Dose Prediction for Radiotherapy of Advanced Stage Lung Cancer

by

Rachna Singh

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for Master of Science

McMaster University
MASTER OF SCIENCE (2020)
Hamilton, Ontario (Radiation Sciences)
TITLE: Dose prediction for radiotherapy of advanced stage lung cancer.
AUTHOR: Rachna Singh
SUPERVISOR: Professor Marcin Wierzbicki
NUMBER OF PAGES: xiv, 68

ABSTRACT

A dose prediction model for treatment planning was generated using U-Net architecture. The model was generated for advanced stage cancer patients. The U-Net architecture was created with depth=6 and kernel=6. The model architecture was successful to reduce the input image size (192X192) to feature map (6X6) which helped to extract the low level features. The dose prediction of the model was trained with depth=6, kernel=6, MSE loss, Adam optimizer, 1000 epochs and a batch size of 4. The predicted dose was rescaled for gamma analysis to quantify accuracy of the model. The renormalized predicted dose was quantified using gamma analysis with a 3mm, 3% dose tolerance. The gamma map was generated to visualize the regions where dose distributions failed. The gamma percentage values obtained on the training set were acceptable. The mean and standard deviation values of gamma pass percentage obtained on training dataset were 97.5% and 1.24% respectively, which concluded that training process was successful and was an almost perfect match of true dose and predicted dose. However, gamma pass percentage values obtained on validation set was not a good representation of the true dose. Nevertheless, the validation dataset was able to predict the approximate highest dose region. A gamma analysis with a 5mm, 5% dose tolerance was performed to test the the level of discrepancy between the predicted and true dose in the validation set. This increased the gamma pass percentage compared to the 3mm, 3% analysis to a mean gamma pass percentage of $26.2 \pm 7.47\%$. Although the predicted dose was not of sufficient accuracy for clinical use, there technique studied in this work show promise for further

development.

DEDICATION

For

My grandfather

who would take pride in every small step I took…….

ACKNOWLEDGMENTS

I express my sincerest gratitude to my advisor, Prof. Marcin Wierzbicki, for his invaluable assistance, support and guidance in this research. Prof Wierzbicki's enthusiasm and aspiration for last one year has helped me to perform this research. I hope to emulate his abilities in my forthcoming research career.

I express my deepest gratitude to Prof. Tom Farrell for recommending me and providing me with confidence and believing in me. I would also like to thank Prof. Orest Ostapiak and Prof. Roxana Vlad for being in my committee.

On a personal note, I would like to thank my cousin, Shashank Bhushan for his helpful inputs during the research.

Finally, I would like to thank my husband, Kamesh, for his support during this journey (especially with the monumental task for teaching me python and pytorch programming) and my children Spriha and Kashvi for not bothering me when I am at my work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

Introduction

1.1 A brief introduction to deep learning

Artificial intelligence (AI) refers to the science and engineering of computer simulation of human intelligence. Its development and deployment has become the most discussed topic in industry and academia. In the last 15 years, machine learning (ML), a subcategory of AI, has been a prominent area of research and development due to its ability to solve complex practical problems by learning from available data[1].In recent years, another exceptional achievement has been obtained in the field of deep learning (DL), a subcategory of ML. DL[2,3] is based on neural networks which are programed to learn with various degree of supervision from unstructured data. DL requires some prior knowledge to learn the system, however it's capability to process the data and extract meaningful representations is outstanding. DL aims to utilize multiple neural networks to hierarchically extract high level features from the low level features of the provided input. DL has overshadowed traditional ML methods due to the accessibility of large-scale datasets in combination with advanced computing systems equipped with model training algorithms.

The main difference between AI, ML and DL is the range of scope by which a problem can be solved. AI[4,5] refers to the intelligence developed by computers to achieve abilities similar to human intelligence. The machine is capable of providing solutions to wide array of problems, where it is not restricted to categorize, comprehend, recognize patterns and can also provide conclusions. On the other hand, ML comprises numerical algorithms mostly restricted to performing established tasks via a specially designed model. Hence, ML algorithms extract statistics from data to find meaningful relations already present in the data. DL[6] employs multiple methods of ML to enhance the ability of the machine to identify the smallest correlations. This

process is known as a deep neural network, where it utilizes multi-layer architecture to computationally generate the predicted output. The diagram below shows the relationship between AI, ML and DL.

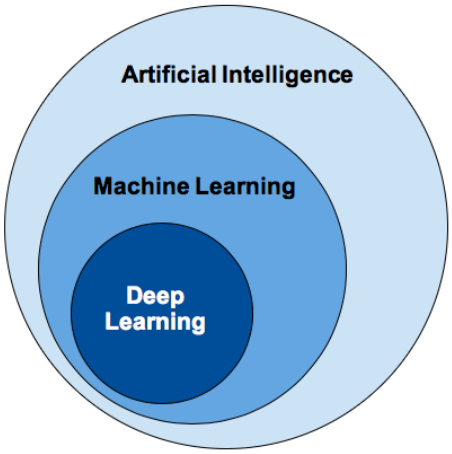


Figure 1.1: Relationship between artificial intelligence, machine learning and deep learning[7].

1.2 Biology of a neural network:

The human brain comprises approximately 10 billion neurons, our basic units of computation. A biological neuron consists of two most important parts: dendrites and axons. Each dendrite communicates with an axon with the help of synapses. The main underlying principle is that each neuron has its own electric potential and will excite only if it reaches a particular threshold. The synaptic strength decides the interaction level of the signals in the neurons. Below is the diagram of the biological neuron of the human brain.

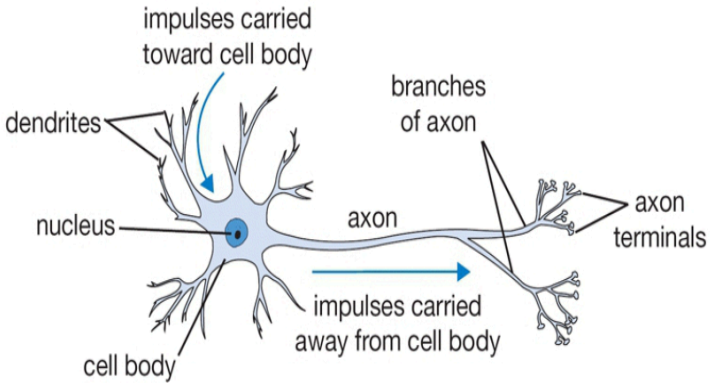


Figure 1.2: Biological neuron[8]

In AI, biological neurons are modelled using representations such as the Perceptron[8] shown in Figure (1.3). Signals x_i arrive at synapses where each is multiplied by a unique weight, w_i. The weighted signals are summed and a bias b is added. The bias represents the firing threshold of the neuron. A non-linear activation function, f is then applied to the total signal to generate the output, y. Equation (1.1) represents decision boundary of such outcome of the activation of a neuron, y_i

$$y_i = f(\sum_i w_i x_i + \mathrm{b}) \tag{1.1}$$

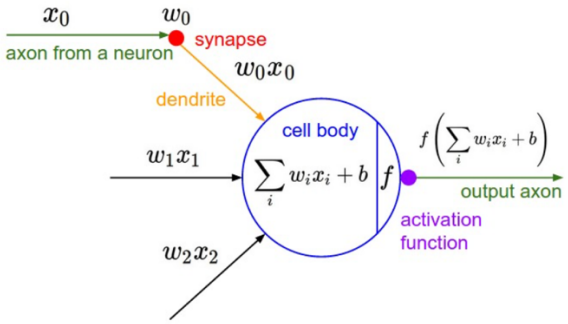


Figure 1.3: Single layer perceptron depicting one decision boundary[8].

The complicated functionality of biological neurons requires the use of a nonlinear activation function in artificial neurons as it is certain that the desired output will not be a linear combination of the inputs.

1.3 Timeline of deep learning

DL has increased learning capabilities due to the large number of neuronal layers. Training a DL model requires the optimization of the network parameters to reduce the error between the computed and the desired output. One network architecture that is interesting for the work in this thesis is the convolutional neural network (CNN). CNNs utilize kernels to extract and preserve the spatial location of features to predict the outcomes in computer vision applications. The first CNN was presented in 1988 by LeCun *et al.*[9] The lack of computational power and knowledge of complicated pipeline to train the model was the major hindrance in its development. It did not gain recognition until 2012 when Alex Krizhevsky e*t al.*[10] used a CNN to win the ImageNet competition[11]. Their group developed a CNN with five convolution layers, comprising 60 million parameters and 650,000 neurons. The network classified 1.2 million images, 50,000 validation images and 150,000 testing images into 1,000 exclusive categories. This became the major breakthrough in the field of deep learning. With further advancement, AI became highly interesting in many fields of study and after 2010, there has been observed increment of publications as shown in Figure (1.4)[12].

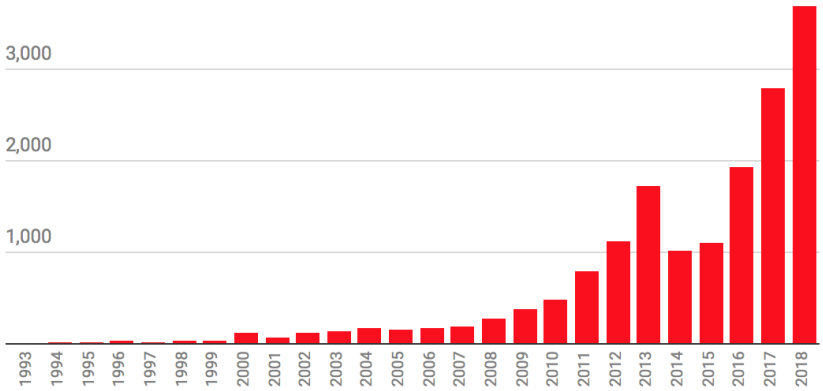


Figure 1.4: Number of papers published with "artificial intelligence" subject till Nov 2018[12].

With the rapid progress in DL, high computational power became increasingly necessary to train neural networks. Thus, there is a correlation between developments in computational power and DL[13,14,15,16].The major contributor to increased computational power has been the replacement of the central processing unit (CPU) with graphics processing units (GPU)[17]. GPUs are highly parallelized and designed for floating point arithmetic, making them well suited to the task of training deep networks.

Another factor leading to increased use of DL is the improving availability of computational packages and frameworks that make implementation straightforward. These frameworks offer predefined activation functions, different methods of loss calculations, optimization algorithms etc., allowing users to apply the functions directly for their required task. This has eased the implementation of the methods such that effort can now be diverted away from the task of basic construction of the models.

PyTorch[14] is one framework that has gained support in last three years. It utilizes the Python programming language which is easy to use and comes with many available packages for data visualization and processing. This has been developed by Facebook's AI Research lab and has been found useful for computer vision programming. It consists of an open source machine learning library based on the Torch library. The Torch library[18,19] supports multidimensional arrays known as tensors. Tensors in PyTorch store matrices, vectors, numbers, arrays etc. It is also embedded with a computationally effective C++ runtime module, which can be used for validation without the need for Python. These properties of PyTorch allows it to enable seamless implementation of DL on the GPU.

