969
2	0.958	0.961	0.734	0.989	0.984
3	0.956	0.960	0.762	0.987	0.983
4	0.959	0.956	0.881	0.973	0.972

The table above presents the key test statistics across the 4 architectures considered. The architecture with the best test Dice coefficient is the one with 2 convolutional layer, 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 respectively. However 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 whatever brings a visual element to this comparison. 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

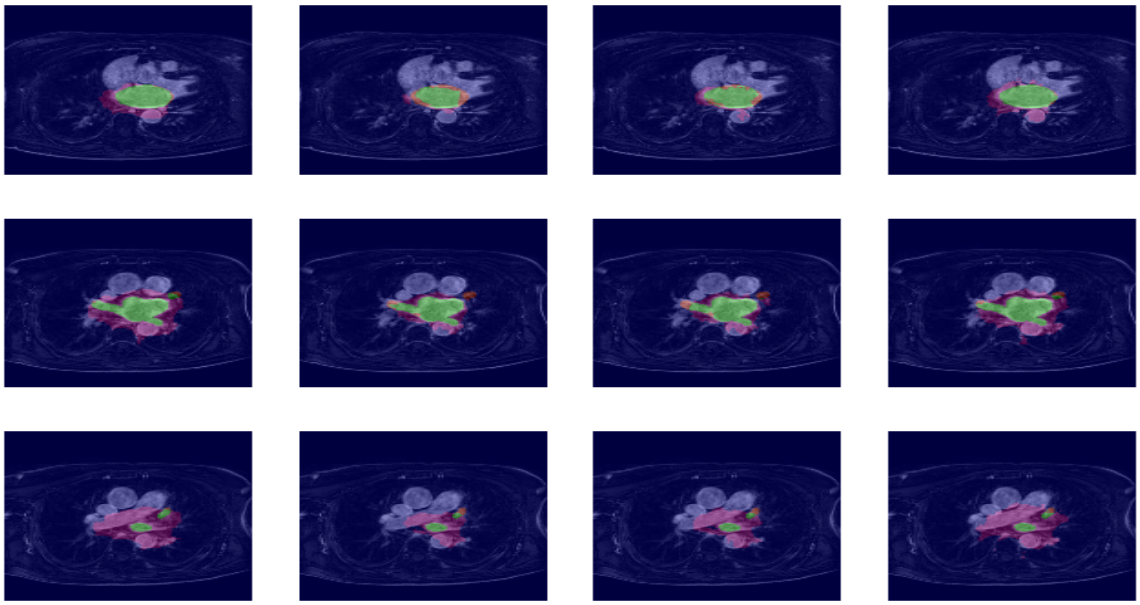

Figure 3.2.: Masks for varying number of convolutional layers

layers.

We choose to go with the architecture with the highest Dice coefficient 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.

# Connected Layers	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
1	0.957	0.959	0.887	0.972	0.971
2	0.958	0.961	0.856	0.974	0.972
3	0.953	0.951	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 that have the highest Dice coefficients of the three have 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 whatever similarly shows very little differences in the colour

