UCL DEPARTMENT OF MEDICAL PHYSICS AND BIOMEDICAL ENGINEERING

Undergraduate Project Tracking Sheet

Module Code:			
Module Title :	**Medical Physics and Biomedical Engineering Research Project**		
Coursework Title :	**Final Report**		
First Supervisor		**1st supervisor's e-mail**	**j.mcclelland@ucl.ac.uk**
Second Supervisor		**2nd supervisors e-mail**	**a.szmul@ucl.ac.uk**
Student's Name:	**Jack Sambourn Weeks**		
Student's e-mail address:	**15064668**		

1) **Hand in a hardcopy to the main Reception Office (2nd Floor, MPEB).**

Coursework Deadline:	**8/04/19**
For Office Use Only	
Date Received:	
Date Returned to Student:	

Before completing and submitting this piece of coursework please check that you are fully aware of the MPBE policy on plagiarism (See your Student Handbook).

Also note the penalties for late submission which can be found in the Academic Manual at the following link:

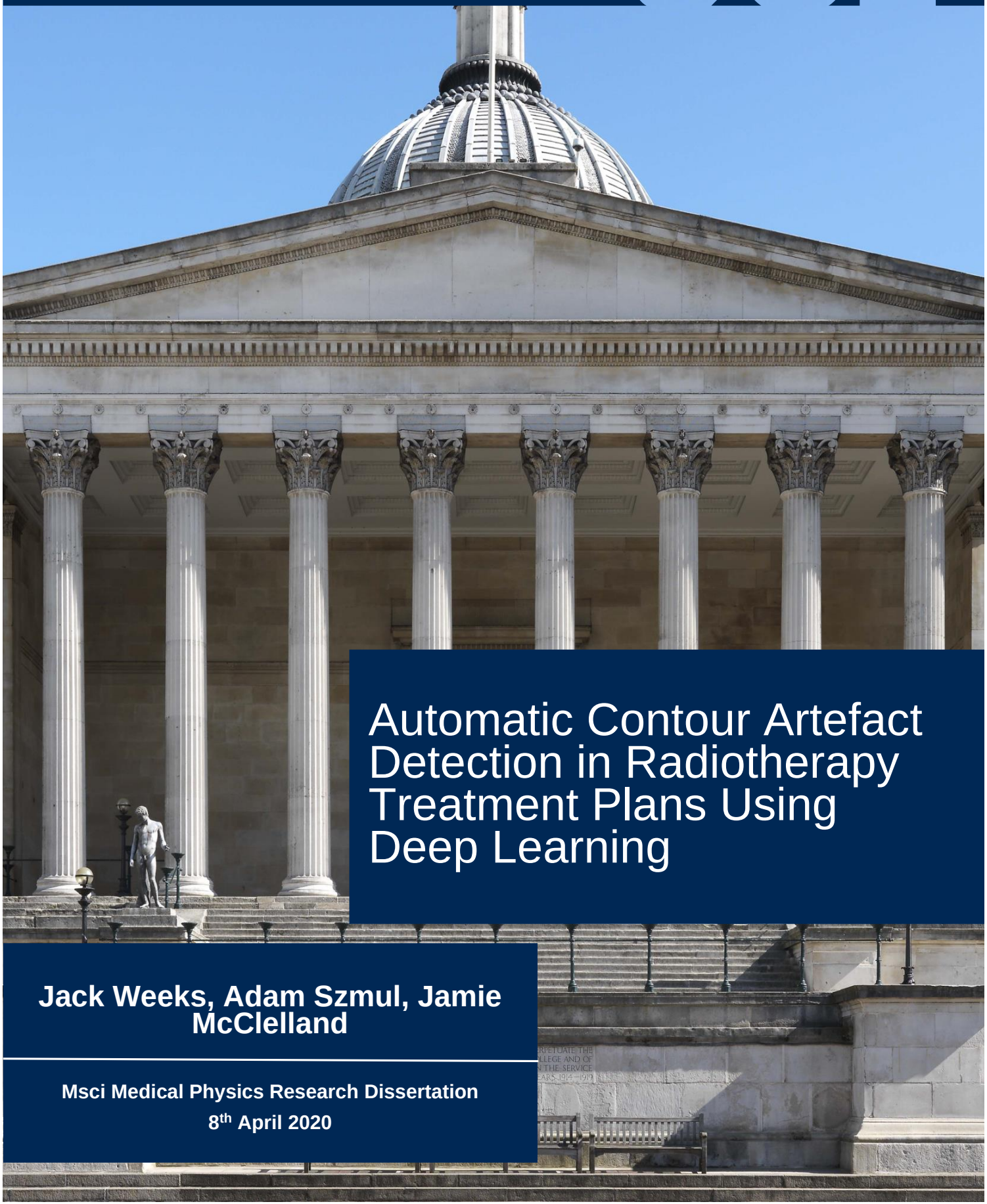
http://www.ucl.ac.uk/basc/current/assessment/latesubmission

Mark (%):

Please note that the mark is provisional and could be changed when the exam boards meet to moderate marks.

For Office Use Only
Reason for late submission of course work:

UCL DEPARTMENT OF MEDICAL PHYSICS AND BIOMEDICAL ENGINEERING



Automatic Contour Artefact Detection in Radiotherapy Treatment Plans Using Deep Learning

Jack Weeks, Adam Szmul, Jamie McClelland

Msci Medical Physics Research Dissertation

8th April 2020

Automatic Contour Artefact Detection in Radiotherapy Treatment Plans Using Deep Learning

Jack Weeks[1], Adam Szmul[1], and Jamie McClelland[1]

University College London, London UK
Department of Medical Physics and Biomedical Engineering
8th April, 2020

Abstract This project intends to enhance the efficiency of radiotherapy treatment workflow by developing a tool to allow clinicians to automatically detect errors in OAR delineation. OAR delineation is essential in ensuring patient safety when undergoing radiotherapy. We aim to provide an alternative to the current method of peer review to detect artefacts in treatment plans by employing deep learning. We make use of a 3D Deep convolutional neural network to analyse delineations of five organs in the upper thoracic region to train models based on the VGG network architecture, two architectures are evaluated. We employ a patch based approach to reduce memory requirements and improve robustness of training, with aims of implementing a localisation tool. Results show both architectures used for experimentation were successful with a maximum error rate of 11.7%

Keywords: Convolutional Neural Network (CNN) · Deep Learning · Healthcare AI · Radiotherapy · Cancer Treatment.

1 Introduction

1.1 Medical Significance

Radiotherapy is the most common cancer treatment in the NHS. The nature of the treatment requires the tumour to receive the correct dose to kill the cancer cells, while minimising dose to surrounding healthy tissue. The treatment plan must be precise to ensure the full dose is administered, failing to do so reduces the effectiveness of the treatment. Irradiation of healthy tissue may result in unnecessary side effects to the patient, such as secondary tumours.

A CT scan is taken prior to treatment such that the tumours (targets) can be carefully delineated from Organs At Risk (OARs). The tumour is enclosed in a gross target volume which must be adequately irradiated to achieve a cure. Delineations are either done manually or with computer assisted methods, both of which are subject to errors. Some of these errors include missing slices and isolated voxels [1]. Left unchecked these errors endanger the patient's success of

cure by degrading the quality of contour based computer assisted interventions and inaccurate dose distributions [2].

Artefacts can originate from brush stroke errors, these result in small round circles, external to the contour as shown in Fig.1a. Another contouring artefact is the presence of anatomically incorrect geometries as shown in Fig.1b. The next most common type of artefact observed arises as a result of a slice missing in the CT volume. Clinicians do not manually delineate every slice and thus if the slice thickness is small interpolation is used between every few slices. Resulting in large rectangular contours as shown in Fig.1c

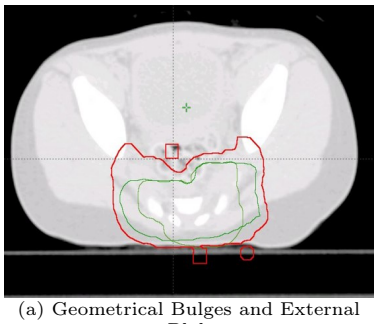
These artefacts can often remain unnoticed until the clinical target volume (CTV) contours are expanded to form the Planned Target Volume (PTV). The tumour's contours are expanded by 5mm - 10mm to allow for any uncertainties in the planning or treatment delivery [3]. Currently peer review is the most common way of resolving these errors. Once detected a clinician goes back and manually corrects the delineation. While peer review seems a reasonable approach to validate the contours, manually checking every slice of an anatomical volume is time consuming and requires the clinician/physicist/dosimetrist to be available and could still result in errors. Peer review is often dependent on the reviewers experience. Additionally technological inefficiencies exist such as poor resolution screens and the lack of quantitative evaluation tools [4]. The time lost due to these artefacts depends on whether the artefact significantly affects the treatment plan and when in the full treatment plan they are detected.

This research intends to fulfil the lack of quantitative evaluation tools available to clinicians to efficiently and accurately validate OAR delineations. We propose a tool based on a Deep Convolution Neural Network (CNN) which will be able to detect these artefacts in real time allowing the clinician to self validate and correct before the treatment plan proceeds. We envision a tool such as this to save the healthcare industry valuable time, while increasing patient safety by equipping clinicians with the right tool to ensure that every treatment plan is as accurate as possible.

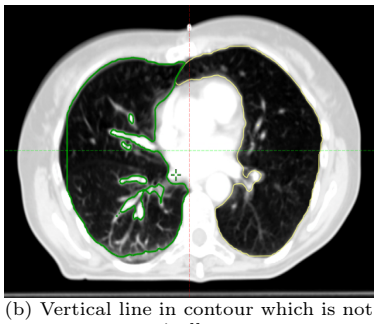
1.2 Computer Science Techniques

Artificial neural networks are computational models inspired by neurological processes in the human brain. The rapidly developing in field can be applied to areas such as speech recognition, computer vision and text processing. Like the brain, the building blocks of a neural network are named neurons. Neurons take a number of inputs and produce an output, much the same as a mathematical operator. These operators are known as activation functions, we'll touch on those in more detail later. A layer in a neural network is a collection of neurons, in all neural networks, the first layer is referred to as the input layer and the final layer is called the output layer. Between them are hidden layers, the role of a hidden layer is to transform the information from the input layer to something the output layer can use to interpret patterns in the data to provide some useful output, such as classification. An example of a simple neural network is shown in Fig.2.

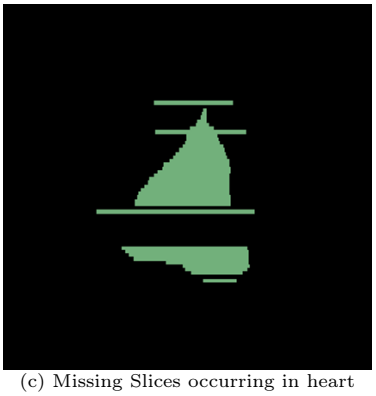
Figure 1: Examples of artefacts seen when delineating OARs from Tumours



(a) Geometrical Bulges and External Blob



(b) Vertical line in contour which is not anatomically correct



(c) Missing Slices occurring in heart contour

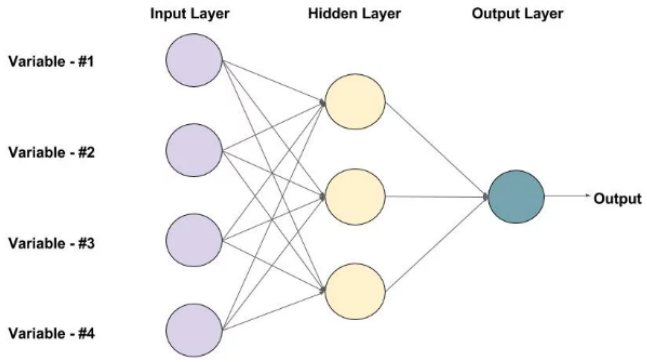


Figure 2: An example of a simple neural network, with four input neurons, a single hidden layer containing three neurons and a single output neuron

These networks can be made deeper by adding more hidden layers, typically increasing the number of hidden layers increases the complexity of the function the network can learn. To appreciate what is meant by the network learning, we must first expand on the connections between layers. The connections between layers are weights, in a standard neural network every neuron in one layer is connected to the next via a weight. It is the sum of these weights which the activation function uses to calculate the neuron output. These weights are not fixed and are updated during training. The weights are updated to minimise a loss function, which compares the output of the network to the 'ground truth', this is called supervised learning and is the most common type of neural network currently.

How Do Convolutional Neural Networks Interpret Images

A convolutional neural network is a variant of a standard neural network and gets its name from the type of hidden layers which make up the network. The hidden layers are formed of convolutional and pooling layers before the outputs of these layers are passed into fully connected layers such as the ones shown above. The main difference between these two types of networks is the neurons are replaced with filters which carry out the convolution operation. These networks commonly use a number of these convolutional layers and as such are described as deep neural networks.

A CNN is designed to learn spacial hierarchies of features, such as edges and patterns. A single convolutional layer contains filters each identifying a different feature, the output of the previous layer is fed into subsequent layers forming a multi scale network of connections. The weights of each filter are updated during training. As the hierarchy of the CNN increases combinations of these filters allow the network to identify increasingly complex patterns.

They key parts of a CNN are:

– Convolutional Layer and Pooling layer - Perform feature extraction
– Fully Connected Layer - Maps extracted features to a label

Training a network means optimising parameters (filter weights) to reduce the difference between the predicted label and the ground truth label.

1.3 Building Blocks of A Convolutional Neural Network

CNNs are used in machine vision tasks because of their ability to process data in a grid like fashion such as image data. To a computer, images are large matrices, with each pixel containing a number as shown in Fig.3, in this project we use a binary mask and as such, every pixel has the value of zero (black) or one (white). The filter can be thought of as a small window, commonly 3x3 pixels which moves over the image in steps called strides, each stride produces a single value in the feature map. The larger the stride value the more down sampled the feature map is in relation to the original image. Down sampling can also be achieved using pooling layers (see pooling section). The filter is also a matrix,

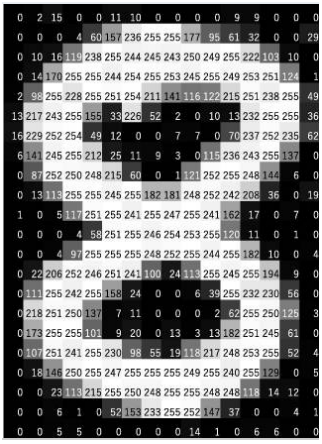
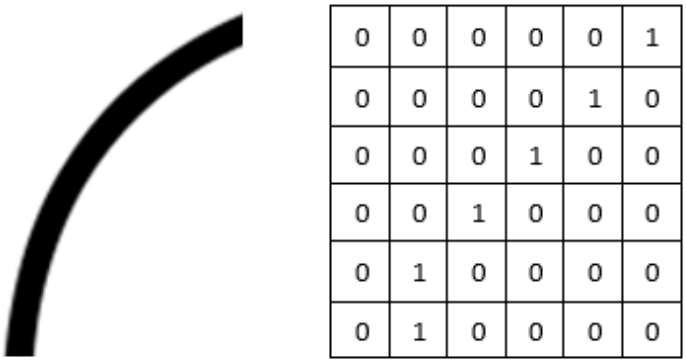


Figure 3: An image of a number 8, overlayed with a matrix of it's pixel values, taken from MNIST dataset [5]

a simplified example is shown in Fig.4 and by matrix convolution determines if the feature is present. Fig.5 shows a 3x3 window moving over an image in strides of one. The original image has no padding and therefore will be down-sampled in the feature map. Another set of filters can be applied to the previous layers output, resulting in an abstract map of features from the original data.



0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

Figure 4: A simplified example of what a curve detection filter looks like in image and matrix form, adapted from [6]

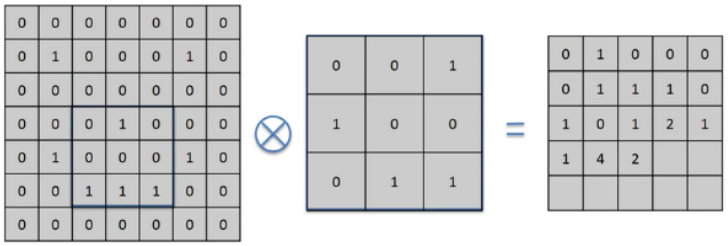


Figure 5: Example showing how an input image (left) is multiplied by a feature detector (centre) to form a feature map (right). The feature map is downsampled from the input image due to the original image not being padded and the 3x3 window taking strides of a single pixel across the image, adapted from [7]

Activation functions The roll of an activation function is to provide non-linearities to a deep learning network, without doing so the network effectively consists of a single layer. The non linearity added by activation functions allows networks to learn complex relationships between variables which may not have a linear relationship. The activation function in a neural network collects the weighted sum of the outputs from the previous layer and them through a function which outputs a number. The output limit varies depending on the type of activation function. We will be focusing on Rectified Linear Unit (ReLU) activation functions as they're the most commonplace for this type of algorithm

[8]. A plot of the ReLU function is shown in Fig.6. From the figure we can see if the weighted sum input is less than 0, the output will be 0, if the input is greater than 0 the output will be linear, this allows ReLU to converge much faster than the sigmoid and Tanh alternatives, which outputs range between (0, 1) and (-1, 1) respectively. Importantly although ReLU is linear in the positive axis, combinations of ReLU are non-linear. However ReLU is not resistant to the vanishing gradient problem. The vanishing gradient problem occurs when we back propagate and differentiate the activation function, ReLU is resistant to this problem in the positive region, however susceptible in the negative region, this can cause the weights to not update and the model to not learn anything, killing the neuron.