1.4 External beam radiotherapy treatment planning

Radiotherapy aims to destroy cancerous cells via energy deposited by ionizing radiation. The basic goal is to maximize the lethal dose of radiation delivered to the tumor while minimizing the dose to the surrounding tissue. The process starts with patient positioning and the acquisition of high quality images. Images are acquired using a variety of modalities, including positron emission tomography (PET), magnetic resonance imaging (MRI) and computerized tomography (CT). Typically, the CT image is the main modality used for target and organ at risk visualization. The image is used to delineate the carcinogenic tissue and identify proximal organs at risk (OARs). CT images also provide attenuation coefficients of tissues and are thus useful for dose computation during treatment planning.

A team of radiation oncologists, radiation therapist and medical physicists then select the appropriate linear accelerator beam energy, position, angles, and so on. In intensity modulated radiotherapy (IMRT), beam parameters are then optimized such that acceptable target coverage and OAR sparing is achieved. Generating a minimally acceptable plan is usually a quick process, however, improving the plan for the patient is laborious and often requires many iterations between the planners and the treatment planning system. In this process, the radiotherapy team may need to interact repeatedly based on intermediate plans. This iterative process requires tremendous human efforts which may result in the delay of patient's treatment or acceptance of a sub-optimal plans. Figure (1.5(A),1.5(B)) demonstrates the present treatment planning structure and a potential future workflow involving AI.

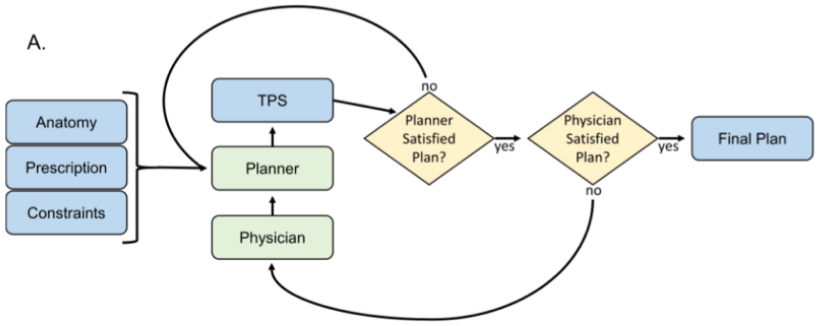


Figure 1.5(A): Present radiation treatment plan structure [22]

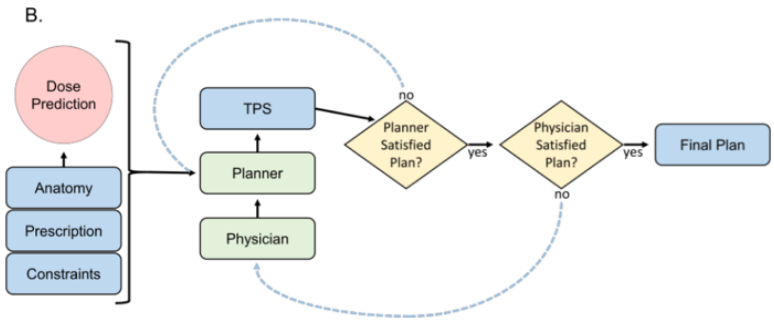


Figure 1.5(B): Future radiation treatment planning structure with AI[22]

The proposed planning method with AI will save time due to fewer iterations between the physician and dosimetrist. The dose prediction generated using the AI method will reduce the time necessary for the dosimetrist to generate an acceptable plan since knowledge of what is achievable will reduce the number of iterations between the physician and dosimetrist. For example, prior dose knowledge may be used to set achievable dose objectives during inverse treatment plan optimization.

Another issue in the current workflow is that the planned treatment is acceptable for the patient anatomy as seen in the original CT scan. Radiation treatment is given in multiple fractions over many days. The anatomy of the patient will change daily which will ultimately lead to change

in the delivered dose distribution. The patient positioning is also an important component in the computation of dose and will have to be readjusted due to the change in anatomy of the patient. AI is expected to play an important in such situations allowing faster evaluation of the impact of positional and anatomical changes on the dose distribution.

1.5 Motivation

Stage III non-small cell lung cancer (NSCLC) is typically treated with chemotherapy along with radiotherapy at a daily dose of about 2 Gy over about 30 daily fractions[20]. The initial radiotherapy treatment plan is designed as described above and the dose distribution is approved by a team comprising radiation oncologists, radiation therapists, and medical physicists. For each fraction, the patient is positioned on the treatment unit and a cone-beam CT (CBCT) image is captured. The CBCT image is used to ensure patient positioning and anatomy are as close as possible to the geometry observed during treatment planning. This ensures the planned dose distribution is delivered each day. However, advanced stage lung cancer patients experience several tissue changes during a treatment course. Patients suffer from weight loss due to the toxicity of the treatment[23]. Also, with the ongoing chemotherapy, the tumor size is reduced which impacts the dose distribution plan. There are also changes in the accumulation of fluid in the lung due to treatment. With all the changes occurring during the treatment, there is an overall variability in patient setup. These changes make it impossible to reproduce the planned dose distribution each day. Instead, the treating radiation therapists are constantly analyzing the differences in anatomy between the planning CT and treatment CBCT images. At some threshold, the therapists decide it is no longer reasonable to continue delivering the original plan and, with further analysis by the attending radiation oncologist, the patient is sent for a new CT scan. The dose distribution is

recomputed by the traditional method discussed above and the acceptability of the plan is determined. This is a laborious and time consuming process. This thesis aims to study if it is possible to predict the dose using the daily cone-beam CT (CBCT) scan to help decide if the treatment dose distribution is acceptable just prior to delivery. Having the full dose distribution of the day also allows one to make decisions on the cumulative dose delivered. Unfortunately, due to time constraints, this work was only able to focus on the prediction of dose based on the planning CT.

Prediction of dose using DL has been a popular area of study in the last two years. Some authors demonstrated outstanding predictions of dose for prostate and head and neck cancers[21,22] using different models and frameworks . Nguyen *et al.*[22] for example, trained U-Net to predict the planned dose given contours of the planning target volume (PTV), bladder, body, left femoral head, right femoral head, and rectum. They achieved a mean dice similarity value of 0.91 for isodose volumes in the range of 0-100% of the prescription dose. The average value of absolute differences of [mean] between predicted and planned doses expressed as a percentage of the prescribed dose were [1.03%] (PTV), [4.22%] (Bladder), [0.48%] (Body), [1.79%] (L Femoral Head), [2.55%] (R Femoral Head), and [1.62%](Rectum). The most outstanding part of the research was that only limited data (88 patients) were used to predict the dose distribution. Another work looked at lung cancer diagnosis using deep learning for the annual Data Science Bowl (DSB) competition [26,23]. In DBS, teams are provided with thoracic CT scans and the goal is to predict if the patient will be diagnosed with lung cancer in the following year. 1972 teams participated and two top teams achieved lowest loss values of 0.85[24] and 0.87[25] .

The goal of this thesis was to predict dose from CT images using U-Net , similar to the approach used by Nguyen *et al.*[22]. The difference with this approach was that only CT images

were used as inputs. This is the preferred approach since no human interaction is required for dose prediction. However, dose prediction based on CT images alone is a difficult problem to solve. Programming was done in the PyTorch framework.

Chapter 2

Basics of Convolutional Neural Network

This chapter describes the basics of deep neural networks including CNNs. Later, the U-Net architecture is discussed. The chapter has been written with reference to books by Nielsen, Michael[26] and Simon Haykin[27].

2.1 Deep neural networks

Single artificial neurons are the basic structure of deep neural networks (Figure 2.1). In a feed-forward neural network, input at the first layer proceeds through hidden layers to produce the output. As each layer is connected to the next layer, the signal is "fed forward" consecutively between the layers in a forward direction. Hidden layers do not provide direct correlation between input and output layers. As deep neural networks comprise multiple hidden layers, the architecture of the model provides complex intuitions as data moves deeper into the network.

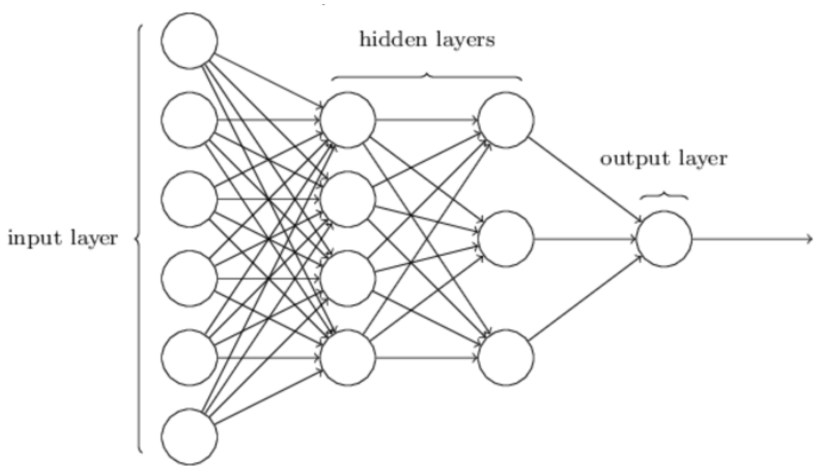


Figure 2.1: Multiple layer neural network[26].

2.2 Convolutional neural networks

A CNN is a type of deep neural network. In CNNs, a multi-channel input image can be processed with learnable weights and biases to understand its complexity and to differentiate it from other images. Figure (2.2) shows an example of a CNN designed to recognize characters.

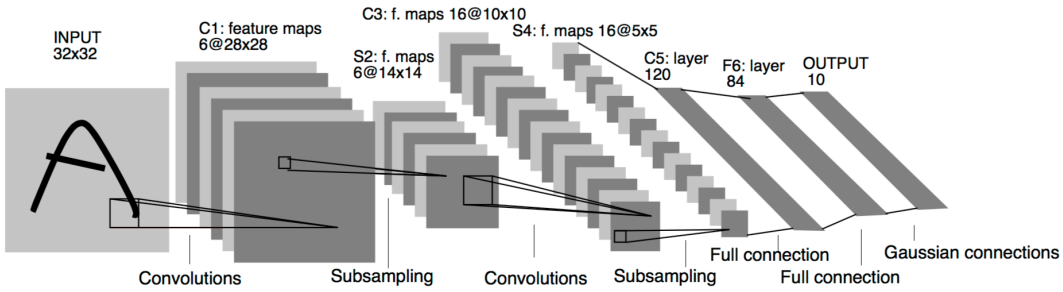


Figure 2.2: CNN architecture for digit recognition[28].

The section below discusses the formulation and terms used in the CNN network.