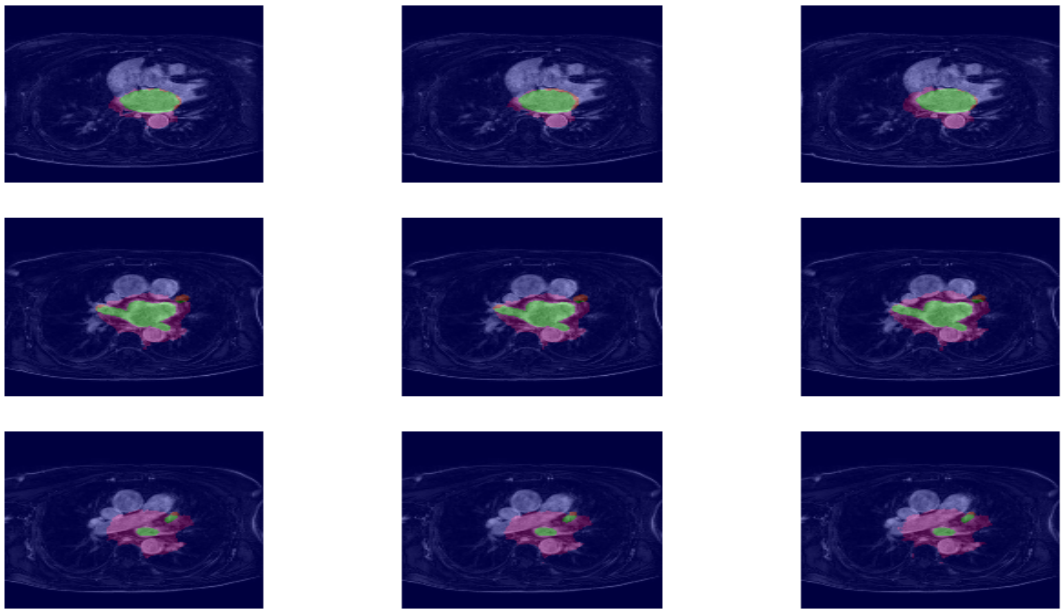

Figure 3.3.: Masks for varying number of connected layers

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.

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.

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

Yet again there is not much difference between the performance of all three architectures. The two with the highest test Dice coefficients are the ones with the first layer having 32 and 64 feature maps both at 0.970 and very similar sensitivities around 0.9. The one 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. The masks show slight differences in the first row of images, with a larger pink region for the 16 feature map architecture.

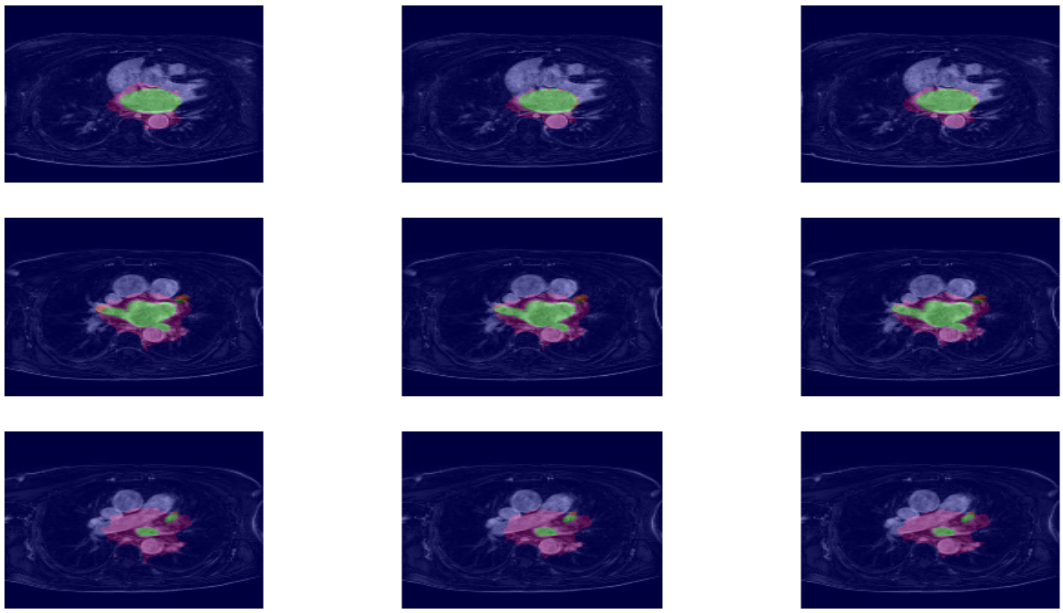

Figure 3.4.: Masks for varying number of feature maps

These statistics don't justify having twice as many feature maps, we opt for having 32 feature maps i the first layer of the 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	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
100	0.956	0.958	0.879	0.972	0.97
200	0.958	0.957	0.878	0.971	0.97
500	0.957	0.955	0.894	0.97	0.968
1000	0.957	0.958	0.892	0.972	0.97

Here the test statistics across all 4 architectures are very similar with a specificity around 0.97 and a sensitivity around 0.89. Given the tie, we select the simpler model with 100 hidden units.

3.4.6. varying the learning rate

Tried different learning rates: 0.01, 0.05, 0.1, 0.5. Huge improvement as we increase the learning rate. A learning rate of 1 doesn't converge. We take LR = 0.5.