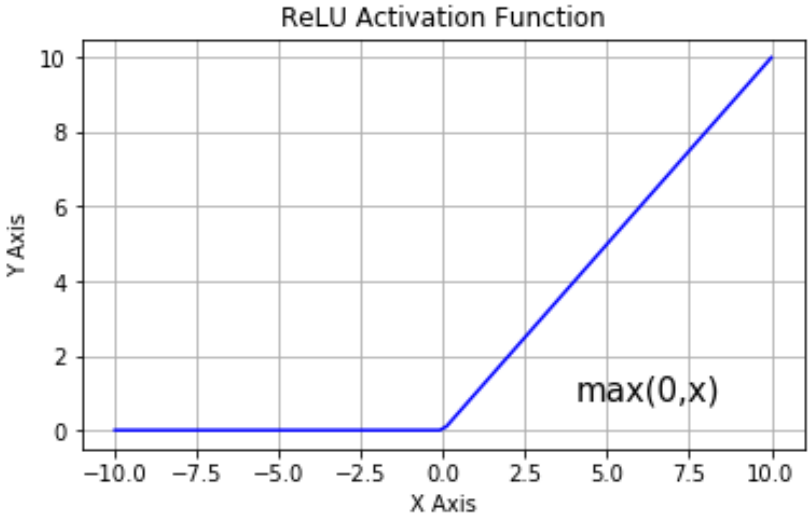


Figure 6: The ReLU activation function, the function rectifies at x = 0 and is linear along the positive axis. [9]

The softmax activation function is used at the end of a network. The roll of the softmax activation function is to convert the numerical output from the fully connected layers into a vector. The length of the vector corresponds to the number of classes in the labelled dataset. The softmax function converts the inputs x_i into a probabilities by taking the exponent of each input and dividing it by the sum of all the exponents (1). This normalisation ensures that the sum of the values in the output vector sum to one. The maximum value in the vector is taken to the classification.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum e^{x_j}} \tag{1}$$

Pooling Each feature map generated after passing through convolutional layers is typically passed through a pooling layer. The pooling operator takes a small square, 2x2 or 3x3 pixel region of the feature map, and produces a single value output for the region. The most common forms of pooling are Max pooling, whereby the largest value in the pooling grid is passed as the output, or average pooling, where an arithmetic mean of the grid is passed to the output. Pooling layers provide some translational invariance to the CNN. The main function of pooling - downsampling, can be realised by constructing a convolutional layer with a larger stride. This can be advantageous as pooling is an operation, while convolutions can be learned. Pooling is computationally cheaper as fewer computations are run and no parameters need to be stored by the model. Springenberg et al 2014, showed that replacing all the pooling layers with additional convolution layers model performance stabilises and improves on the base model.

Fully connected layers The fully connected layers connect every neuron in one layer to that in the next, much like the example shown in Fig.2 In the final fully connected layer the number of neurons corresponds to the number of given classification labels, the output of this layer is passed into the softmax activation function where the outputs of the final layer are transformed into probabilities and the loss can be calculated.

Loss The loss function drives the network learning process. It is the metric that the model optimises during training. The loss is calculated and the model takes a step in the direction down the gradient of decreasing loss as shown in Fig.7. We use Cross Entropy Loss as the loss function for this task which is a simple way of comparing the output vector of predictions from our model to the true value label. Our model is a binary classifier we essentially have Binary Cross entropy loss which is outline in Equation (2)

$$J(w) = -\frac{1}{N} \sum [\ y_n\ log\ \hat{y}_n\ + (1 - y_n)\ log(1 - \hat{y}_n)] \tag{2}$$

With y_n being our true classification, either 0 or 1 and our predictions being $\hat{y}_n$. The equation becomes much simpler if our true label is 0 then the equation becomes

$$J(w) = \frac{1}{N} \sum [log(1 - \hat{y}_n)] \tag{3}$$

and if our true label is 1 we calculate

$$J(w) = \frac{1}{N} \sum [log\ \hat{y}_n] \tag{4}$$

1.4 Regularisation Techniques

Overfitting Overfitting is when the model learns the training set too well and over optimises to fit the training data, i.e getting near 100% accuracy. The model

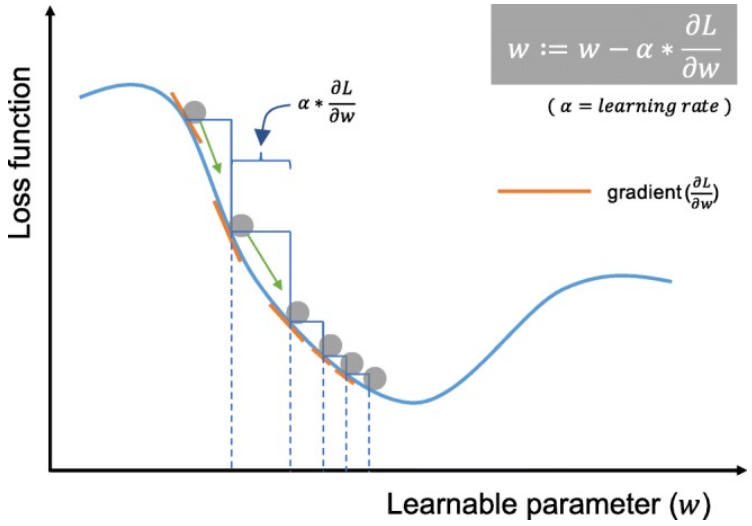


Figure 7: Depiction Of how a model optimises based on the loss function, the learnable parameter (x-axis) is (learned) such that the gradient in the loss function is minimised [10]

is then unable to such success on unseen data. Overfitted models will have low training loss and high validation loss. This is often characterised by a diversion of the validation and training loss curves in training. Training loss will continue to decrease and validation loss will begin to increase.

Overfitting is a good qualitative indicator for ensuring that your model is able to learn the training data. In practical applications we need the model to be able to generalise to larger unseen datasets. To use the patterns and features learned from training in order to make predictions.

We can evaluate the generalisability of our models using a validation set. A validation set is a subsection of our original dataset which is not used in training the model, but is labelled data. We can then use our trained model to make predictions on the validation set and compare the predicted outputs to the ground truth labels.

Often, a cross validation approach is used. Cross validation is a method of model analysis with several different variants as described by Stone 1974 [11]. The most common being K-fold cross validation whereby the entire training dataset is divided into 10 equal splits called folds. Training is then carried out on nine of the folds and validated on the remaining fold. This process is repeated K-1 times, until every fold has been used as a validation set, and model performance recorded. The performances from all runs is then averaged to provide an overview of the models performance, preventing a single overfitted fold from skewing results.

K-fold cross validation is used to finely tune the hyperparameters of a model and to evaluate the performance of the model on a wide range of 'unseen' data.

For each new iteration of k-fold cross validation the model is trained from scratch therefore ensuring the results of each fold are independent. Although the holdout fold is commonly referred to as the test set, final testing is not done on this data.

Testing is carried out on a separate dataset, assigned and left out before any training and validation splits are done. This is to ensure that once the model is ready to be tested, the input data is completely new just as if it were being used in a clinical environment.

NarasingaRao et al 2018 [12] did a survey on the prevention of overfitting and concluded steps to reduce over fitting are:

- Add More data
- Use Data Augmentation
- Use Architectures Which Generalise Well
- Add Regularisation Techniques
- Reduce Architecture Complexity

Dropout A common regularisation technique is called dropout. Dropout means temporarily removing a randomly selected percentage of nodes and their connection pathways for a single training iteration. The aim is to reduce the chance that a node 'co-adapts' [13], reducing their direct dependencies upon one another, a significant proponent of overfitting. An example of dropout is shown in Fig.8

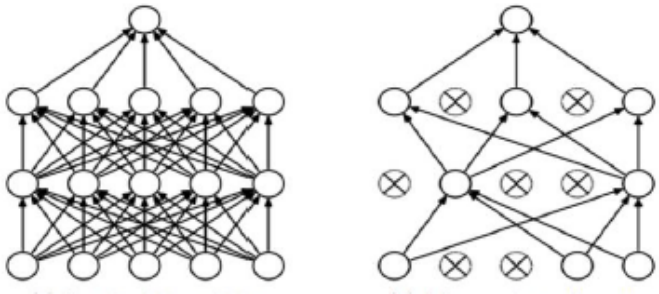


Figure 8: Depiction of dropout, we can see that 40% of the nodes are dropped after the first layer, then 60% and finally 40%. [13]

Batch Normalisation Batch Normalisation layers are placed before activation layers. They output normalised activation maps by calculating the average and standard deviation. Then subtracting each value by the average and then divided by the the standard deviation of the batch. This centres the distribution of outputs of a Batch normalisation layer about zero.

Batch Normalisation is commonly implemented in the training of machine learning models, proposed by Ioffe et al in 2015 [19]. The idea is to standardise

the inputs several times during training. Batch normalisation was designed to speed up training by significantly reducing internal co-variate shift.

Internal co-variate shift is a phenomenon which arises from the distribution of layer inputs changing as they update during training, this can over saturate activation functions, slowing down training by requiring lower learning rates. Making the loss landscape much smoother allows for much less specific parameter initialisation. Batch normalisation also comes with the advantageous effect of regularising networks. When training with Batch Normalisation the training data is seen in conjunction with other members of the batch. After normalisation of the batch, specific training examples no longer produce deterministic values in training. Hence we utilised architectures with batch normalisation enabled to combat overfitting

1.5 CNN Architectures

An architecture is how the building blocks of a CNN are put together to form a model. Each architecture has differing hyper parameters, features of the network we modify before training the network. These are summarised in Table.1

Table 1: Table to explain the difference between parameters and hyper parameters in a CNN

	Parameters	Hyperparameters
Convolutional Layers	Kernel Weights	Kernel Size, Stride, Activation Functions, Padding, Number of kernels
Pooling Layers	No Parameters	Pooling method (Max,Average), Filter Size, Padding, Stride
Fully Connected Layers	Weights	Number of Weights, Activation Functions
Network Design		Architecture, optimisation function, learning rates, loss functions, batch size, epoch, regularisation

CNN's were most notably first used to classify hand written numbers by LeCun et al 1998 [14]. LeCun's pioneering paper showed that complete handwritten sentences could be interpreted by a computer by segmenting each letter and subsequently classifying it. In 2012 a similar approach was being taken to analyse larger images and classifying them with great success in the form of Alexnet [15]. Alexnet is a Deep CNN. Alexnet used a Relu activation function, over a Tanh, this increased the speed of training 6 fold while maintaining the same accuracy. Employed Dropout instead of standard regularisation to prevent overfitting. This was done at the cost of increased training time. The architecture itself contained 5 convolutional layers and 3 fully connected layers, the convolutional layers used filters of size 11x11, 5x5, 3x3, 3x3, 3x3 with strides of 4, 1, 1, 1, 1 respectively.

A group from Oxford University called the Visual Geometry Group (VGG) built upon the aforementioned Alexnet to develop the VGG architecture and found much success in the imagenet challenge in 2014. They developed a collection of architectures as shown in Fig.9. The fundamental differences between the VGG architecture and the original Alexnet are notably, the convolutional layers employ a 3x3 filters. This became the standard as it is the smallest possible size which still encapsulates left/right and up/down.

Initially VGG had 3 fully connected layers the first ones have 4096 input and output channels, essentially giving 4096 different classifications, while the final layer of 1000 channels corresponds to the number of output classes, which in the Imagenet challenge was 1000.

As the field develops, many network architectures are being developed each with increasing excitement and promise, none more so than U-nets [16]. U-nets are widely considered the gold standard in network architectures for applications which require pixel wise output and hence suited to segmentation problems.

Another architecture of note is Facebook's SparseConvNet[17],[18]. This network was designed to perform well on data which is sparse, such as 3d point clouds which contain much less information than traditional photo images for example. It was of interest as to how the problem of overcoming the lack of data was solved by voxelising the objects similar to that of VoxNet[19] and using convolutions which maintained the original sparsity of data as opposed to traditional approaches which decrease the sparsity with each convolutional layer. However these approaches are very specific and once again more specialised towards image segmentation problems underpinned by a Unet.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 9: Configurations of the VGG network in their 2014 paper.

1.6 Shortcomings of CNNs in Medical Image Analysis

With deep learning seeming to be everywhere in novel medical image analysis tools, it is important to remember that these methods are not infallible. While deep neural networks can produce accurate results on complex datasets, they come with many technical challenges, such as how to deal with increasing levels of abstraction and how does one evaluate what makes one model better than another. Supervised learning also requires the input of an independent specialist to label vast datasets to be used in training, this can be expensive and time consuming. Many societal changes are required for these models and their decisions to be trusted by the population.

One of the key areas in medical imaging analysis is that many images are 3D. 3D Convolutional neural networks are significantly more demanding on memory and computational resources.

Data The data utilised in many medical fields is of a sensitive nature, and hence obstacles such as data access, data protection and privacy are issues one must account for. Such data must be de-identified before use in training models. Federated learning is a popular solution which allows the training of models on data stored in many data centres, without ever accessing the data.

Current deep learning medical image analysis research utilises open source, anonymsied datasets or local research partners to generate data however these repositories often cater to specific tasks and cannot be used in training of general

networks. Most open source medical datasets are unable to accomodate for large general training datasets, such as MNIST [5], CIFAR-10 [20]

Training supervised deep learning models requires large, labelled datasets. The dataset should be balanced and representative of the types of data the model will meet in use. Failure to generate a representative dataset will bias the model resulting in poor generalisation and hence performance on completely unseen data. This is often a point of contention in medical image analysis as many tasks deep learning is being tasked with are specialist detection and relatively uncommon. Resulting in a mismatch between the volume of cases required to train a network and the frequency it will detect cases in a clinical setting.

Transfer learning is a common method of circumventing this issue, transfer learning is when a model is 'pre-trained' on another dataset and its weights and biases are transferred into your network, these networks are usually pretrained on large datasets such as ImageNet [21] a repository of 14 million 3x224x224 images for training image classifiers. Such that you only modify the final layers to accommodate for your models use. This is because the early layers in a CNN usually pertain to very basic features and retain their usefulness when analysing any type of images and thus training later layers requires less data.

Datasets can also be synthesised using augmentation techniques, this method aims to transform the training set by applying randomly assigned flips, rotations, noise filters and crops. These transformations must retain the labelled feature however. Augmentation adds some variety to the dataset such that after multiple passes of the data each image does not appear in the same form every time, resulting in reduced overfitting.

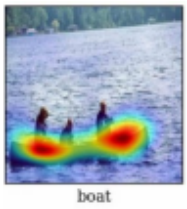
Current research is focussing on the method of automatic data synthesis, with algorithms similar to Generative Adversarial Nets (GANs) [22]. Such tools allow for the expansion of training sets by generating new data based on features of the original data.

Interpretability Interpreting Deep learning models is difficult. People supposed to trust the decision of a computer, even when its designer cannot explain decision process. DARPA, an agency of the United States Department of Defence have launched an explainable AI program which is researching methods to help to unpack this black box and help users interpret which factors contributed to the output decision.

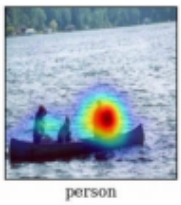
Feature visualisation becomes increasingly difficult as subsequent convolutional layers transform the input image into one of increasing abstraction, no human could possibly understand. To decode this Zhou et al propose class activation maps (CAMs) which localise the where in the image the significant regions used to make the classification [23]. An example of how their model called Grad-Cam localises regions is shown in Figs10a and 10b.