Convolutional layer

Convolutional layers are the building block of the CNN network. In this layer, a mathematical operation is performed to combine input data with filters/kernels to extract feature maps that highlight features such as edges for further processing. Hence, a convolutional layer produces feature/activation maps from low feature input images.

Mathematically, convolution[29] is an operation where two functions (x and w) produce a real valued y, an output. In a neural network, x is an input, w is the kernel and output is known as the feature map. The kernel is an important parameter in a CNN. It consists of a matrix of k_x rows, k_y columns, and d depth that is applied to the region of the input image known as the receptive

field. The depth of the kernel usually represents the number of channels in the input image of size (height=I_x, width=I_y and depth=d). Matrix multiplication of the input image and kernel produces a summed single value assigned to a pixel of the output feature map. The final output feature map is obtained by sliding the kernel over the input image.

During convolutions, kernel stride describes the shape of the output feature map. Stride is defined as the number of pixels the kernel moves on the input image matrix at each successive multiplication. The row and column of output matrix size, (O_x X O_y) with input image dimension (I_x X I_y), kernel size (k_x X k_y) with a stride of s is mathematically formulated as:

$$O_x = \frac{I_x - k_x}{s} + 1 \quad (2.1a)$$

$$O_y = \frac{I_y - k_y}{s} + 1 \quad (2.1b)$$

Below is a simple example where an input image of size (4X4) is convolved with a kernel of size (3X3) with a stride of 1 to produce an output feature map of size (2X2).

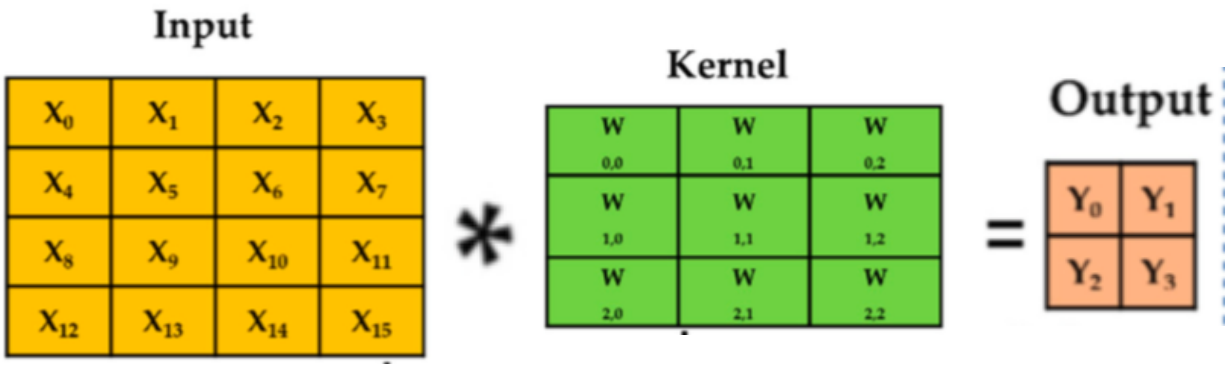


Figure 2.3: Convolution of a (4X4) image with a (3X3) kernel and a stride of 1 to obtain a (2X2) feature map.[30]

It is convenient to represent convolution using a convolution matrix. For the example in Figure (2.3), the convolution matrix would be determined as shown in Figure (2.4).

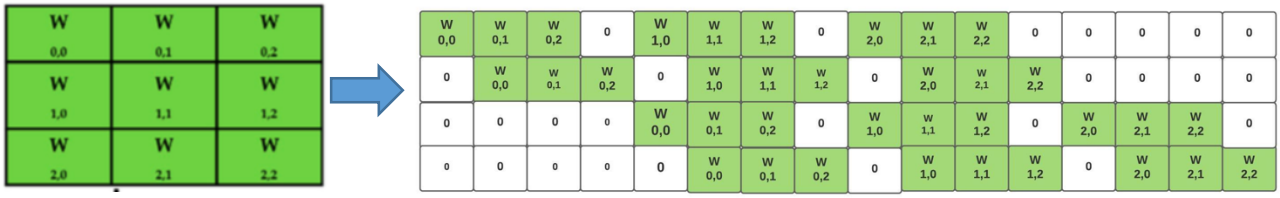


Figure 2.4: Convolution matrix (4X16) where each row represents the position of the kernel (3X3) on the input image (4X4)[30]

To perform convolution, the input image would be flattened as shown below.

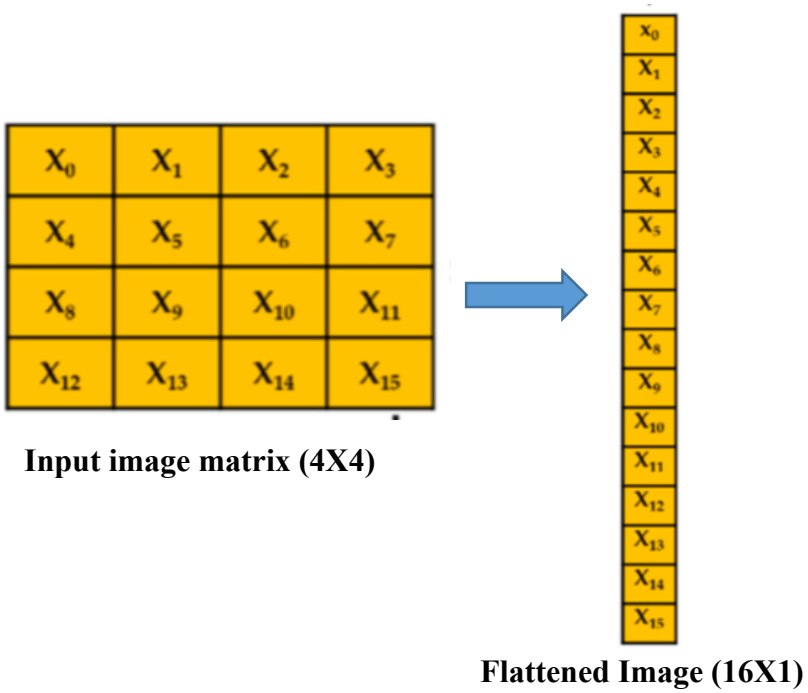


Figure 2.5: Flattened matrix of input image.[30]

The convolution operation is then obtained by matrix multiplication of convolution matrix and the flattened image as shown below in Figure (2.6)

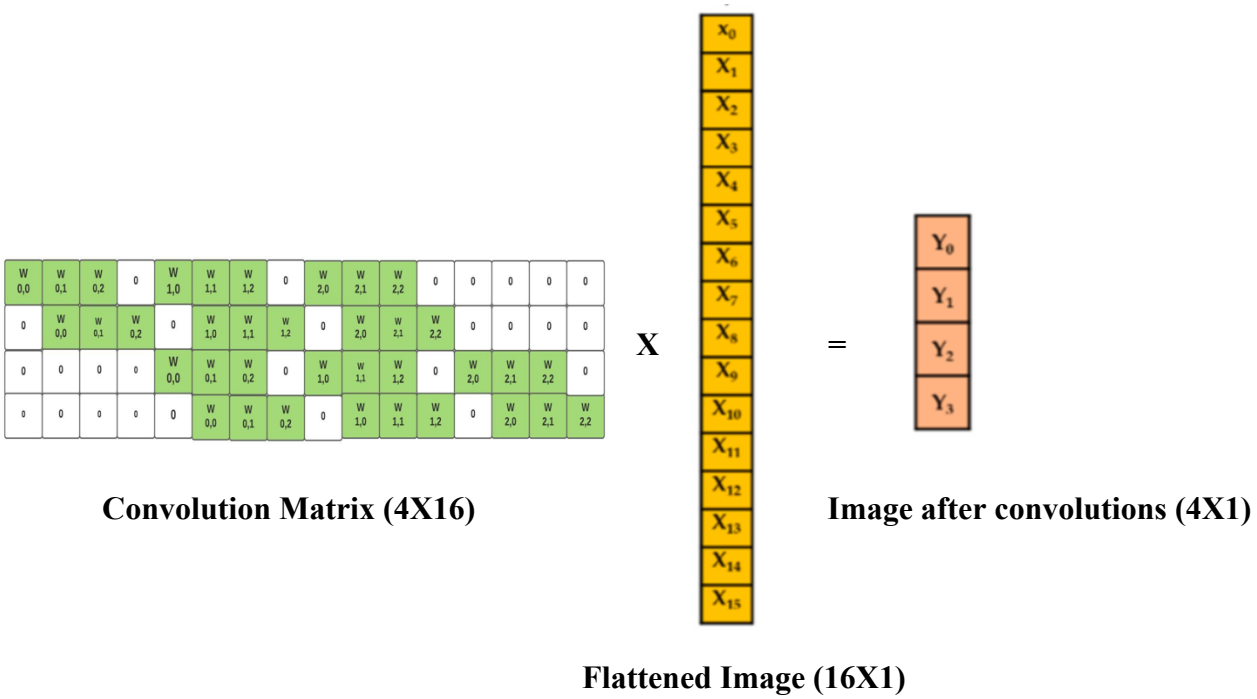


Figure 2.6: Matrix multiplication of convolution matrix and flattened image for general convolution process.[30]

Transposed Convolutions

A convolution is performed to compress the input image to generate the high level feature map with reduced spatial representation. The transposed convolution, also known as an auto encoder, is performed to decompress the abstract mapping to obtain the reconstructed output image. Figure (2.7) shows the process of convolution and transposed convolution to obtain the reconstructed image.

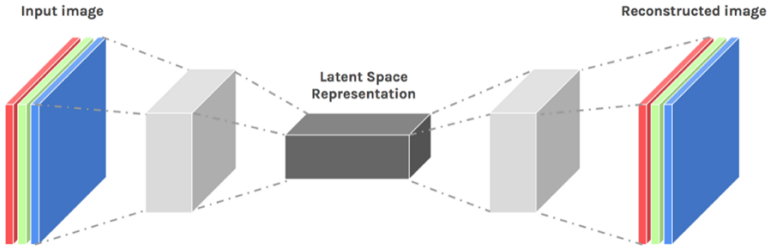


Figure 2.7: Convolution and transposed convolution method to obtain a reconstructed image from an input image.[30]

Transposed convolution can then be obtained by multiplying the transpose of the convolutional matrix and the output of the convolution. This generates the final output, recovering spatial information. Thus, transpose convolution can be thought of as a deconvolution process.