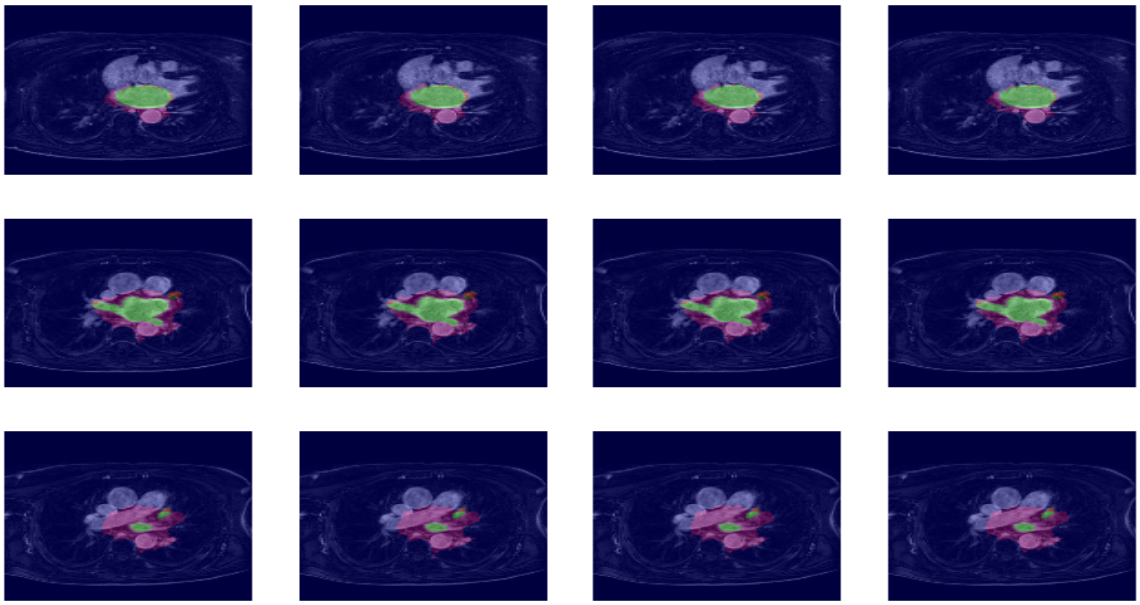

Figure 3.5.: Masks for varying number of hidden units

Learning Rate	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
LR 0.01	0.956	0.957	0.885	0.971	0.97
LR 0.05	0.97	0.97	0.915	0.977	0.976
LR 0.1	0.973	0.973	0.916	0.98	0.978
LR 0.5	0.982	0.973	0.932	0.982	0.981

3.4.7. varying the momentum

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

Momentum	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
0	0.982	0.973	0.906	0.982	0.981
0.05	0.979	0.973	0.916	0.98	0.979
0.1	0.983	0.976	0.9	0.984	0.982
0.5	0.985	0.981	0.83	0.988	0.985