Yosiniski et al 2015 [24] developed a toolkit to help developers visualise what the filters in their CNN models were looking for. An example of the face detection filter is shown in Fig.11 and the levels of abstraction multiple convolutions produces is shown in Fig.12

Figure 10: Examples of How GRADCAM might help convince clinicians/patients how results were evaluated [23]



(a) GradCam heatmap showing how a CNN evaluates a boat [23]



(b) GradCam heatmap showing how a CNN evaluates a person [23]

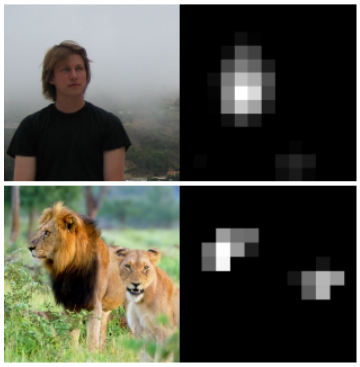


Figure 11: Yosiniski's Deep learning toolkit used to show activations in the face detection filter for a person (top) and lions (bottom) [24]

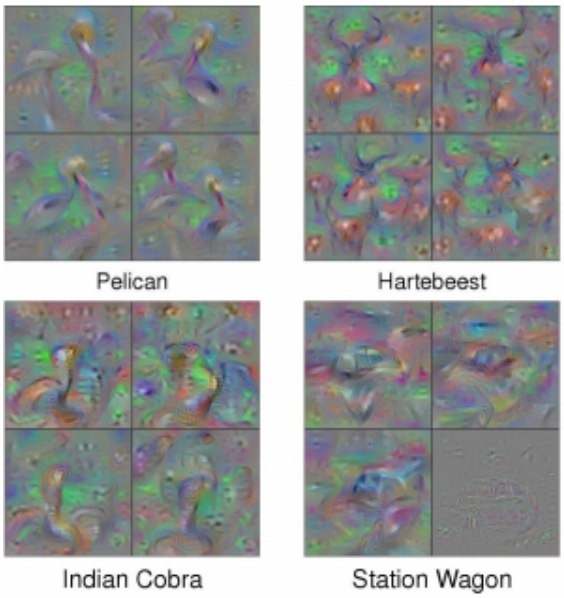


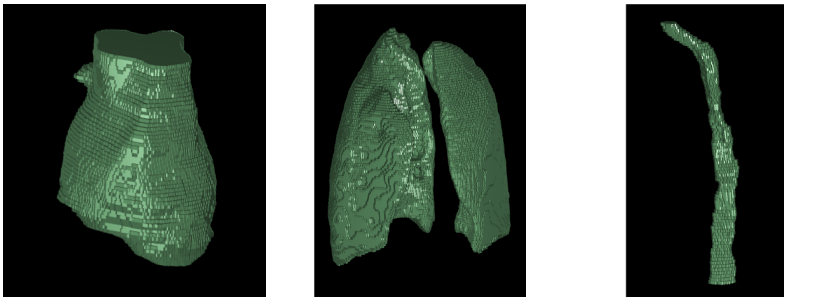
Figure 12: Yosiniski's Deep learning toolkit used to show the outputs of the fully connected layer of their CNN [24]

2 Methods

2.1 Materials

50 patients' delineations were used in the research, with two being removed due to the inability to extract suitable contours. The sparsity of data on regular clinical CT delineations lead us to ask a clinician to synthesise additional contouring errors. This was done to increase the amount of data available to train on. A patch based approach was also taken, a patch is a small sub-volume of the original CT volume. We then can select appropriate size volumes to feed into our network, this is beneficial as it allows to control the memory required to run the training. The full CT Volumes are not suitable for CNN analysis as they are too memory intensive to feed into any commercially available GPU.

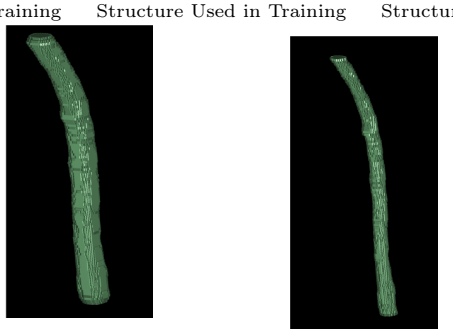
Five structures' contours were extracted from the upper thoracic region, the Heart (13a), the Lungs (13b), the Oesophagus (13c), the Spinal Canal PRV (13d) and the Spinal Cord (13e)



(a) Example of Heart Structure Used in Training

(b) Example of Lung Structure Used in Training

(c) Example of Oesophagus Structure Used in Training



(d) Example of Spinal Canal PRV Structure Used in Training

(e) Example of Spinal Cord Structure Used in Training

Figure 13: Examples of structures from the upper thoracic region of the chest, used in training the model

2.2 Data Protection

To comply with the Data Protection Act. We must ensure that all data we are working with, and publish is completely unidentifiable such that patients cannot be identified by any of the data used. The de-identification was completed by healthcare professionals before being passed on to researchers such that no data used in developing the algorithm can be traced. Subsequently data was stored on a secure password protected UCL server to ensure data was only accessible to those working directly on the project.

2.3 Extracting Contours

Data was received in the DICOM format, a common medical format which contains information pertaining to the conditions under which the scan was taken and is unnecessary for training.

The contours are saved as a list of coordinate points for each structure in the DICOM folder. Coordinates are initially in real world orientation and are first converted into the coordinate system of the CT volume. Each delineated

organ was labelled by the clinician as such we can iterate through each organ, drawing a polygon mask around the coordinates. A binary image each for each slice was generated by checking if the centre of the voxel was inside or outside the polygon. Voxels outside were assigned a value of zero and voxels inside a value of 1. The slices were then stacked to form the extracted 3D volume and saved as a Nifti file. An example of an extracted error file is shown in Fig.14.

The process of extraction was sub optimal as the exact label assigned to each organ to iterate through them had to be known prior and this was inconsistent from patient to patient. For a larger scale study an automatic label extractor should be devised as manually checking 50 patients consumed a number of hours.

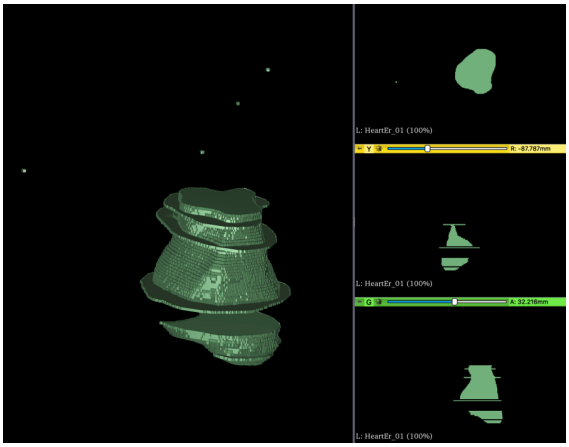


Figure 14: Extracted structure of a heart, with errors used in training the model.

2.4 Splitting into patches

The CT volumes were divided into smaller volumes as one of the aims of the research was to allow us to localise the errors. Using patches would allow us to localise on a patch by patch basis. It also allows us to have more control over the memory requirements to train the model, with larger patches requiring more memory.

The use of patches also provides more training examples to feed into our model which making training a robust model feasible.

Non overlapping patches of size 24x24x24 and 48x48x48 pixels were used, the smaller patches yielding more items of data to feed into the network. Both sized patches originate from the same original CT volumes, therefore total content of the patches would be the same. This provides an interesting offset between quantity of data and the content of the data. Manual inspection of the data shows

that some of these errors are small, thus splitting them into smaller pieces may remove the context in relation to the original contour and be counterproductive in training a classifier.

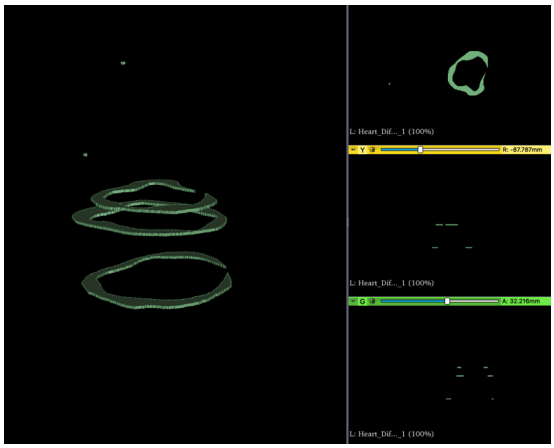


Figure 15: A difference volume used in labelling patches, generated from subtracting correct structures from error files such as the one in Fig.14

The artefact density of the training data is significantly higher than what one would expect in a clinical setting. Each patch was individually classified as opposed to the entire volume in an attempt to balance out the dataset to ensure robust training. All patches containing entirely background were removed for efficiency and to avoid diluting the already sparse data set.

Labelling on a patch by patch basis was done via a difference patch. This was done by subtracting the 'correct' patch from the 'error' patch. An example of a difference structure is shown in Fig.15. If the difference patch was empty the patch was labelled as correct, conversely if the difference patch contained any artefacts, it was labelled error. The patches were subsequently saved as numpy files, a python library which contains a large library of mathematical functions and is good at dealing with 3D arrays such as the ones describing our 3D data. .NPY files are small and efficient to read into CNN Dataloaders and contain only the information required for training and testing. The decision pathway for labelling patches is shown in Fig.16

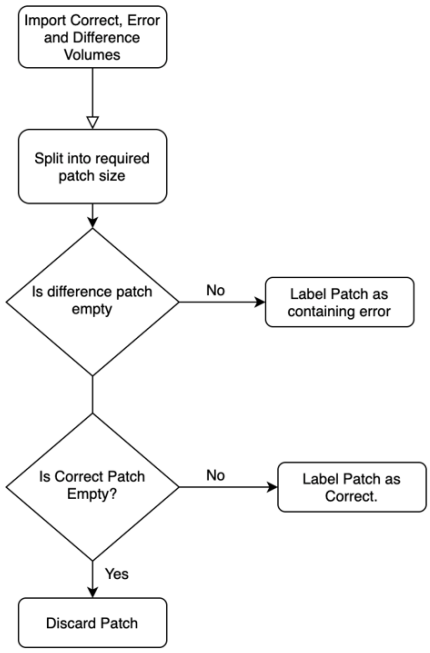


Figure 16: Flowchart describing the process we followed when labelling the extracted patches

2.5 Implementation

We chose to use the Pytorch Library, developed by Facebook, as it is well suited and widely used for computer vision tasks such as this one. Pytorch has a well supported online community, with the added advantage of it being a developed library of tools which was easy to understand and use effectively. We took advantage of PyTorch's Dataset function to assign training labels based on their file path. We wrote our own dataloader to allow the data to be passed into Pytorch's dataloader class. The 3D numpy arrays from the patch .npy files were loaded and cast them to a tensor which the PyTorch library carries out operations on. Once the file has been converted to a tensor, we applied random flips along the x and y axes to augment the data.

Pytorch's dataloader class allowed the data to be shuffled and mini-batched every epoch such that the training data does not appear in the same order for every forward pass.

The application of Pytorch also allowed us to take advantage of a large library of open source architectures which we could adapt such as the VGG network. We adapted the network to accept 3D volumes as inputs and produce a binary classification.

Tensorboard implementation was also supported by Pytorch. Graphically tracking metrics in real time as the training progresses shown in Fig.17. This is particularly beneficial during the tuning phase of model development as adjusting parameters can lead training to get stuck in local minima in the training surface making training inefficient. Tensorboard also allowed us to store these values for later analysis.

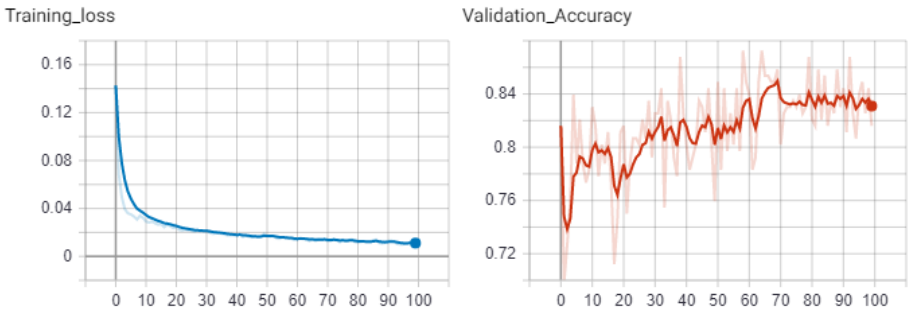


Figure 17: Example of how Tensorboard tracks metrics

Adapted VGG11 Model The VGG 11 model used in this research is depicted in Fig.18. Our model contains 4 convolutional layers containing 32, 64, 64, 128 filters in each layer respectively, a kernel size of 3x3x3 with a padding of 1. Each convolutional layer's output was passed through a Batch Normalisation layer, a ReLU activation layer and a max pooling layer. The max pooling layer used a kernel size of 2 and a stride of 2. The output of the final max pooling layer was passed into an average pooling layer. Three fully connected layers with input sizes 43904, 4096, 4096 respectively. Each layer used a ReLU activation function with 60% dropout after the first fully connected layer and 50% dropout after the second. The final linear layer outputs to a softmax function, resulting in a binary classification.

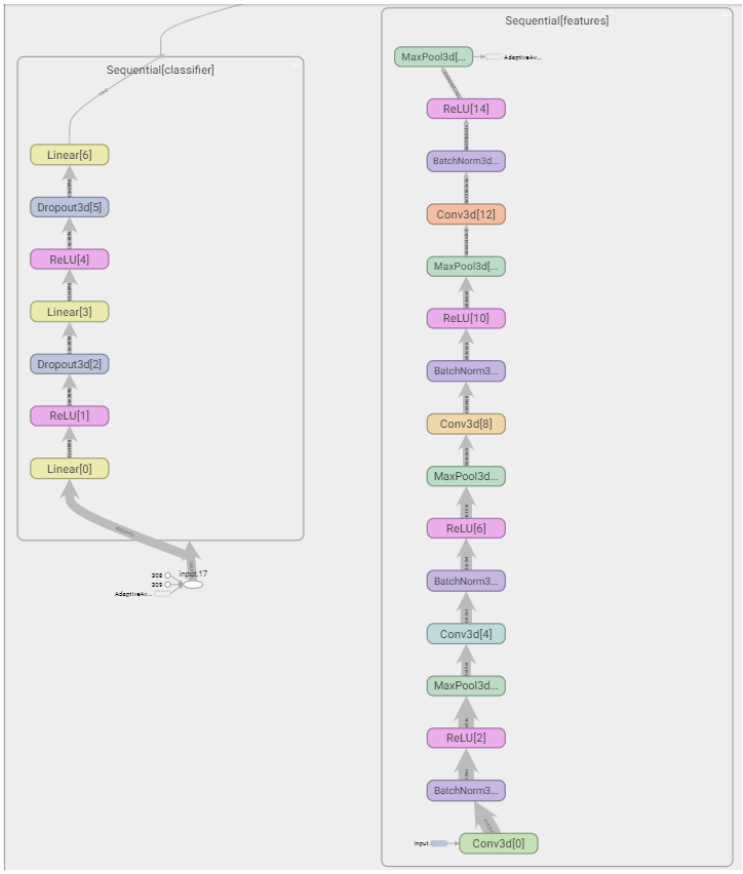
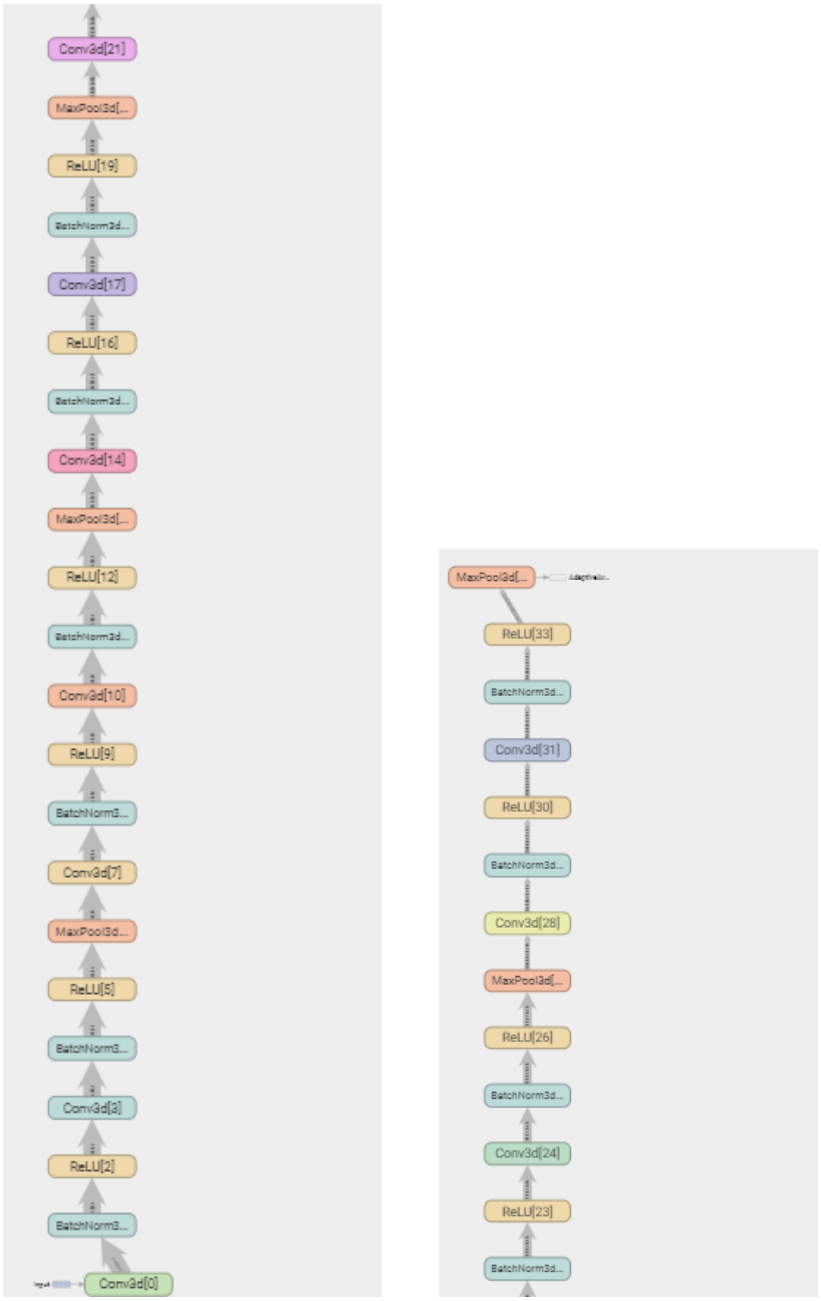


Figure 18: Our adapted VGG11 architecture, (left) fully connected layers leading to the output, (right) the sequence of convolutional, batch normisation, and pooling layers

Adapted VGG13 Model The VGG13 model used in this research is depicted in Fig.19 This model contained 10 convolutional layers, grouped in 5 double convolutional layers. Each group contained 32,64,64,128,128 filters respectively, a kernel size of 3x3x3 and padding of 1. Batch normalisation and ReLU were performed after every convolutional layer. Max pooling was performed after every pair. Max pooling had a kernel size of 2 and a stride of 2 with the output of the final max pooling layer passed into an average pooling layer. Three fully connected layers with input sizes 43904, 4096, 4096 respectively. Each layer used a ReLU activation function with 60% dropout after the first fully connected layer and 50% dropout after the second. The final linear layer output was passed through a softmax function to produce a binary classification. The fully connected layers were the same for both models and are shown in Fig.18 (left).