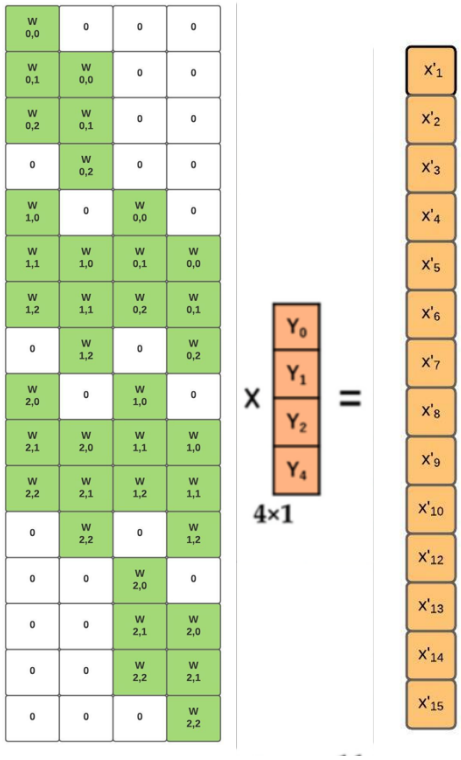


Figure 2.8: The process of transposed convolution where matrix multiplication is performed between the transposed kernel and the output of a convolution to obtain the output image[31].

Convolution implementation in PyTorch

In PyTorch, it is possible to convolve an input matrix with a general number of channels C_{in} to generate an output matrix with a different number of channels C_{out}. This formulation helps to increase or decrease the depth of the output feature map. The Conv2d function implements this using[31]:

$$Out\left(C_{out_j}\right) = bias\left(C_{out_j}\right) + \sum_{k=0}^{C_{in}-1} weight\left(C_{out_j}, k\right) \star input(k), \tag{2.2}$$

where $\star$ is the sliding dot product.

2.3 Activation function

A convolution layer in a CNN produces a feature map. Every pixel in this feature map is then sent to an activation function. Consider the example of a single neuron with four inputs $x_1 = 2, x_2 = 5, x_3 = 9$ and $x_4 = 6$, weights $w_1 = 0.1, w_2 = 0.2, w_3 = 0.3$ and $w_4 = 0.4$, and bias, $b = 7$ as shown in figure (2.9).

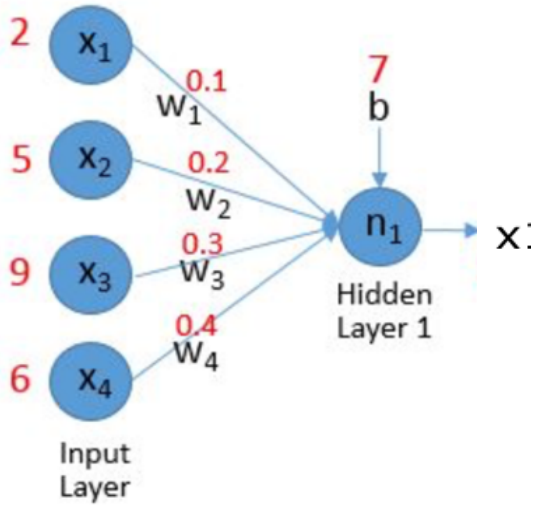


Figure 2.9: A single neuron with inputs $x_1 = 2, x_2 = 5, x_3 = 9$ and $x_4 = 6$, weights $w_1 = 0.1, w_2 = 0.2, w_3 = 0.3$ and $w_4 = 0.4$, and bias, $b = 7$. x is computed as $\sum_{i=1}^{4} x_i w_i$.[32]

The sum of the weighted input and bias is given below:

$$x = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b = 13.3$$

Using sigmoid activation, $f(x) = \frac{1}{1+e^{-x}}$:

$$Output = activation\ of\ the\ input = \frac{1}{1 + e^{-13.3}} = 1$$

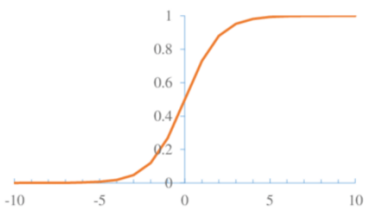
The threshold of sigmoid activation function is 0.5, hence this neuron will be fired.

Some of the most incorporated non-linear activation functions[33] are sigmoid, hyperbolic tangent, and rectified linear units (ReLU). The sigmoid function was used initially since it models the all or none response of biological neurons. However, recently, the ReLU function has is used as it improves convergence during model optimization[34]. One of the advantage of using ReLU is that the activation of neurons occurs at different times and the gradient computation of the deactivated neurons will be assigned zero if the output of linear transformation is less than zero. Equations (2.3a) and (2.3b) describe the functions:

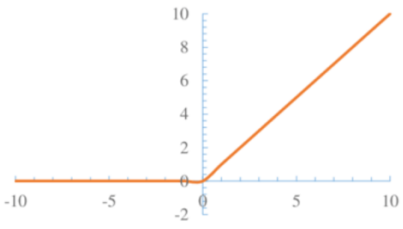
Sigmoid function : $f(x) = \frac{1}{1+e^{-x}}$ (2.3a)

ReLU function: $f(x) = \max(0, x)$ (2.3b)

Below Figure (2.10) are the functions and the activation functions.



(a): Sigmoid Function



(b): ReLU function

Figure 2.10: Example activation functions[35]

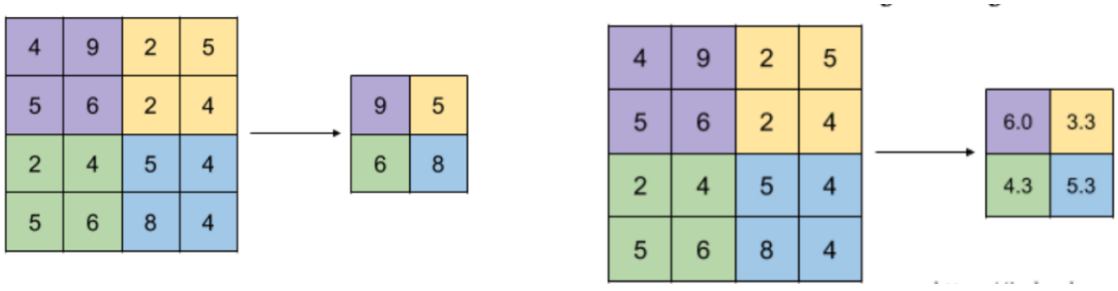
The above functions are applied in the hidden layers of the network. Softmax is another activation function that is applied at the last layer of the network to interpret the final output. It is a multi-class classifier that generates the probability distribution of N different possible outcomes where z is the input vector from the last layer. Mathematically it is represented as[36]:

$$f(z_i) = \frac{e^{z_i}}{\sum_{n=1}^{N} e^{z_n}} \text{ for } i = 1 \ldots \ldots N, \tag{2.4}$$

where z_i are the elements of input values.

2.4 Pooling layer

Pooling is usually applied on each feature map independently after the nonlinear activation function to reduce the spatial size of the representation. Reduction in the amount of parameters leads to reduced computational load in the network and decreases the chance of overfitting. Average pooling and max pooling are common pooling methods. The max pooling method summarizes the presence of the most activated feature whereas average pooling summarizes the presence of the average feature. Another important aspect of the pooling layer is that it is also implemented with using the kernel and stride concept. Figure (2.11) shows a pooling example.



(a) Max Pooling

(b) Average Pooling

Figure 2.11: Max pooling and average pooling example with a kernel of size (2X2) and a stride of length of (2x2).[37]

2.5 Gradient descent:

Gradient descent is an optimization process used to find the global minimum of a function by iteratively taking the negative direction of its gradient. In DL, gradient descent is utilized to update network parameters such as its weights and biases to minimize a loss/cost function that quantifies the difference between the predicted and desired output. Figure (2.12) shows a simple illustration of the gradient descent method.

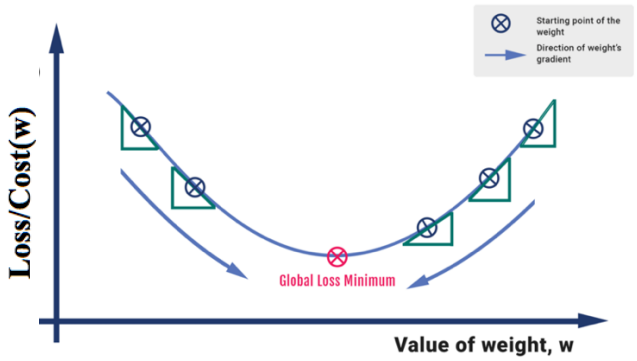


Figure 2.12: Simple illustration of gradient descent of a single weight[38].

For a single node artificial network, the cost function parameters are the weights, w_i and bias, b. A simple loss function is the mean squared difference between the predicted and the desired output as shown below[39]:

$$L(w,b) = \frac{1}{N}\sum_{i=1}^{n}\left(y_i - f(wx_i + b)\right)^2 \tag{2.5}$$

The gradient is:

$$L'(w,b) = \begin{matrix} \frac{dL(w,b)}{dw} \\ \frac{dL(w,b)}{db} \end{matrix} = \begin{matrix} \frac{1}{N}\sum -\frac{2x_i df\left(y_i - f(wx_i+b)\right)}{dw} \\ \frac{1}{N}\sum -\frac{2df\left(y_i - f(wx_i+b)\right)}{db} \end{matrix}. \tag{2.6}$$

For simplicity, let θ denote the parameters w and b. The initial loss is evaluated and the derivative of the loss $\nabla_\theta L(\theta)$ is computed as shown above. The parameters are then updated using:

$$\theta = \theta - \eta \nabla_\theta L(\theta), \tag{2.7}$$

where the learning rate parameter, η is introduced to limit the step size performed at each update. This process is repeated until the desired level of loss function is obtained. However, gradient descent does only one update for each parameter in one iteration of dataset. This process is computationally not effective if the dataset is large. With this consideration, many types of gradient descent was introduced. One of the most common implementations is the Stochastic gradient descent (SGD) method. SGD attempts to find the global minimum after each training process by fine-tuning the network parameters independently for small set of randomly selected inputs. Hence, SGD works with a randomly selected batch of data to move the model from a local minimum to global minimum. This helps to update the parameters faster since only small selection of data is processed in a single iteration of training.

Despite this, SGD is still susceptible to fining the local minimum instead of the global minimum. The computational speed of SGD is improved by introducing the concept of momentum. Momentum is used to update the slope of the loss function based on the slope in the previous iteration. The equations governing momentum in SGD are given below:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta L(\theta), \tag{2.8}$$

$$\theta = \theta - v_t, \tag{2.9}$$

where v_t is the current update vector (modified gradient), v_{t-1} is the past updated vector, t is the iteration number (time) and γ is the momentum coefficient with a typical value of 0.9.

The process of momentum in an optimizer can be compared to the situation where a ball is rolling downhill. The ball gains momentum as it progresses down the slope eventually reaching a constant velocity. Once the other side of the hill is reached, the ball continues rolling uphill due to momentum. This allows the optimizer to search beyond local minimum for possible global minima. However, this technique may still be susceptible to finding the local minimum. Thus, algorithms with adaptive learning rates such as AdaGrad[40], AdaDelta[41], RMSprop[42] and Adam[43] were introduced.