3.4.8. varying the activation function

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

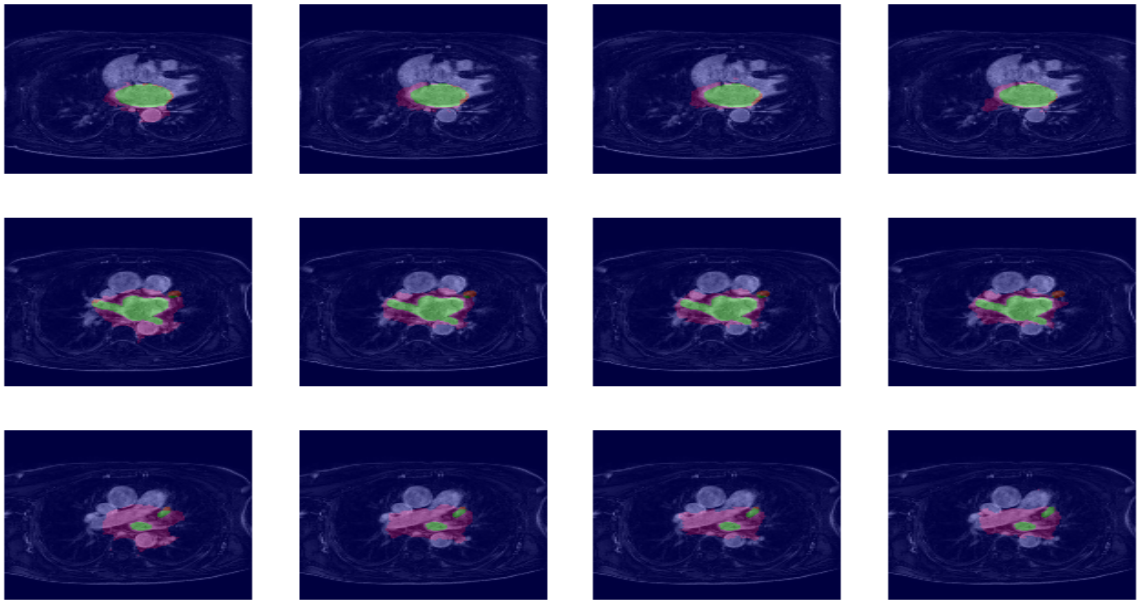

Figure 3.6.: Mask for varying learning rate.