(a) First Half of our VGG13 Architecture

(b) Second Half of our VGG13 Architecture

Figure 19: Our VGG 13 Based CNN, (a) shows input and first set of convolutional, batch normalisation and pooling layers. (b) shows continuation of convolutional layers. Output of the final max pool layers lead into classifier layers, which are the same as Fig.18 (left)

2.6 Training

We have 1997 48x48x48 volumes for use in training. We tested mini-batch sizes ranging between 5 - 50. A mini-batch is the number of items of data we feed into our network at one time. The size of the mini-batch primarily affected the speed of training, with the larger mini-batches training faster, however occasionally caused memory issues. A mini-batch size of 15 was selected as a compromise of the two. The mini-batch size could also be adjusted to accommodate for any memory requirements in training, for example if a larger patch size were used, a smaller mini-batch may be required.

Training was carried out on a Nvidia GTX 2070 8GB GPU, not the UCL cluster as originally planned due to unforeseen technical, and global circumstances. Initial testing of the model, Fig.20 strongly supports the application of a CNN is a viable back-end for the development of the proposed analysis tool. The model quickly learns the training set, reaching an accuracy of 99.8% , 40 incorrect classifications of the training set, after 200 epochs. However this model displayed strong evidence of overfitting as shown in Fig.21 where at around epoch 20 we can see a strong divergence in the loss values. The training loss continues to decrease as the model becomes more optimised to the training set however the validation increases with each iteration, suggesting our model is getting worse at classifying unseen data and thus poor at generalisation.

We used an exponential moving average to fit our data, this smoothes the noisy nature of training and is inline with how Tensorboard smooths their plots.

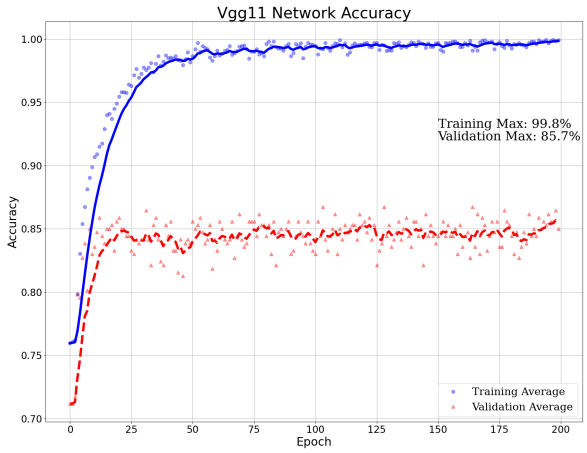


Figure 20: Training (Blue) and validation (Red) accuracy for the VGG11 model on training and validation sets respectively. Maximum values are calculated from the exponential moving average which is fitted

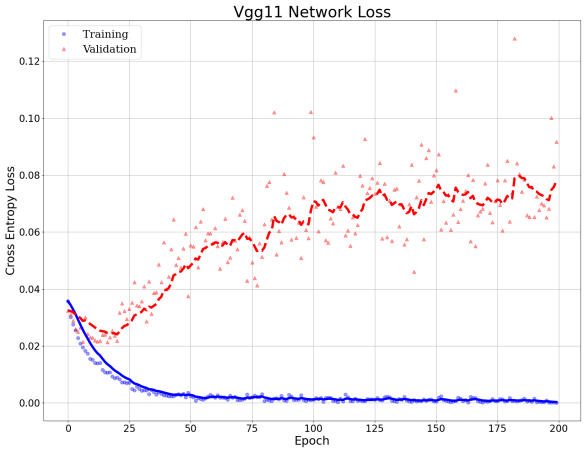


Figure 21: Training (Blue) and validation (Red) loss for the VGG11 model on training and validation sets respectively

Initial results showed that we are heavily overfitting to the training data. To counteract the overfitting we used the techniques from the literature such as increasing the proportion of dropped nodes and introducing data augmentation. We applied a random combination of vertical and horizontal flips to the volumes. These transformations are inexpensive and can be done when patches are fed into training via the network dataloader. This modifies the data such that it does not appear the same through each pass of the network. Randomly cropping the volumes was also considered however due to the form of the input data random crops can often yield entirely background volumes, removing the labelled feature resulting in large anomalies in classification.

We attempt to keep the architecture as simple as possible, choosing to only use the VGG network with 11 and 13 hidden layers as opposed to the larger 16 and 18 variants. One could argue that we should use the 'most accurate' network on the available. Accuracy, particularly in the medical image classification field is a fine balance between sensitivity and specificity. Many of these architectures were developed with natural image processing in mind, and as a result are not always the most appropriate to be applied to tasks rooted in medical data. We chose VGG due to it's record on image classification tasks. Due to the sparsity of data using a larger network runs a higher risk of out performing a simpler network purely on the basis of abstract overfitting of the training data.

When we applied the regularisation techniques to our model, we saw that decreasing the patch size, therefore increasing the number of items used for training, had no significant affect on the overall accuracy of the model. Fig.22 appears to reach higher accuracies faster than that of Fig.23 when we inspect

Fig.24 it suggests that the smaller patches could cause the model to overfit faster than the larger patch. The validation loss for the smaller patch only changes by a minor amount over training, while training loss continues to decrease. This suggests that the model is learning the training data, but is not improving its applicability on the validation set. A likely explanation for this is the fact that reducing the patch size does not introduce any new information for the model while the larger patches give more context to allow better distinction between patches containing error and those which do not. The minibatch size was 15 for both patch sizes, this means the smaller patch model updated its weights significantly more often than the larger patch model.

We observe that the larger patch size with the data augmentation Fig.25 does not overfit in the epoch range the experiment was carried over. A summary of the results are shown in Table.2

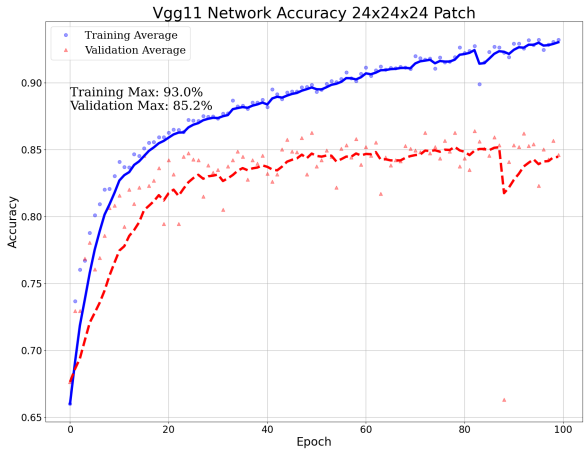


Figure 22: VGG11 Network Accuracy for the smaller patch size of 24x24x24 pixels. Training accuracies (Blue) and Validation accuracies (Red) are plotted. Maximum values are calculated from the exponential moving average

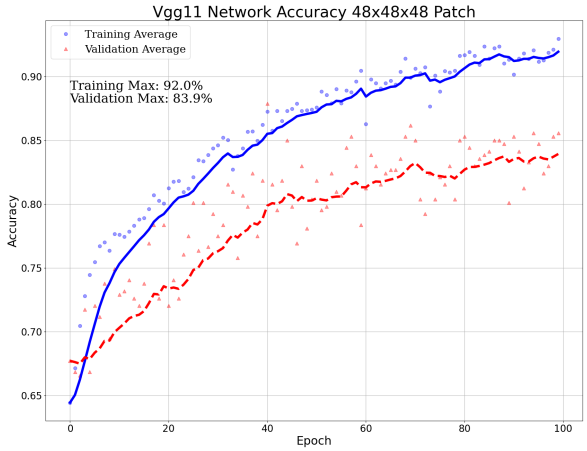


Figure 23: VGG11 Network Accuracy for the larger patch size of 48x48x48 pixels. Training accuracies (Blue) and Validation accuracies (Red) are plotted. Maximum values are calculated from the exponential moving average

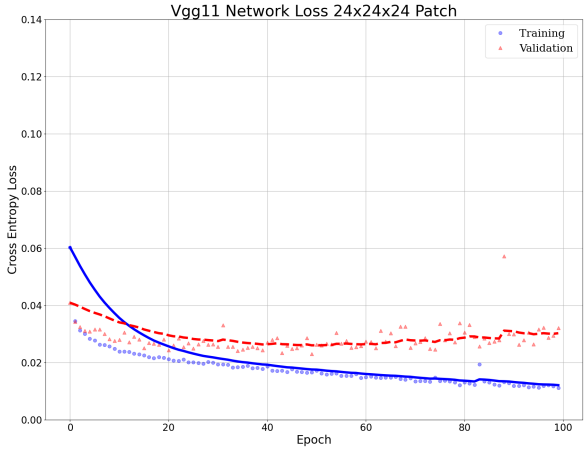


Figure 24: VGG11 Network Loss for the smaller patch size of 24x24x24 pixels. Training Loss (Blue) and Validation Loss (Red) are plotted. Fit is an exponential moving average

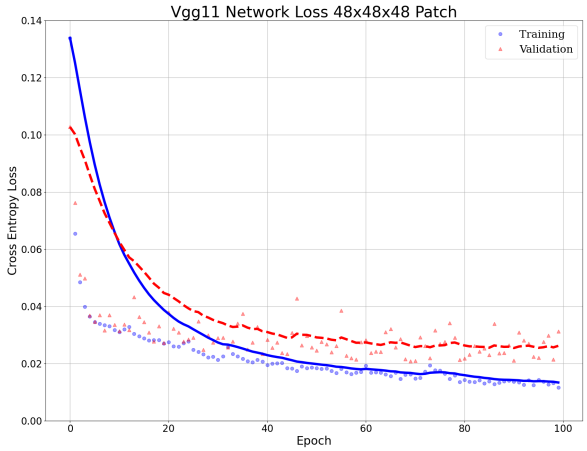


Figure 25: VGG11 Network Loss for the larger patch size of 48x48x48 pixels. Training Loss (Blue) and Validation Loss (Red) are plotted. Fit is an exponential moving average

Table 2: Summary of Patch Size Results

	24x24x24	48x48x48	Evidence of overfitting?
Maximum Training Accuracy	93.0%	92.0%	Yes
Maximum Validation Accuracy	85.2%	83.9%	No

To analyse the impact of the complexity of the network, we tested the VGG13 network with the 48x48x48 patch input. The accuracy of the model is shown in Fig.26. We notice that there is some minor overfitting in Fig.27 from around the 150th epoch. Training accuracy continues to steadily increase over the length of the experiment, yielding an accuracy maximum of 94.0% . This was expected as deeper networks tend to perform better on training sets but are more susceptible to overfitting. The validation accuracy maximum appears to be reached at the 100th epoch and remains steady for the remainder of the experiment, the validation accuracy for the deeper model of 87% is an improvement on the 83.9% score recorded in Fig.23 indicating that the deeper network is performing better without significant signs of overfitting.

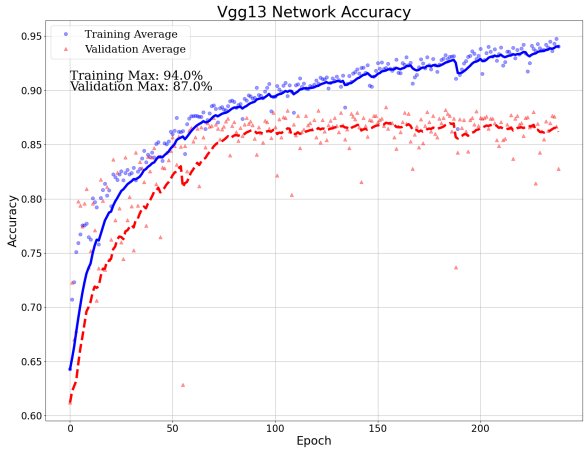


Figure 26: The VGG13 Network Architecture, Training Accuracy (Blue) and Validation Accuracy (Red) are plotted, maximum training accuracy is calculated from the exponential moving average fit.

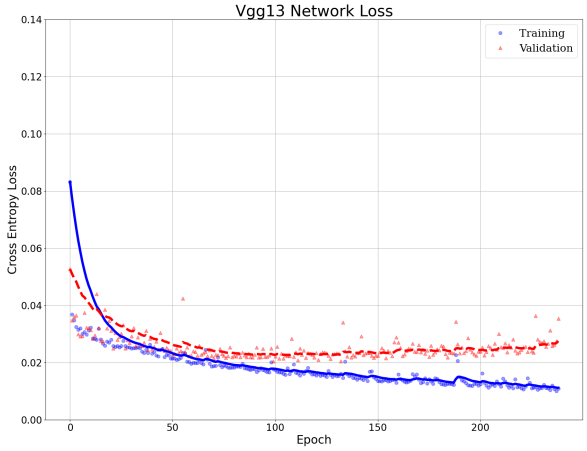


Figure 27: The VGG13 Network Architecture, Training Loss (Blue) and Validation Loss (Red) are plotted, trendlines are Exponential moving averages.

2.7 Tuning

Basic hyper-parameters were selected via manual search, this was done as there were very few impactful parameters we could adjust based on the size of the dataset. These were, batch size and learning rate, these were all decided to be 15 and 0.0001.

Parameters can also be found via random grid search and could have be done if we identified significant pitfalls in the parameters we have chosen [25].

We initially tuned using 40 training patients and 8 validation patients. To ensure the robustness of our model a 10 fold cross validation was used to fine tune the hyper-parameters. 10 Fold Cross Validation required splitting the entire shuffled training data set into 10 equally sized folds. One fold was left out as the test set, while the remaining 9 folds are used as a training set. As shown in Fig.28

The performance was recorded for each model, with every fold being used as the test set in turn. The overall score was an average of performance over all 10 folds. This could then be optimised by adjusting parameters such as regularisation or learning rates. The learning rate was tested at 0.1,0.01,0.001 and 0.0001 values, it was identified that larger learning rates often yielded models which the model did not learn and hence was kept low.

The learning rate is how large of a step the model takes in the direction of the gradient decent. Low learning rates have very long training times and may never converge on the global minima, large learning rates run the risk of completely stepping over the global minima and converging in a very shallow minima. Due to the time constraints of the project, only a single 10 fold cross validation was carried out using both architectures. The cross validation of the vgg11 network was carried out over half the number of epochs to provide a level of comparison of models in the short time available, due to the model being trained on a smaller GPU than originally planned.

Figure 28: Depiction of how a training set is divided and the results calculated in a 10 fold cross validation [26]

3 Results

Both cross validated architectures were used in generating final results for this study. The entire training dataset of 48 patients each had each organ's contour extracted and split into 48x48x48 pixel sized patches, these patches were subsequently shuffled and split into 10 folds, initally each fold contained 38 correct patches and 174 error patches. The number of error patches was divided by two, such that the dataset was somewhat balanced while maintaining a good amount of data. Results are summarised in Table.3

3.1 VGG11 Cross Validation

The results from the 10 fold cross validation using the VGG11 architecture are shown in Fig.29 and Fig.30. The model performed well with a training accuracy maximum of 92.1% and a validation accuracy maximum of 88.3%. The model appears to start over fitting towards the end of the range of iterations analysed and thus results were unlikely to improve. Data contains no outstanding anomalous results and is fairly well grouped and described by the Exponential Moving Average.