The Adaptive Gradient Algorithm (AdaGrad) is controlled by learning rate, decay learning rate, weight decay and epsilon parameters. AdaGrad is able to train large-scale neural networks due to the introduction of the adaptive learning rate. The learning rate of AdaGrad is different for every parameter and at every time and is adapted to the inconsistency in the dataset. The problem encountered with AdaGrad is the accumulation of squared terms in the denominator. This causes the learning rate to become infinitesimally small with iterations, which stops further training of the model. AdaDelta solved the diminishing learning rate problem. It restricts the size of the denominator accumulated over iterations. AdaDelta takes similar parameters as AdaGrad, however it adds $rho(\rho) = 0.9$ and removes decay learning rate. ρ is very similar to the momentum term but is applied only on the current and previous update. Root mean squares propagation (RMSprop) was developed around the time as AdaDelta to solve the AdaGrad problem. It has a similar approach as AdaGrad, however in the denominator it has an exponentially decaying average, which allows the learning rate to adapt as it is near the local minima after each iteration of the dataset. Hence, it allows the learning rate to become big and small with respect to the steepness of the gradient. The most recent optimizer is adaptive moment estimation algorithm (Adam). Adam is governed by parameters α, the learning rate, $\beta_1 = 0.9,\ \beta_2 = 0.999$, and epsilon,

ε = 1.0E-8. ε is set to a small value to allow the algorithm to avoid asymptotes. A small value of ε will make larger weight updates while a larger value of will cause smaller weight updates and will slow the training of the model, hence a small value as 1.0E-8 is preferred . The learning rate, α, controls the step size. Larger step size leads to faster convergence at the risk of potentially missing the global minimum. β_1 and β_2 are the exponential rates of first and second momentum terms. It is recommended that β_1 be near 1 and that β_2 is slightly higher than β_1. Adam has been found to work well in practice as compared to other optimizers[44].

2.6 Backpropagation

A CNN takes images as input and produces an output. Its ability to produce the desired output is controlled by various parameters that are optimized during model training. The learning algorithm process is described below[45]:

1. The network is randomly initialized with network parameters (weights and biases)
2. A validation input is allowed to pass through the network to generate a model prediction
3. The target (known truth) and the prediction are compared and a loss is calculated.
4. Backpropagation is performed to determine the gradient of the loss function with respect to the neural network parameters (details are presented in the example below).
5. The gradient descent method is used to update the parameters such that the total loss is minimized.
6. The process is repeated until the loss below a set threshold

Below Figure (2.13) describes a basic outline of the training model.

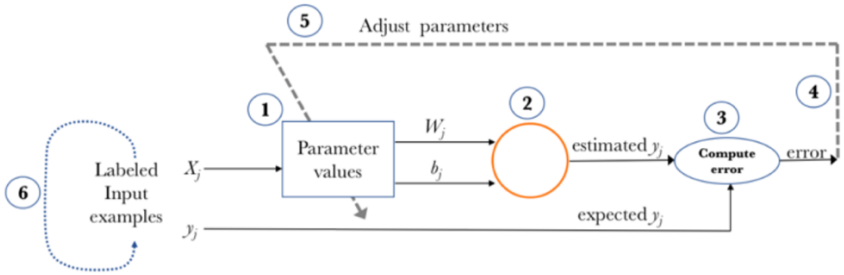


Figure 2.13: Basic outline of training a model with backpropagation[45]. Here j is the j^{th} neuron of the neural network.

During CNN training, multiple forward and backward passes are performed to optimize network parameters. The following paragraphs briefly describe the computational method of the forward and backward pass.

Forward pass

In the forward pass, an image is input into the first network layer and an activation map is produced. This activation map is the input to the first hidden layer and the next activation map is generated. The process continues through many hidden layers until the final network output is computed.

Backward pass

In this phase, the gradient of the loss function with respect to the parameters is computed through backpropagation and the gradient decent method is used to update the weights to minimize the loss. The loss function is usually a predefined function of the network.

During neural network training, an epoch elapses when the complete dataset goes through the forward and backward processes of the network.

Example: Backpropagation algorithm for training a single and multi-layer network

The mathematical foundation[26,27,29,46] in backpropagation for weight updating is an important concept in neural networks. The following derivation refers to the example network depicted below.

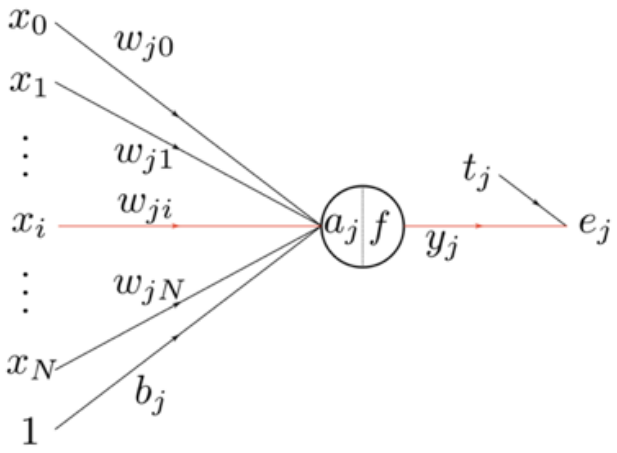


Figure 2.14: A single layer perceptron. $x_0 \ldots \ldots x_i$ are the inputs, the weights are w_{ji} and bias is $b_j = 0$. The activation function is given by f and $a_j = \sum_i x_i w_{ji}$ is the net neuron activation. $y_j, e_j,$ and t_j are the output, error and desired output of the training network[46].

The diagram shows the situation for a single training set x composed of components x_i. To allow for further discussion, let x^n represent the training set n composed of components $x^n{}_i$. The sample x^n is sent to the neural network to produce the output of the j^{th} neuron, y_j^n . The desired output is t_j^n. Bias is considered to be 0 and has thus been removed from the derivation.

The net activation is

$$a_j^n = \sum_i w_{ij} x_i^n \tag{2.10}$$

and the output is

$$y_j^n = f(a_j^n) \tag{2.11}$$

To update the weights, the error, e_j^n of the prediction should be evaluated. The error is the difference between the desired output, t_j^n and the predicted output, y_j^n.

$$e_j^n = t_j^n - y_j^n \tag{2.12}$$

The sum of squared errors of each output is considered as the total loss for this derivation. The division by 2 is used to make the derivation simpler:[47].

$$L^n = \frac{1}{2}\sum_j \left(e_j^n\right)^2 = \frac{1}{2}\sum_j (y_j^n - t_j^n)^2 \tag{2.13}$$

Now the chain rule may be used to determine the gradient of the error with respect to each weight, w_{ij}

$$\frac{\partial L^n}{\partial w_{ij}} = \frac{\partial L^n}{\partial e_j^n}\frac{\partial e_j^n}{\partial y_j^n}\frac{\partial y_j^n}{\partial a_j^n}\frac{\partial a_j^n}{\partial w_{ij}} \tag{2.14}$$

Since $\frac{\partial L^n}{\partial e_j^n} = e_j^n$, $\frac{\partial e_j^n}{\partial y_j^n} = -1$, $\frac{\partial y_j^n}{\partial a_j^n} = f'(a_j^n)$, and $\frac{\partial a_j^n}{\partial w_{ij}} = x_i^n$ the Jacobian of the output layer can be now represented as

$$\frac{\partial L^n}{\partial w_{ij}} = -x_i^n e_j^n f'(a_j^n) \tag{2.15}$$

To simplify changes in synaptic weights, we define the local gradient, δ_j^n as:

$$\delta_j^n = \frac{\partial L^n}{\partial a_j^n} = \frac{\partial L^n}{\partial e_j^n}\frac{\partial e_j^n}{\partial y_j^n}\frac{\partial y_j^n}{\partial a_j^n} = -e_j^n f'(a_j^n) \tag{2.16}$$

Hence equation (2.15) can be written as

$$\frac{\partial L^n}{\partial w_{ij}} = \delta_j^n x_i^n \tag{2.17}$$

The correction factor applied to w_{ij} is thus

$$\Delta w_{ij} = -\eta \frac{\partial L^n}{\partial w_{ij}} = -\eta \delta_j^n x_i^n \tag{2.18}$$

where η is learning rate.

The above equation updates the weights in the training dataset to minimize the loss function for a single layer perceptron. The following is an extension to the network with a single hidden layer shown in Figure (2.15). Further extension to multiple hidden layers would follow a similar process.

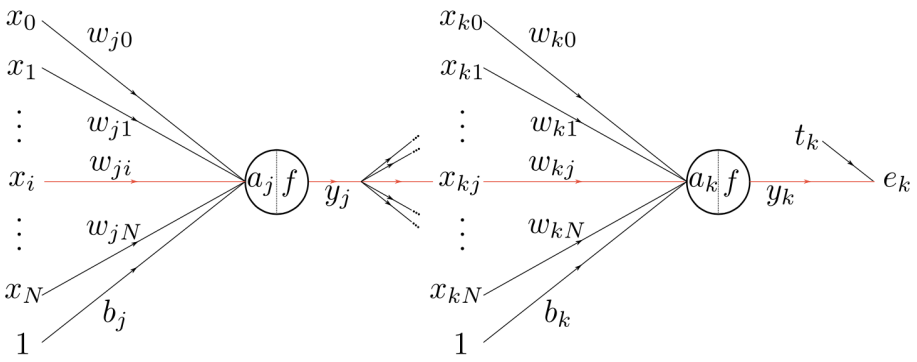


Figure 2.15: Single hidden layer perceptron. The output neurons have weights w_{kj} and the hidden neurons have weights w_{ji}[46].

The output layer's weights can be found easily using equation (2.18) because of the similar single-layer network structure. Hence, using the same delta rule error analysis, we obtain:

$$\frac{\partial L^n}{\partial w_{kj}} = \frac{\partial L^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial w_{kj}} = \frac{\partial L^n}{\partial e_k^n}\frac{\partial e_k^n}{\partial y_k^n}\frac{\partial y_k^n}{\partial a_k^n} y_j^n = \delta_k^n y_j^n \tag{2.19}$$

The hidden neuron does not have a direct relation to the error signal, hence the error calculation is obtained by working backwards with respect to the error signal obtained from all the neurons to which the particular hidden neuron is directly connected. Here the hidden layer is connected to output neuron layer (y_k). This is the main principle of back-propagation.

Hence to derive the weight update for the hidden layer, the partial derivative of error, L^n with respect to hidden weight, w_{ji} is

$$\frac{\partial L^n}{\partial w_{ji}} = \left(\sum_k \frac{\partial L^n}{\partial e_k^n}\frac{\partial e_k^n}{\partial y_k^n}\frac{\partial y_k^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial y_j^n}\right)\left(\frac{\partial y_j^n}{\partial a_j^n}\frac{\partial a_j^n}{\partial w_{ji}}\right) \tag{2.20}$$

In equation (2.20), the term in the first bracket is for the output neuron and the second term is for the hidden neuron.