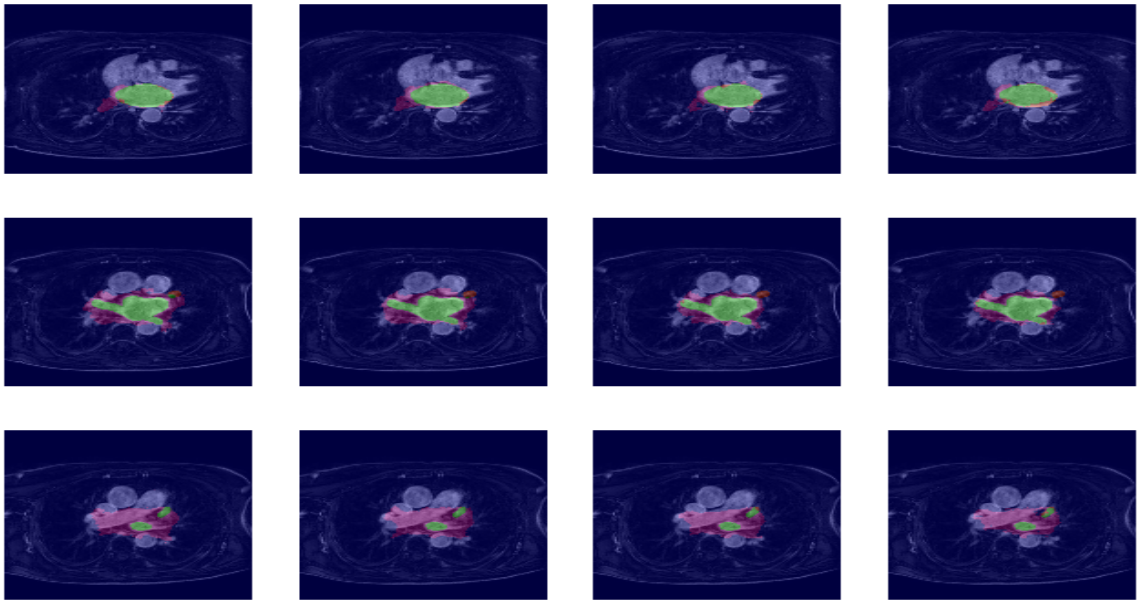

Figure 3.7.: Masks for varying momentum

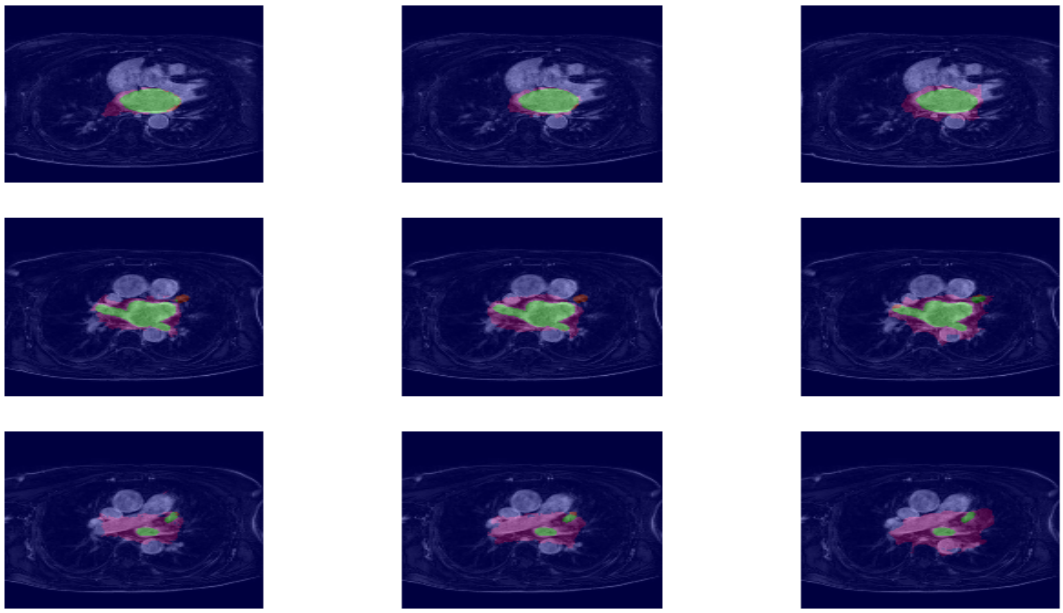

Figure 3.8.: Masks for varying activation function

Activation Function	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
ReLU	0.984	0.978	0.87	0.986	0.984
Tanh	0.973	0.972	0.934	0.98	0.979
Sigmoid	0.958	0.958	0.941	0.968	0.967

3.4.9. 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.

The results of the three training runs are shown in Figure whatever. From the testing

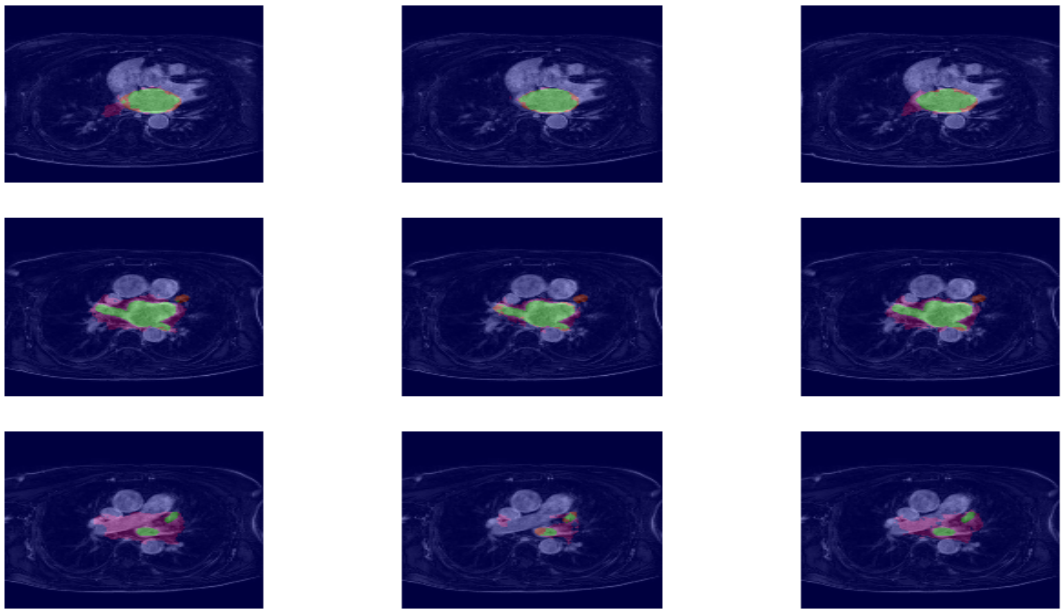

Figure 3.9.: Masks for varying training dataset

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.

Training Dataset	Dice training set	Dice testing set	Sensitivity	Specificity	Dice test CT scan
No Atrium Box	0.982	0.978	0.881	0.986	0.984
Small Atrium Box	0.96	0.987	0.838	0.994	0.991
Large Atrium Box	0.963	0.983	0.871	0.989	0.987

3.4.10. varying the data size

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

4. Conclusions and Further Work

Bibliography

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

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

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

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

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

[6] 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.

[7] J. 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].

[8] 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.

Appendices

A. Some Appendix

We will only be showing the two most important parts of our code for training neural networks: the model and the training function. In the interest of brevity, we omit the rest which have to do 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
```