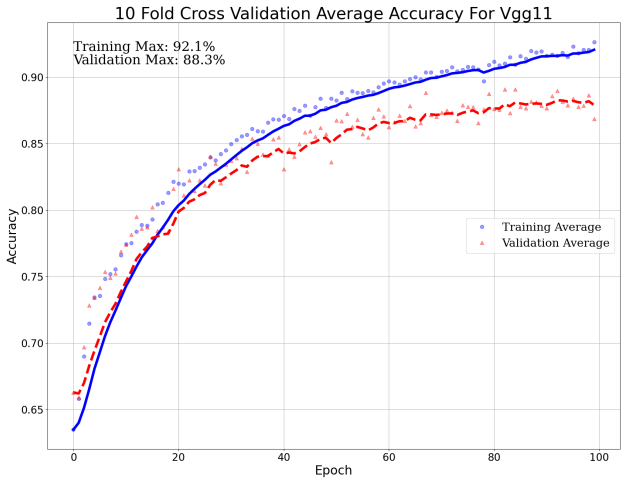


Figure 29: VGG11 10 Fold Cross Validation Accuracy. Training Accuracy (Blue) and Validation Accuracy (Red). Maximum accuracies are calculated from the exponential moving average (trendline)

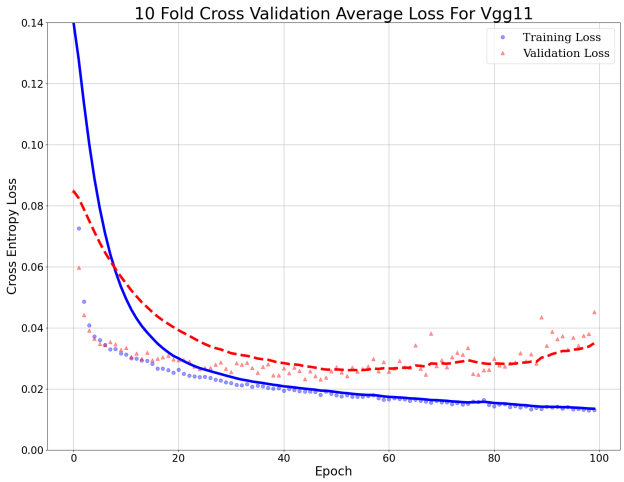


Figure 30: VGG11 10 Fold Cross Validation Loss. Training Loss (Blue) and Validation Loss (Red) are plotted, with an exponential moving average fitted

3.2 VGG13 Cross Validation

The VGG13 model's cross validation results are displayed in Fig.31 and Fig.32 the model reached higher accuracy than the VGG11 architecture. This is likely due to the increased number of epochs training was continued for. However the loss lower for the Vgg11 indicative of more confident predictions. The model appears to mildly overfit at around epoch 100 as the loss begins to diverge. Epoch 150 where the model appear to spike in validation loss while accuracy remains stable. This appears slightly anomalous and should be retested before drawing any significant conclusions. Loss resumes a downward trend as the model draws to the end of training, there is potential for the model to improve it's accuracy but validation remains somewhat stable for the last 75 epochs of training. Training yielded a maximum training accuracy of 94.0% and a maximum validation accuracy of 89.8%, both accuracies are described well by the exponential moving average, describing a stable model.

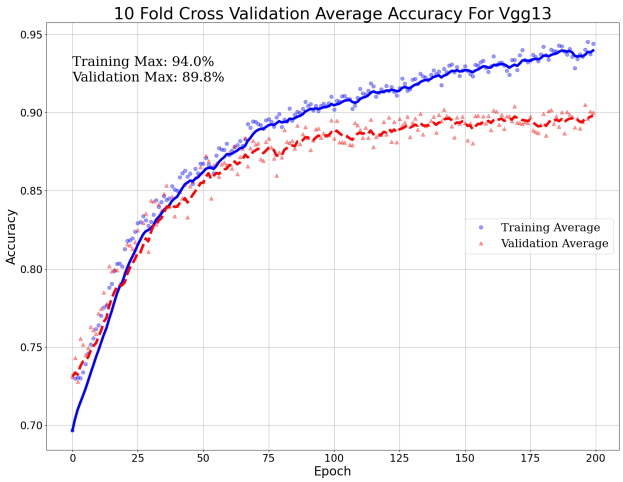


Figure 31: VGG13 10 Fold Cross Validation Accuracy. Training (Blue) and Validation (Red) maximum accuracies are calculated from the exponential moving average (trendline)

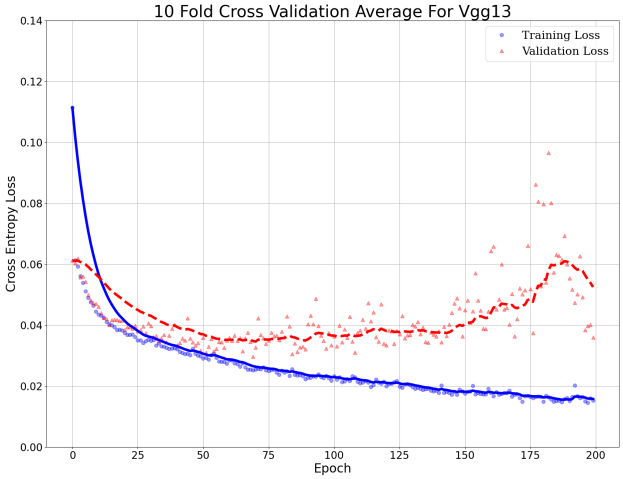


Figure 32: VGG13 10 Fold Cross Validation Loss. Training Loss (Blue) and Validation Loss (Red) are plotted, with an exponential moving average fitted

Table 3: Summary of Cross Validation Results

	Maximum Training Accuracy	Maximum Validation Accuracy
VGG11	92.1%	88.3%
VGG13	94.0%	89.8%

4 Discussion

4.1 Results and Significance

From the results in Table 3 We can see that there are strong grounds for a tool to be developed using deep learning, such as the one we describe. Producing a model which at worst has an error rate of 11.7% can be considered a success based on the quantity of data originally used to train the network.

The way we have analysed our model has primarily been the shape of the loss curves and the overall model accuracy. However we must appreciate the difference in correct and incorrect classification.

It is important to look not only at the accuracy, or maybe more appropriately the error rate, but at the sensitivity and specificity of the outputs.

There are actually four different outcomes to a classification problem such as this one. The case where there is an error and we classify as an error (True Negative) and where the contour is free of any errors and the model accurately predicts that (True Positive). These outcomes are both favourable and made up the accuracy score in model evaluation. In the medical industry, more potently with a tool aimed to improve patient safety. Consideration of the False Negative case, where there isn't an error and the model says there is, reduces the maximum efficiency our tool. But is not detrimental to the patients welfare. However if our algorithm were to classify an error patch as correct (False Positive), this type of classification could lead to some serious implications. If this research were to advance to any clinical stage any user would be fully aware of the error rate and self validation would still be advised. However it remains our aim to minimise these occurrences and we could do so by analysing and optimising our model for sensitivity and specificity.

Table 4: Table outlining the different forms of accuracy

	Predicted Error	Predicted Correct
True Error	True Negative (TN)	False Positive (FP)
True Correct	False Negative (FN)	True Positive (TP)

$$Sensitivity = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \tag{5}$$

and specificity is defined as, when the label is correct and it classifies it as correct

$$Specificity = \frac{True\ Negatives}{(True\ Negatives + False\ Positives)} \tag{6}$$

To summarise the above sensitivity is the actual proportion of positive cases which were correctly labelled positive and specificity is the proportion of negative cases which were correctly labelled negative.

To fully evaluate model performance evaluation metrics, such as a confusion matrix or a reciever operating characteristic (ROC) curve should be implemented. The confusion matrix would allow us to record the number of TN, FP, FN, TP errors and display them in a matrix, identifying whether there is a bias towards a particular type of classification.

The ROC curve, or more specifically the area under the curve will give us a numerical metric as to the performance of the model, with a larger area meaning the model is performing well.

A rudimentary method of evaluating the number of TN, FP, FN, TP of our current network would be to feed in patches with known labels, from our training set and manually count the occurrences of each type of classification.

To further optimise our network we could implement an attention mechanism, which essentially focuses, or draws attention to a specific aspects of the input. Allowing training to optimise based on reducing the number of false positives or false negatives specifically. Commonly attention mechanisms are used in text analysis to identify key words in sentences to generate responses an example of which is demonstrated in Bachrach et al 2017 [27]. With such a mechanism we could tune the model to severely punish false positives, as it is the most detrimental outcome.

The spike in Fig.32 was unexpected as it does not replicate behaviour seen in any other of the experiments. It is unclear what caused such a spike and should be retested.

Interestingly accuracy of the model appears relatively unaffected while the loss significantly increases. Accuracy is evaluated by cross evaluating the highest softmax output with the training label. Crucially it does not depend on the value of the output, just which is largest, this allows for a situation where the classifier maintains a correct classification, but with less confidence. This is to say an image may be classified as containing an error with a corresponding softmax value of 0.9 or 90% confidence contributes the same amount to the accuracy metric as the same image being classified with a softmax of 0.51. While the latter has a significantly higher loss. The peak may be due to a significant number of these type of events occurring during that training epoch. Overfitting may be the cause of such as spike, but further experimentation is required.

Based on the results of this study I believe it is feasible that, with a small tweaks and improvements based on things learned from carrying out this research, a clinically available prototype can be developed.

The results presented in this paper are by no means comprehensive and require deeper analysis and experimentation to progress research. We have succeeded in analysing the affect of using different sized patches and can conclude

that it is important to maintain some context within the patches to help the model avoid overfitting. Regularisation techniques, such as dropout, data augmentation and cross validation are all encourage in all further research in this area and should be expanded where possible.

Shortcomings of this research project are, the cross validation models only being ran once each. Due to time constraints the VGG11 architecture was ran for half as many epochs as the VGG13 variant. Without equal control measures it makes it difficult to draw quantitative conclusions from results. Furthermore results could be more deeply analysed in order to fully evaluate which architectural hyperparameters are most optimal for a 3D image classification task such as this one.

4.2 Future Areas of Study and Research

Differences in training and real world data: Due to the necessity of requiring a large volume of data to train a CNN, the dataset we are training and validating against is heavily skewed in favour of error detection. We could try to balance the dataset, at the cost to the amount of data we have available to train upon, the effect of this is an area for further experimentation.

In reality, some artefacts are more prevalent. Accidental brush strokes and missing slices missing slices have far more common occurrences in the clinical setting. Our training dataset is not is not fully representative of the datasets seen in the clinical environment, as such it would be important to robustly test it on real clinical cases. Data with significantly fewer errors and without data synthesis and carefully look at how the model performs.

Should more data be available, the data pre-processing and training inputs could be evolved such that structures are retained together. Assisting in localisation and allowing for classifications on particular structures. Currently we have no understanding of whether we can successfully classify errors better on the heart or the lungs. Due to anatomy, we simply have more data patches originating from the lungs than the heart. Structures like the spinal column are simpler to detect abnormalities due to their geometry.

It may be deemed appropriate to adjust patch size based on the organ to refine classification in the future. Should structural retention be achieved, classification on a structural level is something Lead Dosimetrist at University College Hospitals has suggested. The idea consisting of a traffic light approach, whereby each delineated structure was listed and a traffic light assigned based on the model's classification.

Computationally such an approach is straightforward to implement. The final layer output gives the probability of classification, such that we could assign a threshold to each of the traffic light colours. Setting such thresholds raises an interesting question of what level of 'accuracy' is appropriate and what do we mean by accuracy in this context. The output of our current model is a binary decision of either 1 or 0. These values are derived from the most likely probability output by the softmax. By accessing the probabilities directly from the softmax output allows us to see the models 'confidence' in its prediction. One might argue

setting the amber warning for anything between 70-95% based on current model performance, with the implementation of further optimisation and performance metrics this could be re-evaluated.

The model currently evaluates on a patch by patch basis, such that no contextual information about neighbouring patches or underlying CT data is passed into the network. This is particularly significant in the evaluation of the abnormal geometries artefact, as without the underlying CT data what constitutes abnormal is not defined. Further development of the algorithm to include the underlying CT would provide the clinician with a reference to make evaluations of these types of artefact

Increasing Number of Classification Classes: By increasing the number of classification classes, one for each type of artefact studied, we could offer more help to clinicians in their evaluation of whether the artefact is clinically significant and worth checking under the current system, ideally we envisage the system to make the process easier, such that every artefact is corrected. However the time constraints clinicians work under often requires them to make these types of decisions.

The implementation would likely require more data. By splitting the current classes of correct and error into four, it could open the option of slightly weighting the dataset towards the errors which occur more often, but must ensure that the training set is reasonably balanced and representative of the population.

4.3 Difficulties Faced

Manually Checking Data The process of extracting the contours would have to be refined if this project were to develop further. Current methodology is inappropriate for large scale contour extraction. This could be remedied by having a method whereby a single contour, or a set list of ordered contours are input at any one time.

The problem currently arises from the requirement of outputting a file which is labelled with the structure it contains. Remove this requirement would allow the process to be significantly more automated. Having a set ordered list would allow us to simply iterate through them, while labelling based on their index in the list as a possible work around.

Delay in Data Due to the requirement for additional contouring artefacts to be synthesised by clinicians. Data took two months longer than expected to be suitably de-identified and passed over from UCH. This unforeseen setback has resulted in some of the original aims of the project not being met. Most notably the localisation aspect of the original brief. The patch based approach we took to lends itself well to this implementation, as we would have attempted a flag to an entire patch resulting in a sub region to be checked. This is still very much possible, and even more feasible than initially thought based on our research, implementation would require the original location of the patch to be tracked

and indexed then a user interface would display the input volume with each patch filled in red or green as to whether the algorithm predicted there to be an error or not.

5 Conclusion

In this research we have successfully trained a Deep Convolutional Neural Network Classifier based on two forms of the VGG network architecture which yield cross validation results of 88.3% and 89.8% with the VGG11 and VGG13 architectures. We have inspected an analysed the effect for varying the patch size for two evaluation metrics, accuracy and loss and concluded that using larger patches is more beneficial for model training.

This is a positive start for the development of a clinical support tool,filling the gap in quantitative evaluation tools in radiotherapy delineations. Increasing the efficiency and accuracy of contour artefact detection, freeing up valuable time for clinicians while maintaining patient safety .

We saw how data regularisation techniques could be applied to our data in order to increase generalisation on unseen datasets and minimise overfitting our model. Subsequently utilising a 10 fold cross validation to evaluate the model based on the entire dataset, opposed to a subset as was initially done

We provide a proof of concept that applying a deep CNN network to aid in detection of artefacts generated in the process of delineating tumours and OARs in radiotherapy treatment plans. Direction for future research as to how such a tool can be developed and improved based on current technologies and research, such as structural based classification, patch based localisation and introducing alternative evaluation metrics. The shortcomings of our research include the minimal quantity of data and its impact on suitable blind testing, additionally the use of surface level metrics to evaluate the model and the inadequacies of doing so to appropriately describe the models success in the medical field. Deeper evaluation metrics should be implemented to provide rigorous testing evidence before this tool can be used in a clinical setting.

6 Acknowledgements

Thank you to UCH for providing the data used to carry out this research.

Thank you to Ed Chandy for synthesising the errors on our training data. Special thanks to Pravesh Bhudia for inspiring the project and providing clinical insight.

Thank you to Adam Szmul and Jamie McClelland for supervising this project and providing your guidance and support throughout.

References

1. L. Santanam, R. S. Brame, A. Lindsey, T. Dewees, J. Danieley, J. Labrash, P. Parikh, J. Bradley, I. Zoberi, J. Michalski *et al.*, "Eliminating inconsistencies

in simulation and treatment planning orders in radiation therapy," *International Journal of Radiation Oncology* Biology* Physics*, vol. 85, no. 2, pp. 484–491, 2013.
2. J. Stanley, P. Dunscombe, H. Lau, P. Burns, G. Lim, H.-W. Liu, R. Nordal, Y. Starreveld, B. Valev, J.-P. Voroney *et al.*, "The effect of contouring variability on dosimetric parameters for brain metastases treated with stereotactic radiosurgery," *International Journal of Radiation Oncology* Biology* Physics*, vol. 87, no. 5, pp. 924–931, 2013.
3. N. G. Burnet, S. J. Thomas, K. E. Burton, and S. J. Jefferies, "Defining the tumour and target volumes for radiotherapy," *Cancer Imaging*, vol. 4, no. 2, p. 153, 2004.
4. H.-C. Chen, J. Tan, S. Dolly, J. Kavanaugh, M. A. Anastasio, D. A. Low, H. Harold Li, M. Altman, H. Gay, W. L. Thorstad *et al.*, "Automated contouring error detection based on supervised geometric attribute distribution models for radiation therapy: A general strategy," *Medical physics*, vol. 42, no. 2, pp. 1048–1059, 2015.
5.
6. C. C. Chatterjee, "Basics of the classic cnn," Jul 2019. [Online]. Available: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
7. Datahacker.rs, "002 cnn edge detection," Oct 2019. [Online]. Available: http://datahacker.rs/edge-detection/
8. A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
9. A. Sharma, "Understanding activation functions, avaliable at https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/," Oct 2017.
10. R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.
11. M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 111–147, 1974. [Online]. Available: http://www.jstor.org/stable/2984809
12. M. NarasingaRao, V. V. Prasad, P. S. Teja, M. Zindavali, and O. P. Reddy, "A survey on prevention of overfitting in convolution neural networks using machine learning techniques," *International Journal of Engineering and Technology (UAE)*, vol. 7, no. 2.32, pp. 177–180, 2018.
13. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
14. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
15. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
16. O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a
17. B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," *arXiv preprint arXiv:1706.01307*, 2017.
18. B. Graham, M. Engelcke, and L. van der Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," *CVPR*, 2018.