From equation (2.19), we can rewrite equation (2.20) as

$$\frac{\partial L^n}{\partial w_{ji}} = \left(\sum_k \delta_k^n y_j^n \frac{\partial a_k^n}{\partial y_j^n}\right)\left(\frac{\partial y_j^n}{\partial a_j^n}\frac{\partial a_j^n}{\partial w_{ji}}\right) \tag{2.21}$$

Using equation (2.10), the net activation of the hidden layer is $a_k^n = \sum_j w_{kj} y_j^n$. and da/dy $= w_{kj}$.

Also , $\frac{\partial y_j^n}{\partial a_j^n} = f'(a_j^n)$, and $\frac{\partial a_j^n}{\partial w_{ij}} = x_i$, hence equation (2.21) can be given as

$$\frac{\partial L^n}{\partial w_{ji}} = \left(\sum_k \delta_k^n y_j^n w_{kj}\right) f'(a_j^n) x_i \tag{2.22}$$

Using the local gradient definition to hidden layer from equation (2.18)

$$\delta_j^n = \frac{\partial L^n}{\partial a_k^n} = \left(\sum_k \frac{\partial L^n}{\partial e_k^n}\frac{\partial e_k^n}{\partial y_k^n}\right)\frac{\partial y_k^n}{\partial a_k^n} = \left(\sum_k \delta_k^n y_j^n w_{kj}\right) f'(a_j^n) \tag{2.23}$$

Hence,
$$\frac{\partial L^n}{\partial w_{ji}} = \delta_j^n x_i \tag{2.24}$$

2.7 Loss function:

Loss functions quantify the difference between the predicted output and the desired output. The lower the loss function the more accurate the model. For successful training it is important that

the loss function sufficiently quantifies the differences between the predictions and desired output. Typical loss functions used in neural networks are the mean absolute error (MAE) and mean squared error (MSE).

$$\text{MAE} = L(x, \hat{x}) = \frac{1}{N}\sum_{i=0}^{N}|x - \hat{x}| \tag{2.25}$$

$$\text{MSE} = L(x, \hat{x}) = \frac{1}{N}\sum_{i=0}^{N}(x - \hat{x})^2 \tag{2.26}$$

Of the two, MSE may be more effective at training models that are able to reproduce a more varied output due to the presence of the squared term. On the other hand, MAE may be more effective at avoiding overfitting and the production of models that are more resistant to outliers.

2.8 Hyperparameters and training

As discussed before, backpropagation plays a major role in the training process of the network. Another component of network training is the setting of hyperparameters, the values in the network structure that are manually set before training begins. The learning rate, choice of optimizer, and selecting appropriate batch size before the training process are a few of the hyperparameters that need to be adjusted before the training process. The learning rate regulates the changes in network weights with respect to the loss gradient. A small rate may lead to longer optimization of the network while larger value may lead the training to converge or diverge too soon leading to incorrect trained model.

Batch size is defined as the number of training examples used in one epoch of model training. Batch size can be chosen from one of three options; batch mode where batch size is equal to total number of training cases, mini-batch mode where batch size is greater than one but less than the total number of cases, and stochastic mode where batch size is one. In batch mode, the

number of gradient decent iterations over the entire dataset and the number of epochs are equal, in mini-batch mode there will be multiple optimization iterations in one epoch and in stochastic mode, the number of iterations in one epoch is equivalent to the length of dataset. Very similar to the learning rate, choosing the appropriate batch size is imperative because a small batch size will converge or diverge faster but a large batch will attain global minima faster. The type of dataset also contributes to the decision of the batch size.

There are hyperparameters which are used for a particular network architecture. Among them are the size of convolutional kernel, type of activation functions and loss functions, et cetera. Convolutional kernel size establishes the size of receptive field in the CNN. Kernel size of (3×3) and (5×5) are common in CNNs that analyze medical images[48].

Another parameter to select is the number of training epochs. A large number of epochs may lead to model overfitting while insufficient iteration leads to underfitting. In either case, the model will not be effective in real world applications.

Setting hyperparameter is complex process in network training. The most appropriate choice is often discovered through careful observation of results rather than through theoretical knowledge of the model

2.9 U-Net architecture

U-Net is a CNN architecture purposed in 2015 by Ronneberger *et al*.[49]. It was specifically designed for biomedical image segmentation. In the original work, the input[50] consisted of 30 sections from a serial section Transmission Electron Microscopy (ssTEM) data set of the Drosophila first instar larva ventral nerve cord (VNC). Each image was accompanied with its segmentation map in black and white corresponding to membranes and cells respectively. The output image was the segmentation map of the input. It was divided into two classes, (cells) foreground and (membrane)

background class (black and white respectively). The network is very effective in providing accurate localized lesions and segmentation outputs can be obtained with a small training dataset. The training dataset can have different scales and resolutions. As presented in Figure (2.16), the name U-Net is given for its U-shape in which layers are sequenced.

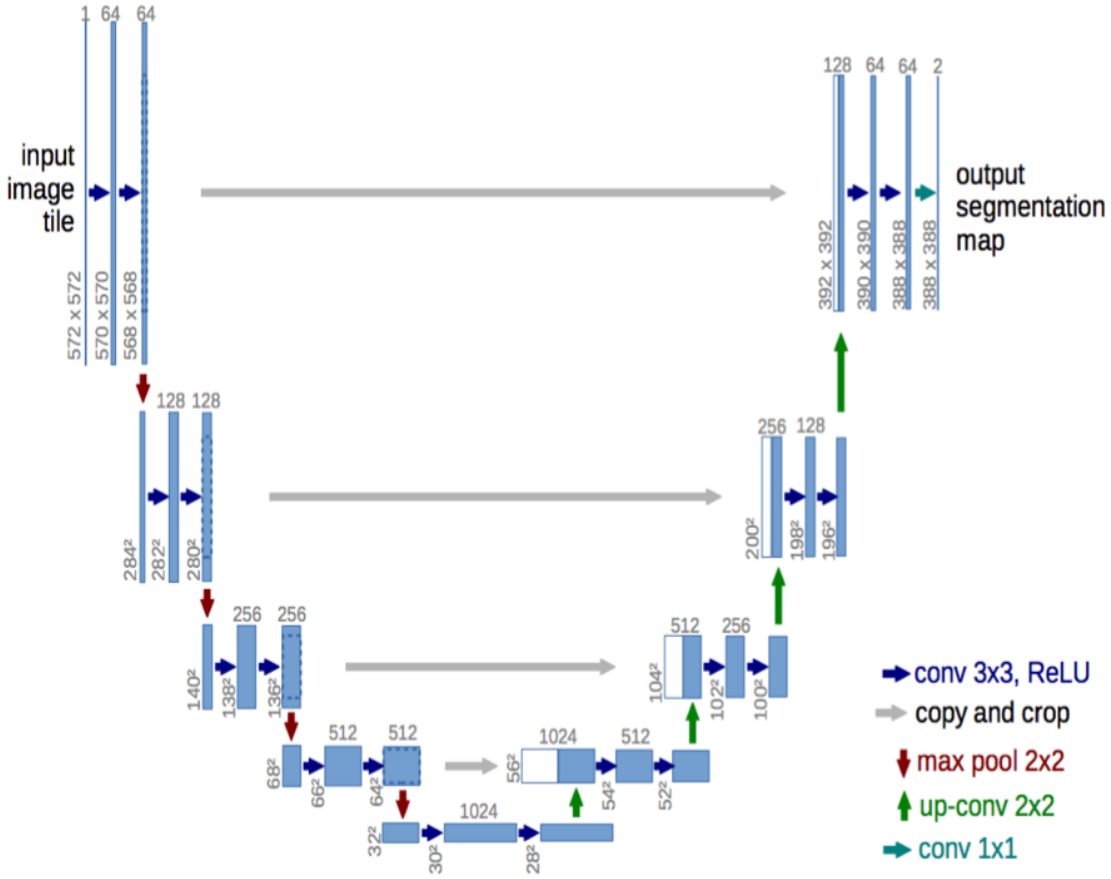


Figure 2.16: Original U-Net architecture described by Ronneberger et al.[49]

The U-Net architecture consists of an encoder and decoder and has 23 convolutional layers. The encoder, also known as the contracting path, is the part where sequential convolutions are performed to perceive detailed features. The size of input image is (572X572X1) and after four blocks of two (3X3) convolution layers and (2X2) max pooling we obtain image of size of

(64X64X512). After the encoder path, high features are obtained however spatial information is lost due to repeated convolution. Hence, the decoder is the expansive path is introduced where transposed convolutions are performed to restore the spatial information. However, before the decoder path, bottleneck part is performed, where two convolutions of (3X3) is performed without any max pooling and the size of image obtained is (28X28X1024). This bottom part allows compression of the input data to extract the useful high feature map which can be later used to reconstruct the segmentation map. In the decoder path, features along with spatial information are combined using skip connections. Skip connections are links between convolutional and transposed convolution layers. As in the above Figure (2.16), each block in the expansion path, consists of (3X3) convolutional layer followed by (2X2) up-convolutional layer. The following layer appends half of the feature map with the half of the corresponding contraction path. Though there is lack of theoretical evidence, U-Net has been able to obtain the spatial information of the high-level features to output the predicted image from skip connections. The last uppermost section of U-Net was to reshape the image to obtain the desired output. The softmax function is used to obtain the prediction. Sigmoid was the right choice in this scenario because the desired image was the segmentation map into two classes, background and foreground of the input image. Hence, U-Net architecture has proven to be successful in the segmentation task in the biomedical imaging field.