19. D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.
20. A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
21. "Imagenet." [Online]. Available: http://www.image-net.org/
22. M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.
23. R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
24. J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.
25. J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.
26. K. Rosaen, "k-fold cross-validation, hyperparameter tuning, avaliable at http://karlrosaen.com/ml/learning-log/2016-06-20/," Jun 2016.
27. Y. Bachrach, A. Zukov-Gregoric, S. Coope, E. Tovell, B. Maksak, J. Rodriguez, and C. McMurtie, "An attention mechanism for answer selection using a combined global and local view," *arXiv preprint arXiv:1707.01378*, 2017.

A Extracting Contours

```
import os, sys, glob
import numpy as np
import pydicom
import nibabel as nib
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.pyplot as plt
# matplotlib inline
from skimage.draw import polygon
import dicom2nifti
from scipy import ndimage


def read_structure(structure, structure_name):
    contours = []
    for i in range(len(structure.ROIContourSequence)):
        if structure.StructureSetROISequence[i].ROIName ==
    structure_name:
            contour = {}
            contour['color'] = structure.ROIContourSequence[i
    ].ROIDisplayColor
            # contour['number'] = structure.ROIContourSequence
    [i].RefdROINumber
            contour['number'] = 1
```

```
            contour['name'] = structure.
    StructureSetROISequence[i].ROIName
            # assert contour['number'] == structure.
    StructureSetROISequence[i].ROINumber
            assert contour['number'] == 1
            contour['contours'] = [s.ContourData for s in
    structure.ROIContourSequence[i].ContourSequence]
            contours.append(contour)
    return contours


def get_mask(contours, slices):
    z = [s.ImagePositionPatient[2] for s in slices]
    pos_r = slices[0].ImagePositionPatient[1]
    spacing_r = slices[0].PixelSpacing[1]
    pos_c = slices[0].ImagePositionPatient[0]
    spacing_c = slices[0].PixelSpacing[0]

    label = np.zeros_like(image, dtype=np.uint8)
    for con in contours:
        num = int(con['number'])
        for c in con['contours']:
            nodes = np.array(c).reshape((-1, 3))
            assert np.amax(np.abs(np.diff(nodes[:, 2]))) == 0

            zNew = [round(elem, 1) for elem in z]
            try:
                z_index = z.index(nodes[0, 2])
            except ValueError:
                z_index = zNew.index(nodes[0, 2])

            r = (nodes[:, 1] - pos_r) / spacing_r
            c = (nodes[:, 0] - pos_c) / spacing_c
            rr, cc = polygon(r, c)
            rr[rr > (label.shape[0] - 1)] = label.shape[0] - 1
            cc[cc > (label.shape[1] - 1)] = label.shape[1] - 1

            if z_index < label.shape[2]:
                label[rr, cc, z_index] = num

    colors = tuple(np.array([con['color'] for con in contours
    ]) / 255.0)
    return label, colors

structure_names = ['Spinal CanalEr','Spinal CanalCor','
    SpinalCanalEr','Spinal Canal', 'SpinalCanal', 'SpinalCorEr
    ', 'CANALCor', 'CANALEr','Heart', 'HeartEr',
                   'HEART', 'HEARTEr', 'LUNGS','LUNGSCor', '
    LUNGSEr','LungsEr', 'L Lung', 'L LungEr', 'Lt Lung', '
    LtLung', 'Lung_L', 'Lung_LEr',
```

```
                    'LLungEr', 'RLung', 'R Lung', 'Rt Lung', '
    R_Lung', 'Lung_R', 'RightLungCor', 'R_LungEr', 'Lung_REr',
                    'RightLungEr', 'RTLungEr', 'RLungEr', 'Rt
    LungEr', 'OesophagusCor','Oesophagus', 'OesophagusCor',
                    'CanalPRVCor','CanalPRVEr','Canal PRVCor',
    'Canal PRVEr', 'Spinal CanalPRVEr', 'CANAL PRV', 'CANAL
    PRVEr',
                    'Canal + 5mm', 'Canal + 5mmEr', 'Spinal
    canal PRV', 'canalprvER', 'Canal PRV']

structure_out_names = ['SpinalCanalEr','SpinalCanalCor', '
    SpinalCanalEr1', 'SpinalCanalCor1', 'SpinalCanalCor2', '
    SpinalCanalEr2', 'SpinalCanalCor3', 'SpinalCanalEr3',
                    'HeartCor', 'HeartEr','HeartCor1', '
    HeartEr1', 'LungsCor','LungsCor1', 'LungsEr','LungsEr1', '
    LLungCor', 'LLungEr', 'LLungCor1', 'LLungCor2', 'LLungCor3
    ',
                    'LLungEr1', 'LLungEr2', 'RLungCor', '
    RLungCor1', 'RLungCor2', 'RLungCor3', 'RLungCor4', '
    RLungCor5', 'RLungEr', 'RLungEr1',
                    'RLungEr2', 'RLungEr3', 'RLungEr4', '
    RLungEr5', 'OesophagusCor','OesophagusCor1', 'OesophagusEr
    ', 'CanalPRVCor', 'CanalPRVEr',
                    'CanalPRVCor1', 'CanalPRVEr1', '
    CanalPRVEr2', 'CanalPRVCor2', 'CanalPRVEr3', 'CanalPRVCor4
    ', 'CanalPRVEr5',
                    'CanalPRVCor6', 'CanalPRVEr7', '
    CanalPRVCor8']

train_data_path = "/Volumes/My Passport for Mac/Project/Dicom/
    "
nifti_out_dir = '/Volumes/My Passport for Mac/Project/NewNift/
    '
if not os.path.isdir(nifti_out_dir):
    os.makedirs(nifti_out_dir)

train_patients = [os.path.join(train_data_path, name)
                  for name in os.listdir(train_data_path) if
    os.path.isdir(os.path.join(train_data_path, name))]

while '/Volumes/My Passport for Mac/Project/NewNift/' in
    train_patients: train_patients.remove(
    '/Volumes/My Passport for Mac/Project/NewNift/')


# for i in range(48, len(train_patients)):
for i in range(42, len(train_patients)):
    patient = train_patients[i]  # Just get the first patient
    for demo
    print(patient)
```

```
    print(i)

    if i < 11:
        output_ct_dir = nifti_out_dir + 'UCL_LUNG_0' + str(i+1)
        if not os.path.isdir(output_ct_dir):
            os.makedirs(output_ct_dir)
        output_ct_file = output_ct_dir + '/CT_UCL_LUNG_0' + str(i+1) + '.nii'

    elif i > 10:
        output_ct_dir = nifti_out_dir + 'UCL_LUNG_' + str(i+1)
        if not os.path.isdir(output_ct_dir):
            os.makedirs(output_ct_dir)
        output_ct_file = output_ct_dir + '/CT_UCL_LUNG_' + str(i+1) + '.nii'



    temp_dir = patient + '/'
    dcms_org = glob.glob(os.path.join(temp_dir, "*.dcm"))

    dcms = dcms_org[0:-2]  # RT structures removed from the list of DCM files
    if len(dcms) > 0:

        if len(dcms) == 1:
            structure = pydicom.dcmread(os.path.join(temp_dir, dcms[0]))
            contours = read_structure(structure)
        elif len(dcms) > 1:
            dicom2nifti.dicom_series_to_nifti(temp_dir, output_ct_file, reorient_nifti=True)
            slices = [pydicom.dcmread(dcm) for dcm in dcms]
            slices.sort(key=lambda x: float(x.ImagePositionPatient[2]))
            image = np.stack([s.pixel_array for s in slices], axis=-1)

            structure = pydicom.dcmread(dcms_org[-1])

            for structure_number in range(0, len(structure_names)):
                print(structure_names[structure_number])

                contours = read_structure(structure, structure_names[structure_number])
                label_im, colors = get_mask(contours, slices)
                label_im = np.swapaxes(label_im, 0, 1)
                label_im = np.flip(label_im, 1)
```

```
                ct_im_nii = nib.load(output_ct_file)
                label_im_nii = nib.Nifti1Image(label_im,
    ct_im_nii.affine, ct_im_nii.header)

                if i < 11:
                    output_ct_dir = nifti_out_dir + '
    UCL_LUNG_0' + str(i + 1)
                    label_im_path = output_ct_dir + '/' +
    structure_out_names[structure_number] + '_0' + str(i + 1)
    + '.nii'

                elif i > 10:
                    output_ct_dir = nifti_out_dir + 'UCL_LUNG_
    ' + str(i + 1)
                    label_im_path = output_ct_dir + '/' +
    structure_out_names[structure_number] + '_' + str(i + 1) +
     '.nii'

                nib.save(label_im_nii, label_im_path)
```

extract_seg_errors_rt.py

B Removing Empty Files

```
import os
import nibabel as nib
import numpy as np
patient_num = []
sourcedir = '/Volumes/My Passport for Mac/Project/NewNift/
    UCL_LUNG_'
for x in range(51):
    patient_num.append(str(x).zfill(2))
patient_num.remove('00')


structure_out_names = ['Spinal CanalEr','Spinal CanalCor', '
    SpinalCanalEr1', 'SpinalCanalCor1', 'SpinalCanalCor2', '
    SpinalCanalEr2', 'SpinalCanalCor3', 'SpinalCanalEr3',
                       'HeartCor', 'HeartEr','HeartCor1', '
    HeartEr1', 'LungsCor','LungsCor1', 'LungsEr','LungsEr1', '
    LLungCor', 'LLungEr', 'LLungCor1', 'LLungCor2', 'LLungCor3
    ',
                       'LLungEr1', 'LLungEr2', 'RLungCor', '
    RLungCor1', 'RLungCor2', 'RLungCor3', 'RLungCor4', '
    RLungCor5', 'RLungEr', 'RLungEr1',
                       'RLungEr2', 'RLungEr3', 'RLungEr4', '
    RLungEr5', 'OesophagusCor','OesophagusCor1', 'OesophagusEr
    ', 'CanalPRVCor', 'CanalPRVEr',
```

```
                        'CanalPRVCor1', 'CanalPRVEr1', '
    CanalPRVEr2', 'CanalPRVCor2', 'CanalPRVEr3', 'CanalPRVCor4
    ', 'CanalPRVEr5',
                        'CanalPRVCor6', 'CanalPRVEr7', '
    CanalPRVCor8']

for i in range(len(patient_num)):
    print(patient_num[i])

    for name in structure_out_names:
        if i < 9:
            structure_file_name = sourcedir + patient_num[i] +
     '/' + name + '_' + str(patient_num[i]) + '.nii'
        elif i > 9:
            structure_file_name = sourcedir + patient_num[i] +
     '/' + name + '_' + str(patient_num[i]) + '.nii'

        if os.path.exists(structure_file_name):

            load_struct = nib.load(structure_file_name)
            structure_vol = np.asarray(load_struct.get_fdata()
    )
            if np.count_nonzero(structure_vol) > 0:
                print(structure_file_name, 'is correct')

            else:
                os.remove(structure_file_name)
```

Remove Empty.py

C Extracting Patches

```
import numpy as np
import nibabel as nib
import os

organs = ['CanalPRV', 'Heart', 'LLung', 'RLung', 'Oesophagus',
     'SpinalCanal', 'Lungs', 'Spinal Canal']
patient_num = []
for x in range(50):
    patient_num.append(str(x).zfill(2))
patient_num.remove('00')


def compare_patches(cor_filepath, er_filepath, dif_filepath, x
    , y, z,cor_path,er_path):

```

```
    image_dif = nib.load(dif_filepath)
    image_er = nib.load(er_filepath)
    image_cor = nib.load(cor_filepath)

    imgdata_dif = np.asarray(image_dif.get_fdata())
    imgdata_er = np.asarray(image_er.get_fdata())
    imgdata_cor = np.asarray(image_cor.get_fdata())

    img_dif_datapad = np.pad(imgdata_dif, ((528 - imgdata_dif.shape[0], 0), (528 - imgdata_dif.shape[1], 0),
                                           (528 - imgdata_dif.shape[2], 0)), constant_values=0)
    img_er_datapad = np.pad(imgdata_er, ((528 - imgdata_er.shape[0], 0), (528 - imgdata_er.shape[1], 0),
                                         (528 - imgdata_er.shape[2], 0)), constant_values=0)
    img_cor_datapad = np.pad(imgdata_cor, ((528 - imgdata_cor.shape[0], 0), (528 - imgdata_cor.shape[1], 0),
                                           (528 - imgdata_cor.shape[2], 0)), constant_values=0)
    lenx = int(528 / x)
    leny = int(528 / y)
    lenz = int(528 / z)

    er_array = np.zeros((1, x, y, z))
    cor_array = np.zeros((1, x, y, z))

    er_num = 0
    cor_num = 0


    cor_path = str(cor_path)
    er_path = str(er_path)
    for i in range(0, lenx):
        for j in range(0, leny):
            for k in range(0, lenz):
                temp_patch_er = img_er_datapad[i * x:(i + 1) * x, j * y:(j + 1) * y,
                                k * z:(k + 1) * z]
                # temp_patch_dif = img_dif_datapad[i * x:(i + 1) * x, j * y:(j + 1) * y,
                #                  k * z:(k + 1) * z]
                temp_patch_cor = img_cor_datapad[i * x:(i + 1) * x, j * y:(j + 1) * y,
                                 k * z:(k + 1) * z]

                if np.count_nonzero(temp_patch_er - temp_patch_cor):
```

```
52                      # er_array = np.vstack((er_array, np.
    expand_dims(temp_patch_er, axis=0)))
                        er_nii = nib.Nifti1Image(temp_patch_er,
    image_er.affine, image_er.header)
54                      #nib.save(er_nii, er_path)
                        np.save(er_path + str(er_num),
    temp_patch_er)
56                      er_num += 1

58                  else:
                        if np.count_nonzero(temp_patch_cor):
60

62                              #cor_array = np.vstack((cor_array,
     np.expand_dims(temp_patch_cor, axis=0)))
                            #cor_nii = nib.Nifti1Image(
    temp_patch_cor, image_cor.affine, image_cor.header)
64                          #nib.save(cor_nii, cor_path)

66                          np.save(cor_path + str(cor_num),
    temp_patch_cor)
                            cor_num += 1
68


70


72      return er_array, cor_array

74


76
#
78
# def generate_patches(img,x,y,z):
80 #     img = nib.load(img)
#      imgdata = np.asarray(img.get_fdata())
82 #      padimage = np.pad(imgdata, ((528 - imgdata.shape[0], 0),
     (528 - imgdata.shape[1], 0),
#                                                 (240 - imgdata.
    shape[2], 0)), constant_values=0)
84 #      lenx = int(528 / x)
#      leny = int(528 / y)
86 #      lenz = int(240 / z)
#
88 #      out_array = np.zeros((1,x,y,z))
#
90 #      for i in range(0, lenx):
#          for j in range(0, leny):
92 #              for k in range(0, lenz):
```

```
#                   temp_patch = padimage[i * x:(i + 1) * x, j *
    y:(j + 1) * y,
#                                         k * z:(k + 1) * z]
#                   # if np.count_nonzero(temp_patch):
#                   out_array = np.vstack((out_array,np.
    expand_dims(temp_patch, axis= 0)))
#
#     return out_array

#generate_patches(er_filepath,24,24,24)


for number in range(0,50):
    sourcedir = 'NewNift/UCL_LUNG_' + patient_num[number]
    dif_dir = 'NewNift/Differences/' + patient_num[number]


    for x in organs:
        if number <= 40:
            destdir_cor = 'NewNift/Patches/train24/correct/' +
    str(number) +'_'
            destdir_er = 'ewNift/Patches/train24/error/' + str
    (number) +'_'
            if not os.path.isdir(destdir_er):
                os.makedirs(destdir_er)
            if not os.path.isdir(destdir_cor):
                os.makedirs(destdir_cor)
        if number > 40:
            destdir_cor = 'NewNift/CV/valid24/correct/' + str(
    number)+ '_'
            destdir_er = 'NewNift/CV/valid24/error/' + str(
    number)+ '_'
            if not os.path.isdir(destdir_er):
                os.makedirs(destdir_er)
            if not os.path.isdir(destdir_cor):
                os.makedirs(destdir_cor)
        cor_filepath = sourcedir + '/' + x + 'Cor_' +
    patient_num[number] + '.nii'
        er_filepath = sourcedir + '/' + x + 'Er_' +
    patient_num[number] + '.nii'
        dif_filepath = dif_dir + '/' + x + '_Diff_' +
    patient_num[number] + '.nii'
        if os.path.isfile(cor_filepath) and os.path.isfile(
    er_filepath) and os.path.isfile(dif_filepath):

            y = compare_patches(cor_filepath, er_filepath,
    dif_filepath, 24, 24, 24, destdir_cor, destdir_er)
            print(len(y[0]))
```

Extracting Patches.py

D Models

```
import torch
import torch.nn as nn



__all__ = [
    'VGG', 'vgg11', 'vgg11_bn'
]



class VGG(nn.Module):

    def __init__(self, features, num_classes=2, init_weights=True):
        super(VGG, self).__init__()
        self.features = features
        self.avgpool = nn.AdaptiveAvgPool3d((7,7,7))
        self.classifier = nn.Sequential(
            nn.Linear(43904, 4096),
            nn.ReLU(True),
            nn.Dropout3d(0.6),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout3d(),
            nn.Linear(4096, num_classes),
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        print(x.shape)
        x = self.features(x)
        x = self.avgpool(x)
        #print(x.shape)
        #x = x.view(x.size(0), -1)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        #print(x.shape)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv3d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
```

```
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm3d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)


def make_layers(cfg, batch_norm=False):
    layers = []
    in_channels = 1
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool3d(kernel_size=2, stride=2)]
        else:
            conv3d = nn.Conv3d(in_channels, v, kernel_size=3,
    padding=1)
            if batch_norm:
                layers += [conv3d, nn.BatchNorm3d(v), nn.ReLU(
    inplace=True)]
            else:
                layers += [conv3d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)


cfgs = {
    'A': [32, 'M', 64, 'M', 64, 'M', 128, 'M'],
    'B': [32, 32, 'M', 64, 64, 'M', 64, 64, 'M', 128, 128, 'M'
    , 128, 128, 'M'],
}


def _vgg(arch, cfg, batch_norm, pretrained, progress, **kwargs
    ):
    if pretrained:
        kwargs['init_weights'] = False
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm),
     **kwargs)
    return model


def vgg11(pretrained=False, progress=True, **kwargs):
    r"""VGG 11-layer model (configuration "A") from
    `"Very Deep Convolutional Networks For Large-Scale Image
    Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_
    Args:
        pretrained (bool): If True, returns a model pre-
    trained on ImageNet
```

```
88          progress (bool): If True, displays a progress bar of
        the download to stderr
            """
90      return _vgg('vgg11', 'A', False, pretrained, progress, **
        kwargs)

92  def vgg11_bn(pretrained=False, progress=True, **kwargs):
        r"""VGG 11-layer model (configuration "A") with batch
        normalization
94      `"Very Deep Convolutional Networks For Large-Scale Image
        Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_
        Args:
96          pretrained (bool): If True, returns a model pre-
        trained on ImageNet
            progress (bool): If True, displays a progress bar of
        the download to stderr
98      """
        return _vgg('vgg11_bn', 'A', True, pretrained, progress,
        **kwargs)
100
    def vgg13(pretrained=False, progress=True, **kwargs):
102     r"""VGG 13-layer model (configuration "B")
        `"Very Deep Convolutional Networks For Large-Scale Image
        Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_
104     Args:
            pretrained (bool): If True, returns a model pre-
        trained on ImageNet
106         progress (bool): If True, displays a progress bar of
        the download to stderr
        """
108     return _vgg('vgg13', 'B', False, pretrained, progress, **
        kwargs)


110
    def vgg13_bn(pretrained=False, progress=True, **kwargs):
112     r"""VGG 13-layer model (configuration "B") with batch
        normalization
        `"Very Deep Convolutional Networks For Large-Scale Image
        Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_
114     Args:
            pretrained (bool): If True, returns a model pre-
        trained on ImageNet
116         progress (bool): If True, displays a progress bar of
        the download to stderr
        """
118     return _vgg('vgg13_bn', 'B', True, pretrained, progress,
        **kwargs)
```

models.py

E Dataloader

```
from __future__ import print_function, division
import os
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms,utils, datasets
import nibabel as nib
import torchvision.transforms as tf
from PIL import Image
import random
from scipy.ndimage import zoom
# def data_loader(file_path):
#     structure_nii = nib.load(file_path)
#     image_array = np.asarray(structure_nii.get_fdata())
#
#     return image_array
# def load_data(root = '/Volumes/My Passport for Mac/Project/
    NewNift/Patches/train/error/'):
#     data_transform = transforms.Compose([transforms.ToTensor
    ',
#                                          transforms.
    RandomHorizontalFlip])
#     RT_Dataset = datasets.DatasetFolder(root= root, loader=
    data_loader,
#                                         transform=
    data_transform, extensions=('.nii'))
#     dataset_loader = torch.utils.data.DataLoader(RT_Dataset,
     batch_size= 5)
#
#     return dataset_loader

# train_loader = load_data()
#
# for batchidx, img_label in enumerate(train_loader):
#     print(batchidx,img_label)

# def segmentation_Dataset(Dataset):
#     def __init__(self, nii_file):
#         self.nii_patch = nii_file
#         return
#     def __len__(self):
#         return len(self.nii_patch)
#     def __getitem__(self,idx):
#         if torch.is_tensor(idx):
```

```
#             idx = idx.tolist()
#         image =
#
#         return


def npy_loader(path):
    sample = np.load(path)
    sample = torch.Tensor(sample)
        #Random Horizontal Flip
    if random.random() > 0.5:
        sample = torch.flip(sample,[0,2])

        #Random vertical flipping
    if random.random() > 0.5:
        sample = torch.flip(sample,[1,2])
    return sample


# dataset = datasets.DatasetFolder(
#     root='NewNift/Patches/train2',
#     loader=npy_loader,
#     extensions='.npy'
# )
```

Dataloader.py

F Training

```
import os
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torchvision import datasets
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from Models import vgg11_bn, vgg11, vgg13, vgg13_bn
from DataLoader import npy_loader
from torch.utils.tensorboard import SummaryWriter
import torchvision.transforms as tf


def get_num_correct(preds, labels):
    return preds.argmax(dim=1).eq(labels).sum().item()

```

```
22 # def accuracy(predictions,labels):
   #     '''
24 #     Accuracy of a given set of predictions of size (N x
       n_classes) and
   #     labels of size (N x n_classes)
26 #     '''
   #     return np.sum(np.argmax(predictions,axis=1)==np.argmax(
       labels,axis=1))*100.0/labels.shape[0]
28
   # transforms = tf.Compose([
30 #     transforms.RandomHorizontalFlip(),
   #     transforms.RandomVerticalFlip(),
32 #     transforms.ToTensor(),
   #])
34 dataset = datasets.DatasetFolder(
       root='NewNift/Patches/train48/',
36     loader=npy_loader,
       extensions='.npy',
38     #transform= transforms
   )
40 validset = datasets.DatasetFolder(
       root='NewNift/Patches/valid48/',
42     loader=npy_loader,
       extensions='.npy',
44     #transform= transforms
   )
46 batch_size = 15
   learning_rates = [0.001, 0.0001, 1e-4, 1e-5, 1e-6]
48


50
   num_epoch = 100
52


54
   net = vgg13_bn()
56 print(net)
   # dataiter = iter(dataset_loader)
58 # images, labels = dataiter.next()
   # images = images.unsqueeze(1)
60 #
   # writer.add_graph(net, images)
62 criterion = nn.CrossEntropyLoss()


64
   writer = SummaryWriter(comment='train')
66 writer2 = SummaryWriter(comment='valid')


68
```

```
70 optimizer = optim.Adam(net.parameters(), lr=0.0001 , betas
       =(0.9, 0.999), eps=1e-08, weight_decay= 0, amsgrad=False)
   ## Perform training
72 for epoch in range(num_epoch):  # loop over the dataset
       multiple times
       dataset_loader = torch.utils.data.DataLoader(dataset,
       batch_size=batch_size, shuffle=True)
74     validset_loader = torch.utils.data.DataLoader(validset,
       batch_size=batch_size, shuffle=True)
       dataiter = iter(dataset_loader)
76     images, labels = dataiter.next()
       images = images.unsqueeze(1)
78
       writer.add_graph(net, images)
80     criterion = nn.CrossEntropyLoss()
       print('Epoch: ', epoch)
82
       running_loss = 0.0
84     total_correct = 0.0
       val_correct = 0.0
86     test_loss = 0.0

88     for i, data in enumerate(dataset_loader):
           # get the inputs; data is a list of [inputs, labels]
90         # print('In the epoch ', epoch, ' running iteration:
       ', i)

92         device = torch.device("cuda:0" if torch.cuda.
       is_available() else "cpu")
           net.to(device)
94
           # print(data)
96
           inputs, labels = data
98         # print(labels)
           inputs = inputs.unsqueeze(1)
100
           inputs = inputs.to(device=device)
102        #
       ######################################################################

           labels = labels.to(device=device)
104
           # zero the parameter gradients
106        optimizer.zero_grad()

108        # forward + backward + optimize
           outputs = net(inputs.float())
110
```

```
        outputs = outputs.to(device=device, dtype=torch.
    float32)
112         _, preds = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)
114         # print(preds)
            # print(labels)
116
            loss.backward()
118         optimizer.step()
            torch.no_grad()
120
            # print statistics
122         running_loss += loss.item()
            total_correct += get_num_correct(outputs, labels)
124
            # Validation
126
            # if i % 30 == 29:  # print every 100 mini-batches
128         #
            #     # for j, val_data in enumerate(validset_loader):
130         #     #
            #     #     val_inputs, val_labels = val_data
132         #     #     val_inputs = val_inputs.unsqueeze(1)
            #     #     val_inputs = val_inputs.to(device=device)
134         #     #     #
    ##########################################################################

            #     #     val_labels = val_labels.to(device=device)
136         #     #     val_outputs = net(val_inputs)
            #     #
138         #     #     val_loss = criterion(val_outputs,
    val_labels)
            #     #     loss_cpu = val_loss.cpu().detach().numpy()
140         #     #     test_loss = test_loss + loss_cpu
            #     #     val_correct += get_num_correct(val_outputs
    , val_labels)
142         #
            #     print('[%d, %5d] Training loss: %.3f' %
144         #           (epoch + 1, i + 1, running_loss / 30))


146
                    # test_loss = 0.0
148                 # running_loss = 0.0

150             # print(total_correct)
    for j, val_data in enumerate(validset_loader):
152         val_inputs, val_labels = val_data
            val_inputs = val_inputs.unsqueeze(1)
154         val_inputs = val_inputs.to(device=device)
```

```
        #
    ###################################################################

156         val_labels = val_labels.to(device=device)
            val_outputs = net(val_inputs)
158
            val_loss = criterion(val_outputs, val_labels)
160         # loss_cpu = val_loss.cpu().detach().numpy()
            test_loss += val_loss.item()
162         # print('Correct Lables Are:', val_labels)
            # print('Per Batch accuracy', get_num_correct(
    val_outputs, val_labels)/len(val_outputs))
164         # print('Predicted Labels are:', val_outputs.argmax(
    dim=1))
            val_correct += get_num_correct(val_outputs, val_labels
    )
166
    writer.add_scalar('Training loss', running_loss/len(
    dataset), epoch)
168 writer.add_scalar('Train_Accuracy', total_correct / len(
    dataset), epoch)
    writer2.add_scalar('Validation Loss', test_loss/len(
    validset), epoch)
170 writer2.add_scalar('Validation Accuracy', val_correct /
    len(validset), epoch)
    writer.add_scalar('loss', running_loss/len(dataset), epoch
    )
172 writer2.add_scalar('loss', test_loss/len(validset), epoch)
    writer.add_scalar('Accuracy', total_correct / len(dataset)
    , epoch)
174 writer2.add_scalar('Accuracy', val_correct / len(validset)
    , epoch)
    print(val_correct)
176 print(len(validset))

178 writer.close()
    writer2.close()
180 torch.cuda.empty_cache()
    print('Finished Training')
```

Training.py

G Plotting

```
0 import matplotlib.pyplot as plt
  import pandas as pd
2


4 # def plotter(filename1, filename2):
  #     Train = pd.read_csv(filename1)
```

```
#       Valid = pd.read_csv(filename2)
#       x1 = Train['epoch']
#       x2 = Valid['Step']
#       y1 = Train['Value']
#       y2 = Valid['Value']
#       plt.plot(x1, y1, label='Training', color= 'r')
#       plt.xlabel('Epoch')
#       plt.ylabel('Accuracy')
#       plt.plot(x2, y2, label='Validation', color= 'bo')
train_csv ='Results/Vgg1310CVTrainAccuracy.csv'
valid_csv ='Results/Vgg1310CVValidAccuracy.csv'
train = pd.read_csv(train_csv)
valid = pd.read_csv(valid_csv)
train['EMA'] = (train['average'].ewm(span=8,adjust=False).mean
    ())
valid['EMA'] = (valid['average'].ewm(span=8,adjust=False).mean
    ())
train_max = train.idxmax(axis=0)
Training_EMA_max = train_max['EMA']
Training_Accuracy_max = train.iloc[Training_EMA_max]['EMA']
valid_max = valid.idxmax(axis=0)
Valid_EMA_max = valid_max['EMA']
Valid_Accuracy_max = valid.iloc[Valid_EMA_max]['EMA']
plt.figure(figsize=[20,15])

plt.grid(True)
plt.rc('font', family='serif')
plt.title('10 Fold Cross Validation Average Accuracy For Vgg13
    ', fontsize = 32)
plt.yticks(fontsize = 20)
plt.xticks(fontsize = 20)
plt.ylabel('Accuracy', fontsize = 24)
plt.xlabel('Epoch', fontsize = 24)
plt.plot(train['average'], 'bo', label= 'Training Average',
    markersize = 7, alpha = 0.4)
plt.plot(valid['average'], 'r^', label= 'Validation Average',
    markersize = 7, alpha = 0.4)
plt.plot(train['EMA'], color='blue', ls = 'solid', linewidth =
     5)
plt.plot(valid['EMA'], color='red', ls = 'dashed', linewidth =
     5)
plt.legend(loc=7, fontsize = 22)
plt.annotate('Training Max: ' +"{:.1%}".format(
    Training_Accuracy_max), xy=(Training_EMA_max,
    Training_Accuracy_max), xytext=(0, 0.93),
              fontsize= 26)
plt.annotate('Validation Max: ' +"{:.1%}".format(
    Valid_Accuracy_max), xy=(Valid_EMA_max, Valid_Accuracy_max
    ), xytext=(0, 0.92),
              fontsize= 26)
```

```
    plt.savefig('10FoldAccuracy')
46  plt.show()
```

Plot_CV_accuracy.py

H Presentation

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Automatic contour artefact detection in radiotherapy planning

1

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

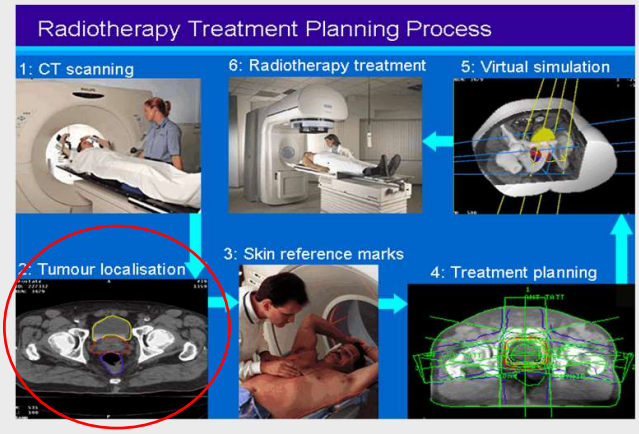
What problem are we trying to solve?

- **During the treatment planning process for radiotherapy it is necessary to delineate organs and risk (OARs) from the tumor volume**

2

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

What problem are we trying to solve?

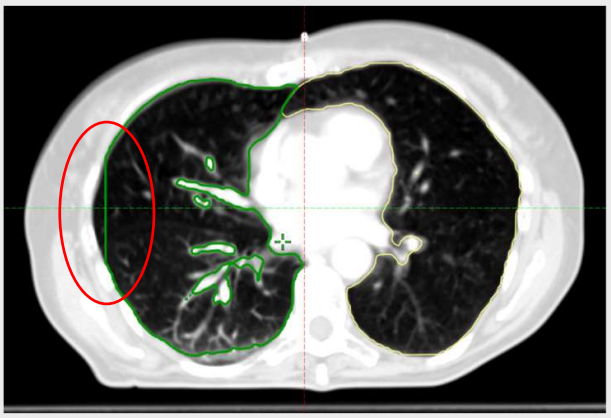


3

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

What problem are we trying to solve?

Occasionally delineation errors occur.