Dose prediction using U-Net

U-Net architecture has been used to predict dose for prostate cancer patients, planned with seven IMRT fields using contours of the planning target volume and organs at risk[22]. The U-Net architecture consisted of seven blocks of both contraction path and expansion path and a bottle-neck at the very bottom part. Below is the Figure (2.17) of the U-Net architecture used:

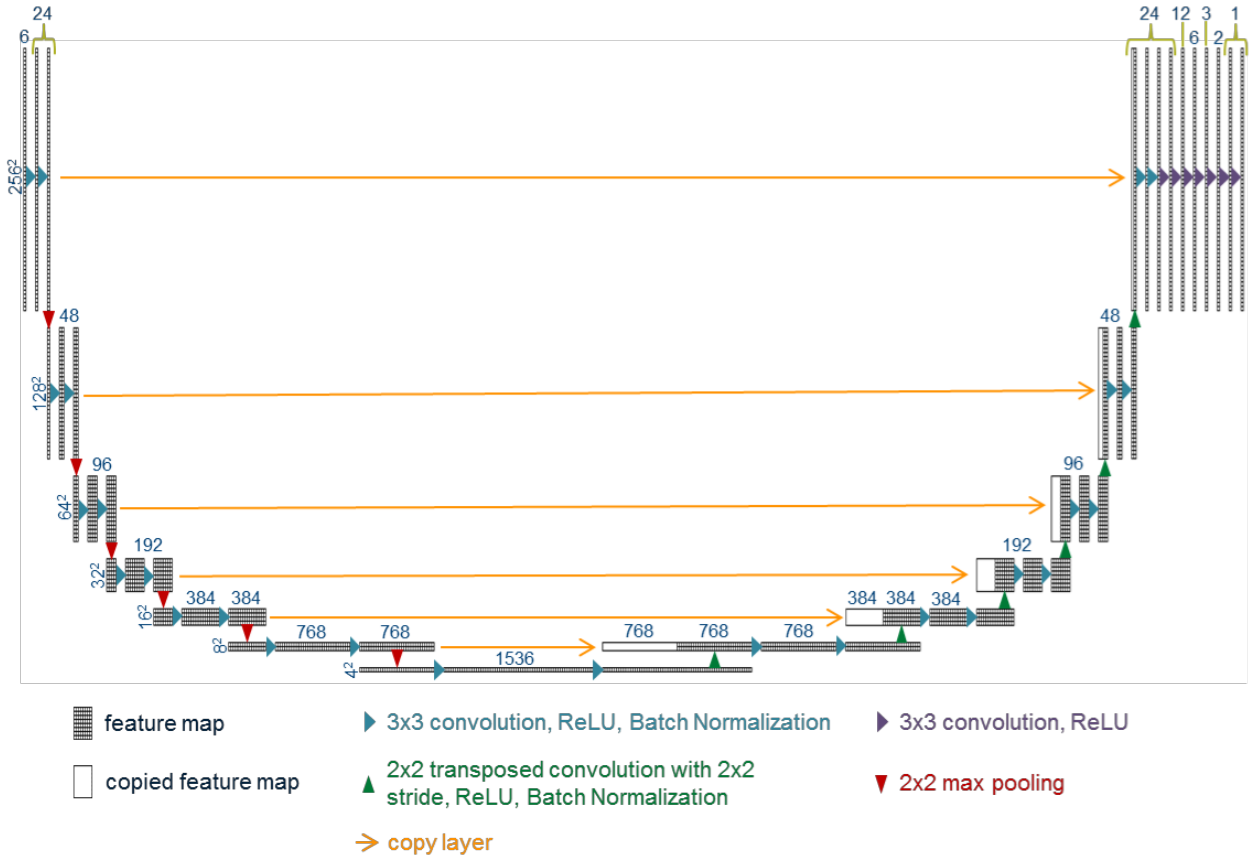


Figure 2.17: U-Net architecture used for dose prediction for IMRT of prostate cancer[22].

The input image consisted of six contours (planning target volume, bladder, body, left femoral head, right femoral head and rectum) of each patient. Single slices of each contour were used to train the model. The size of input image was (256X256) with one channel per contour for a total of 6 channels. Another important aspect is the block of seven convolutions. The operation of seven block of two (3X3) convolutions and (2X2) max pooling allows the reduction of input image to (4X4). The reduced (4X4) image at the bottleneck allows two (3X3) convolution to attain the knowledge of connection between the center of the tumor and the edge of the patient's body. The expansion path followed the similar concept as the original U-Net paper[49]. Padding was set to zero to maintain the size of the feature. This process will allow high precision of dose distribution. Dropout, a process in which randomly neurons are not trained, was performed to avoid

overfitting due to large valued dataset and the dropout parameters were chosen such that validation and training loss were close. The training model was implemented in the Keras with Tensorflow framework.

The next step was to train the architecture for the dose prediction of prostate cancer treated using a seven beam IMRT approach. For training and testing purpose, the dataset was divided into three groups, train set, validation set and test set. The training datasets is the one which updates the weight, validation dataset tests how the trained model generalizes the dataset which is not a training set and test dataset is used to perform an unbiased evaluation of the final model. The total of 88 patients, six contours were used for the study. The test set comprised data from patient 1 to 8. For the first run, the validation set comprised data from patient 9 to 16 with remaining data used to train the model. In the second run, the validation set comprised data from patient 17 to 24 with remaining data used to train. Thus a 10 fold validation was performed. . MSE loss function was considered to calculate the loss between the actual dose and predicted dose. Adam optimizer with parameters ($\beta_1 = 0.9, \beta_2 = 0.99$ and $\text{decay} = 0$) was used. With this implementation, Nyugen *et.al*[22] were able to predict the dose distribution for prostate cancer patients within 5% of that obtained clinically.

Chapter 3

Methods

Chapter 3 describes the method use to predict dose distributions given CT images used in radiotherapy treatment planning for advanced stage lung cancer patients. The beginning of the chapter describes the dataset and its processing. The next section gives detail about the training of the model.

3.1 Dataset details:

The data used in this thesis comprises 3D CT images and their corresponding dose grids generated for 22 advanced stage, lung cancer patients at the Juravinski Cancer Centre between March and June of 2020. The CT images were acquired using either the Brilliance Big Bore (Philips N.V., Amsterdam, Netherlands) or the Somatom (Siemens Healthineers AG, Erlangen, Germany) scanners. Slice thickness of 3mm was reconstructed on a (512 X 512) matrix with a pixel size of about 1.17mm. All patients were planned with radical intent to a prescription of 63 Gy in 30 fractions. The plans were developed using the v9.10 Pinnacle Treatment Planning System (Philips N.V., Amsterdam, Netherlands). Four to five coplanar beams were used, predominantly on the ipsilateral side of the target. The beam angles were standard but were modified by the planner to accommodate the anatomy of the patient. The target was contoured using PET and CT images. Very similar dose objectives were used in IMRT optimization although each case was tweaked by the planning team to achieve patient-specific dose goals. Final dose computation was performed with the Adaptive Convolve Algorithm on a dose grid with 2.5 mm voxels. To improve computational efficiency yet maintain some 3D capabilities, the dose prediction model was trained

with 3, 2D slices extracted from each patient's set. The 3 slices were obtained using nearest neighbor interpolation and coincided with the location of the prescription point ± 2 cm in the superior/inferior direction.

The dataset was divided into two groups. The training set was utilized to train the dose prediction model and the validation set was used to test the effectiveness and to quantify the developed model. The data was divided such that 45 image/dose pairs (15 patients) formed the training set and 21 image/dose pairs (7 patients) formed the validation set.

Computational Environment

This work was performed on a Dell laptop equipped with 16 GB of RAM, i7 processor (US Intel Corporation, Santa Clara, California, United States) and a GeForce GTX 1050 Ti graphics card (Nvidia, Santa Clara, USA). Software was developed using the Jupyter environment (Project Jupyter) making use of the PyTorch framework (Facebook's AI research lab, Cupertino, USA).

Preprocessing of datasets

The pixel size of the original CT images was 1.17mm in-plane with 3mm slices. The voxel size in the dose images was 2.5mm in all three dimensions. Each CT slice spanned an area of 60 cm by 60 cm while the dose grids were smaller, corresponding only to the relevant anatomy (e.g. 38 by 20 cm).

Early in the work it became apparent that processing (512 X 512) datasets required significantly more RAM than available in the computational environment. Thus, the CT images were downsampled to a matrix size of (256 X 256) comprising of 2.34 X 2.34 X 3 mm voxels using nearest neighbor interpolation.

U-Net requires that the input and output data be on the same matrix with a 1:1 pixel spatial correspondence. Thus, the dose grids were resampled to match the (256 X 256) CT images. Resampling was performed by computing the world coordinate for every CT pixel (in mm) using:

$$image_{cord} = image_{index} * image_{pixdim} + image_{start}, \quad (3.1)$$

where $image_{index}$ is the pixel number in the CT image, $image_{pixdim}$ is the pixel size of the CT image (mm), and $image_{start}$ is the coordinate of the first pixel in the CT image (mm). This CT pixel coordinate was then converted to a matrix index in the dose grid using:

$$dose_{index} = \frac{image_{cord} - dose_{start}}{dose_{pixdim}}, \quad (3.2)$$

where $dose_{start}$ is the coordinate of the first pixel in the dose image (mm), and $dose_{pixdim}$ is the pixel size in the dose grid (mm).

Finally, bilinear interpolation is performed in the original dose grid to obtain the dose at the CT image pixel location. Bilinear interpolation involves computing the weighted average of the intensities at the four nearest neighbors. As shown in Figure (3.1), if the four nearest neighbors co-ordinates are (x_1, y_1), (x_2, y_1), (x_1, y_2) and (x_2, y_2) and the unknown pixel is at (x, y), then the intensity at (x,y) is determined using equation (3.3).

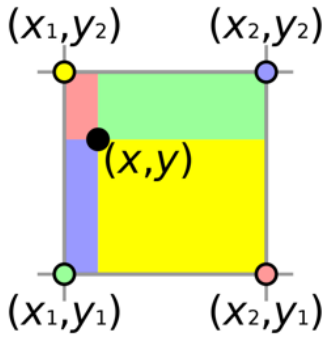


Figure 3.1: Four neighboring pixels of (x,y) pixel.

$$I(x,y) = \frac{I}{(x_2-x_1)(y_2-y_1)}\begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix}\begin{bmatrix} I(x_1,y_1) & I(x_1,y_2) \\ I(x_2,y_1) & I(x_2,y_2) \end{bmatrix}\begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix} \quad (3.3)$$

The last step was to further reduce the image size by removing the outer border of the image as it was normally filled with zero valued pixels. This was done by removing 32 pixels around the CT images and the resampled dose grids to obtain a final data matrix size of (192 X 192).

Data standardization

Standardization in this case refers to rescaling of data intensities across the available cases. Standardization of data is used during DL to reduce the impact of variability in specific cases on the final model[51]. An often used method attempts to rescale the pixel values to achieve a zero mean and unit standard deviation across all cases. In a regular analysis, each pixel intensity would thus be standardized by subtracting the mean dividing by the standard deviation. However, since the data was fed in batches in this work, the regular method would only compute the mean and standard deviation for one batch. This would have to be saved and re-applied to the data before training the model. To avoid this, the running mean and standard deviation[52] may be updated recursively as the data are loaded.

Mean: $\mu_{mean} = \frac{m}{n+m}\mu_{old} + \frac{n}{n+m}\mu_{new}$

Standard deviation: $\sigma_{stdv} = \frac{m}{n+m}\sigma_m^2 + \frac{n}{n+m}\sigma_n^2 + \frac{mn}{(m+n)^2}(\mu_{old} - \mu_{new})^2$

where m is the number of previous observations, n is the number of new observations, μ_{old} and μ_{new} are the means for the old and new observations, respectively, and σ_m and σ_n are the standard deviations for the old and new observations, respectively.

Once all the data were loaded and the mean and standard deviation computed as shown above, each image or dose grid intensity was modified by subtracting the mean and dividing by the standard deviation. The mean and standard deviation were stored in a file for later recovery of the standardized data into their original form.