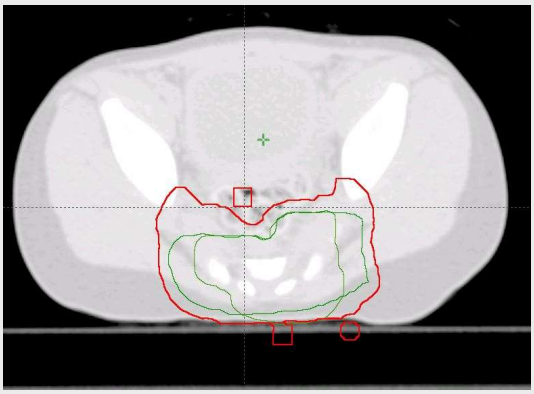


4

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

What problem are we trying to solve?

They can also be of the form of

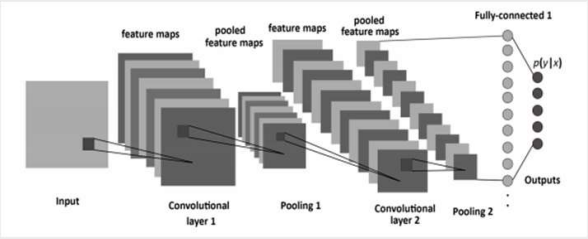


5

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

How do we intend to solve this problem?

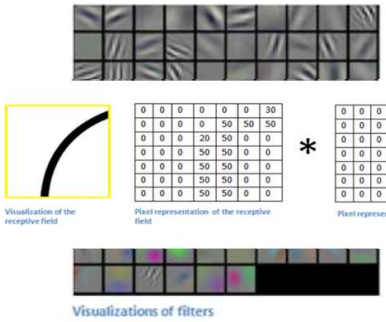
We employ a Deep Convolutional Neural Network (CNN).

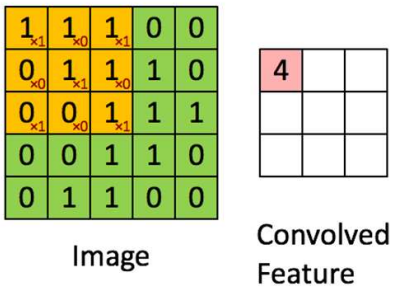


A Framework for Designing the Architectures of Deep Convolutional Neural Networks. S.Albelwi et all 2017

6

UCL Engineering
Department Of Medical Physics and Biomedical Engineering





What is a CNN and how does it work?

For more information check Jason Yosinski –Deep – Visualisation Toolkit

https://www.youtube.com/watch?v=AgkfIQ4IGaM

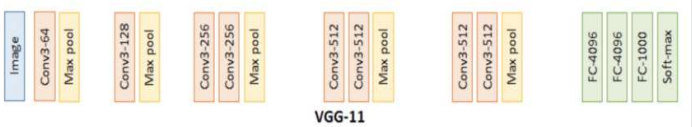
7

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

What Our Network Looks Like*

*Currently

- **Currently using a 2D Vgg11 architecture which we have adapted for 3d inputs**

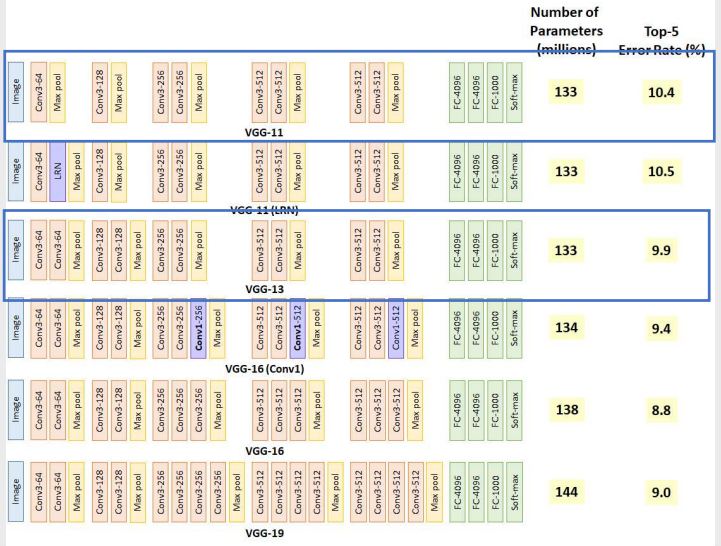


8

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

UCL

VGG Architecture Review



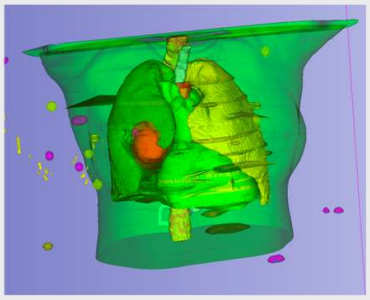
9

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

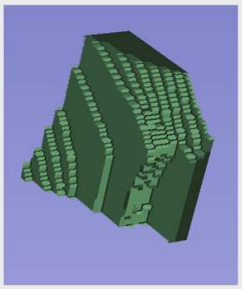
UCL

Data Preprocessing

Going from…
A full volume CT



…..To extracting
A 48x48x48 Volume



10

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

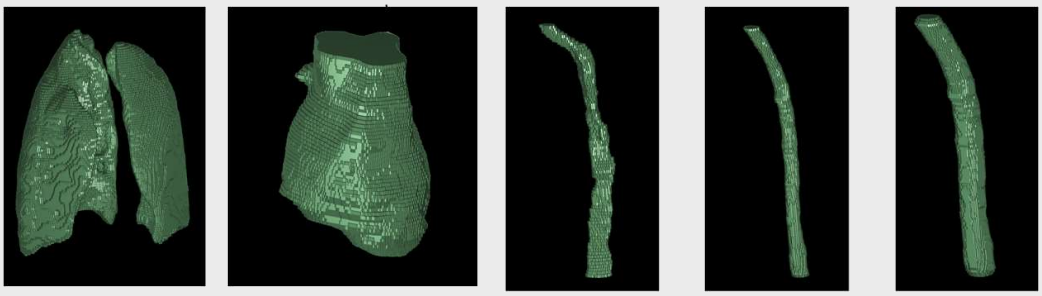
The Data

- Originally 50 patients
- 2 removed due to difficulties with contour extraction
- 40 patients for training
- 8 for validation
- 5 – 6 organ structures, depending if lungs were separated.
- Each organ split into 48x48x48 patches. Approximately 100 patches per organ
- 1997 items of training data
- 347 items of validation data

11

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Structures Examined

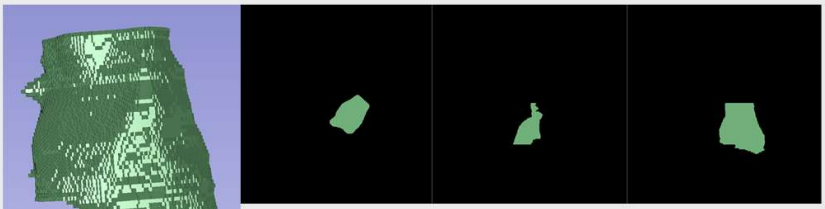


Heart | Lungs | Oesophagus | Spinal Cord | Spinal Canal PRV

12

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Contour Extraction

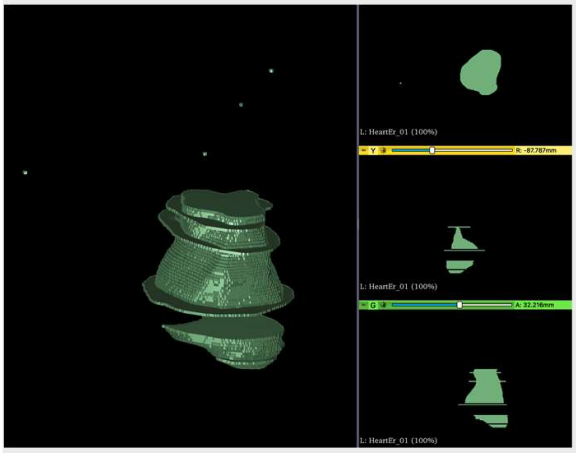


```
structure_names = ['Spinal CanalEr','Spinal CanalCor','SpinalCanalEr','Spinal Canal', 'SpinalCanal', 'SpinalCorEr', 'CANALCor',
                   'CANALEr','Heart', 'HeartEr','HEART', 'HEARTEr', 'LUNGS','LUNGSCor', 'LUNGSEr','LungsEr', 'L Lung',
                   'L LungEr', 'Lt Lung', 'LtLung', 'Lung_L', 'Lung_LEr',
                   'LLungEr', 'RLung', 'R Lung', 'Rt Lung', 'R_Lung', 'Lung_R', 'RightLungCor', 'R_LungEr', 'Lung_REr',
                   'RightLungEr', 'RTLungEr', 'RLungEr', 'Rt LungEr', 'OesophagusCor','Oesophagus', 'OesophagusCor',
                   'CanalPRVCor','CanalPRVEr','Canal PRVCor', 'Canal PRVEr', 'Spinal CanalPRVEr', 'CANAL PRV', 'CANAL PRVEr',
                   'Canal + 5mm', 'Canal + 5mmEr', 'Spinal canal PRV', 'canalprvER', 'Canal PRV']
```

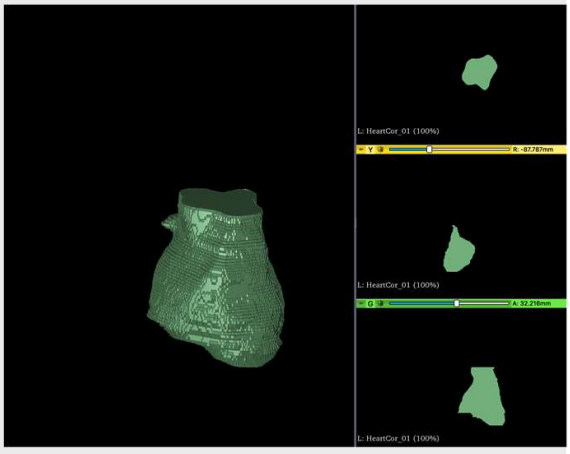
13

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Example of a structure containing errors

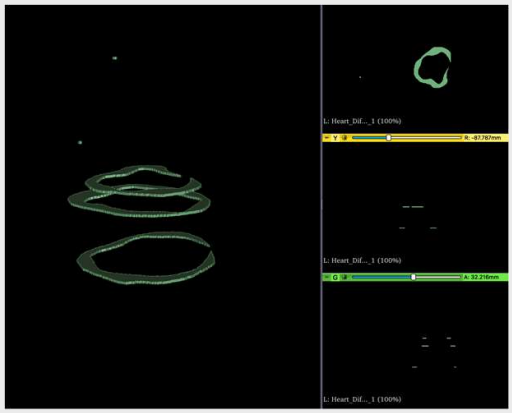


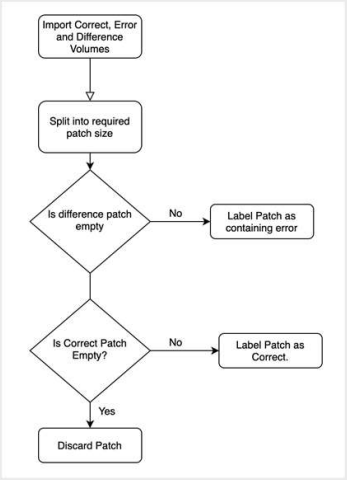
Example of a correct structure



14

Patch Extraction





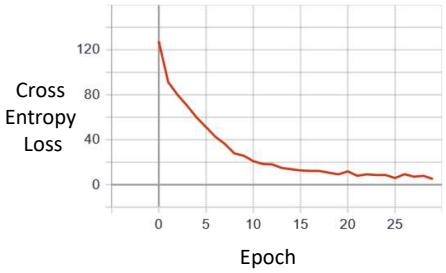
15

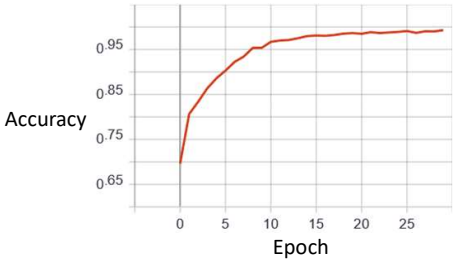
UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Results

16

UCL Engineering
Department Of Medical Physics and Biomedical Engineering



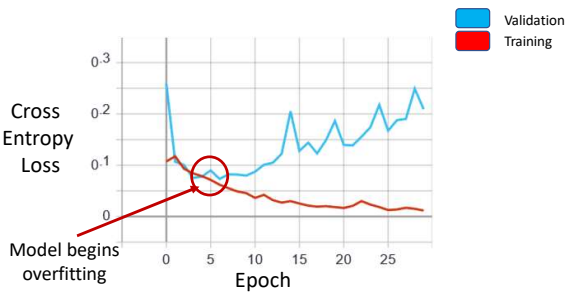


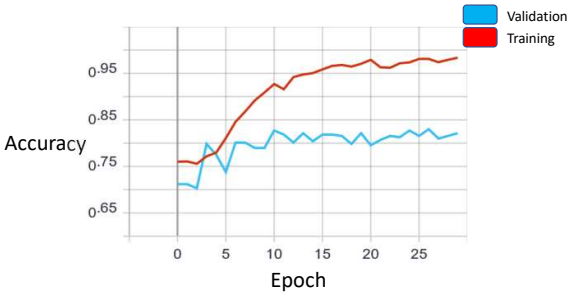
Initial Training Performance 40 Patients in Training set

VGG11, Batch Size = 5, Patch Size = 48x48x48

17

UCL Engineering
Department Of Medical Physics and Biomedical Engineering





Validation Performance – 8 Patients in Validation

VGG11, Batch Size = 5, Patch Size = 48x48x48

18

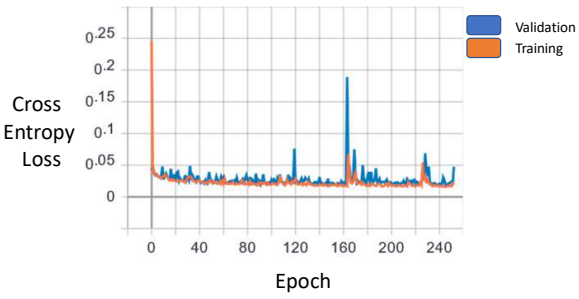
UCL Engineering
Department Of Medical Physics and Biomedical Engineering

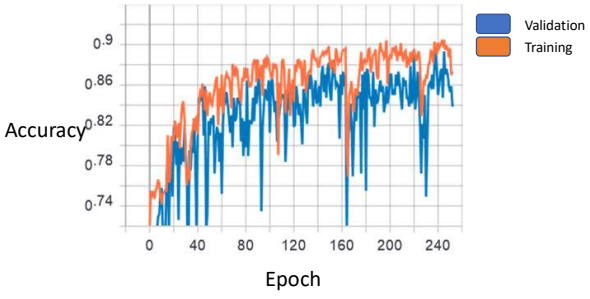
Reducing Overfitting

- Using a deeper network, attempting VGG13. Some early success in training, validation is currently no better than VGG11.
- Adjusting numbers of filters in each layer.
- Normalisation and regularization techniques to improve generality. (Dropout, more data.)
- Introduce early stopping
- If we use a patch size of 24x24x24, 6789 items in the training dataset

19

UCL Engineering
Department Of Medical Physics and Biomedical Engineering



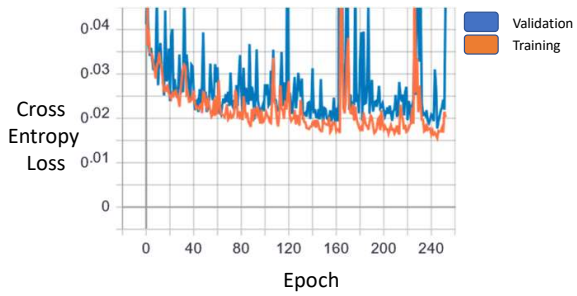


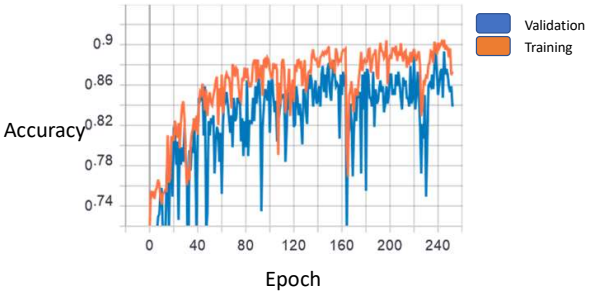
Validation Performance – 8 Patients in Validation

VGG13, Batch Size = 15, Patch Size = 48x48x48, Data Augmentations = Random Vertical and Horizontal Flips

20

UCL Engineering
Department Of Medical Physics and Biomedical Engineering





Validation Performance – 8 Patients in Validation

VGG13, Batch Size = 15, Patch Size = 48x48x48, Data Augmentations = Random Vertical and Horizontal Flips

21

UCL Engineering
Department Of Medical Physics and Biomedical Engineering

Project Future.

- **Develop a suitably accurate model.**
- **Introduce a localisation tool.**
- **Expand classification from binary to specify each type of error suspected.**

22

Any Questions?