3.2 Training methods

Model parameters

The U-Net model was trained to predict the dose distribution given the CT image. The U-Net architecture code was downloaded from github.com[53] and implemented as a class in the Jupyter application. Jupyter is a web developer tool used to access Python and PyTorch environments. The Anaconda[54] package manager was used to install Python 3.7.3, PyTorch 1.3.0 and Jupyter. The gold standard dose distribution for each CT image was the resampled dose grid from treatment planning. U-Net has the following hyperparameters: in_channels, n_classes, depth, wf and padding. In_channels defines the number of input channels for the input image, n_classes is the number of channels in the output, depth defines the level of feature extraction, wf is the number of filters applied and padding ensures that the matrix size in the input is maintained in the output. All convolutions are performed using a (3 X 3) kernel. The two most important parameters in U-Net are depth and wf. Both parameters control the architecture of U-Net. The value of depth determines how many convolutions/deconvolutions will take place that reduce/increase the matrix size of the feature map. Simultaneously, wf controls the number of channels produced/reduced during convolutions/deconvolutions. The number of channels at a particular depth is given by $2^{(wf+depth)}$. For example: at the first depth with wf = 6 the number of channels for feature detection will be $2^{(1+6)} = 128$. Increasing the depth of U-Net allows the extraction of features at

a particular image resolution while increasing wf improves the ability to extract those features. The default values for U-Net parameters were in_channels=1, n_classes=2, wf=5, depth=5, and padding was False. For simplicity, wf will be referred to as kernel in the remainder of the thesis.

For lung dose prediction training, the U-Net model was set with following parameters: n_channels=1 (input is a single channel CT image), n_classes=1 (output is a single channel dose grid) and padding = True (same output matrix as input was desired). The depth and kernel were the primary hyperparameters in the U-Net model which required optimization to obtain the predicted dose. Initially, the optimizer was SGD and the loss function was MSE. The number of parameters and the estimated memory needed to store the parameters were observed form the U-Net model to help decide the experimental parameters. The most obvious benefit of choosing more trainable parameters was the potential to extract more complex features. Hence, initial experimentation evaluated the effect of changing depth and kernel values for a single case. For this case, the predicted dose grids appeared to approach the computed dose after depth=5 and kernel=5. With a depth of 6 and kernel of 6 an even lower loss value was obtained with improved predicted dose appearance compared to the computed. With this knowledge, training of all 45 images was performed. To study the loss value for various kernel and depth values, the depth was initialized at 6 and kernel was changed from 1 to 6. Later to observe the depth effect on the training model, the kernel was kept at 6 and the depth was changed from 1 to 6. The depth and kernel could not be increased beyond 6 due to limited RAM in the computational environment.

The training was performed with different layers of convolutions, transposed convolutions and skip connections throughout the experiment. The contraction and expansion path structure were not changed from the original. Below is the final U-Net architecture.

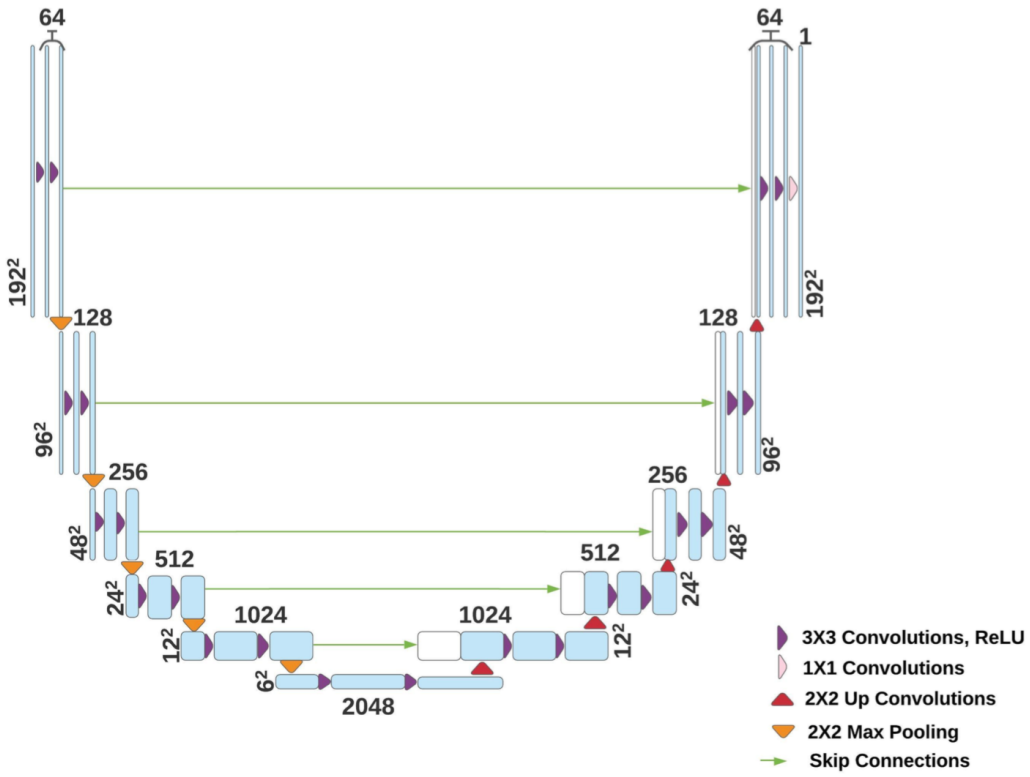


Figure 3.2: U-Net architecture.

Optimizer

The model was trained using SGD, AdaGrad, AdaDelta and ADAM optimizers. The optimizer can be changed using the torch.optim package. Below is the illustration with code to show the implementation of the four optimizers. The hyperparameters of the optimizer have already being explained in chapter 2.

SGD optimizer:

```
optimizer = optim.SGD(unet.parameters(), lr=0.0001)
```

AdaGrad optimizer:

```
optimizer=optim.Adagrad(unet.parameters(), lr=0.0001, lr_decay=0, weight_decay=0, initial _
        accumulato_value=0)
```

AdaDelta optimizer:

```
optimizer=optim.Adadelta(unet.parameters(), lr=0.0001, rho=0.9, eps=1e-06,weight_decay=0)
```

Adam Optimizer:

```
optimizer = optim.Adam(unet.parameters(), betas=(0.9, 0.999),lr=0.0001,weight_decay=0)
```

Loss Function

The MSE loss function was used to compute the loss of the model. The MSE loss was computed between the predicted and the computed dose distributions.

Batch Size:

As discussed in Chapter 2, batch size is also a hyperparameter. Previous studies[55] have found that larger batch sizes converge faster and provide better accuracy. The number of training datasets was 45 but training with such a large batch was impossible due to limited RAM availability. In fact, a batch size of four was the maximum possible so all training was performed with a batch of four.

Model Training

The U-Net model was trained by back propagation method. The following pseudo-code was used for training the model:

```
#Iterate over entire dataset multiple times
for epoch in range(number of iterations):
        # Placeholder for cumulative loss at each epoch
        cumulative_loss = 0.0
        #Iterate over each batch to compute loss
        for i, data in enumerate(train_loader,0):
            # Label the inputs and targets
            inputs, labels = data[0].to(device), data[1].to(device)
            #Clears old gradient from previous step
            optimizer.zero_grad()
            #Labels the output
            outputs = unet(inputs)
            #Loss is calculated between predicted and given labels
            loss = criterion(outputs,labels)
            #Backward propagation is performed
            loss.backward()
            #Parameters update w.r.t current gradient
            optimizer.step()
            #Cumulative loss computation
            cumulative_loss += loss.item()
```

Validity of dose prediction

The accuracy of the predicted dose was assessed visually and quantitatively using gamma analysis. Gamma analysis involves computing the dose difference to compare predicted and target dose distributions. It encompasses discrepancy in both the dose and spatial position of the dose. The minimum value of gamma index,γ[56] is given as

$$\gamma(\vec{r}_r, \vec{r}_e) = min\left(\sqrt{\frac{|\vec{r}_r - \vec{r}_e|^2}{\Delta d^2} + \frac{|D_r(\vec{r}_r) - D_e(\vec{r}_e)|^2}{\Delta D^2}}\right) \quad (3.4)$$

where, $|\vec{r}_r - \vec{r}_e|$ is the distance between the position $\vec{r}_r$ in the reference dose grid and the position $\vec{r}_e$ in the evaluation dose grid; $|D_r(\vec{r}_r) - D_e(\vec{r}_e)|$ is the absolute difference between doses at $\vec{r}_r$ and $\vec{r}_e$; Δd is the distance to agreement criterion, ΔD is the dose difference criterion, and min indicates

that for each reference dose grid location, the location in the evaluation dose grid that results in the minimum gamma is sought.

Gamma analysis was performed using the Python class available on Github[57]. Implementation of this in the computational environment was as simple as including the class in the main code. Gamma evaluation was performed with tolerances of 3mm and 3% of the prescribed dose. Pixels below 20% of the maximum true dose were not analyzed.

Chapter 4

Results and Discussion

Model performance was highly dependent on the choice of hyperparameters. The first three sections of this chapter discuss the performance of the model due to changes in depth, kernel, optimizer and epochs. The chapter ends with results and conclusions.

4.1 Assessment of depth and kernel:

The training of U-Net was performed with different depths, kernels, and optimizers. Table (4.1) and Table (4.2) below show the loss obtained, number of trainable parameters and estimated size of parameters.

Table 4.1: Loss, number of trainable parameters and estimated total size of parameters obtained with kernel 6 and various depths over 1000 epochs

Depth	Loss	No of trainable parameters	Estimated total size of parameters (MB)
1	$5.37X10^{-1}$	$3.76X10^{4}$	36.7
2	$1.83X10^{-1}$	4.0210^{5}	73.9
3	$1.60X10^{-2}$	1.8610^{6}	98.0
4	$6.39X10^{-3}$	$7.70X10^{6}$	129
5	$2.25X10^{-3}$	$3.10X10^{7}$	222
6	$2.00X10^{-3}$	$1.24X10^{8}$	580

Table 4.2: Loss, number of trainable parameters and estimated total size of parameters obtained with depth 6 and various kernel over 500 epochs

Kernel	Loss	No of trainable parameters	Estimated total size of parameters(MB
1	$1.18X10^{-1}$	$1.2\ X10^{5}$	4.19
2	$6.64X10^{-1}$	$4.87\ X10^{5}$	8.89
3	$6.78\ X10^{-1}$	$1.94\ X10^{6}$	21.1
4	$1.31\ X10^{-2}$	$7.78\ X10^{6}$	56.5
5	$7.41\ X10^{-3}$	$3.11\ X10^{7}$	118
6	$3.73\ X10^{-3}$	$1.24\ X10^{8}$	580

The above results confirm that loss decreases as depth and kernel increase. The two experiments were performed with a different number of epochs due to a hardware failure in the laptop's GPU. Training with 1000 epochs was performed before the failure, employing the GPU. After the failure, only 500 epochs were ran on the CPU. The epoch number could not be increased on the CPU because one epoch took about 50 minutes at full depth compared to approximately 10 seconds needed on the GPU. Below Figure (4.1) shows an approximate exponential trend for trainable parameters (and thus, memory required) for computation during model training.

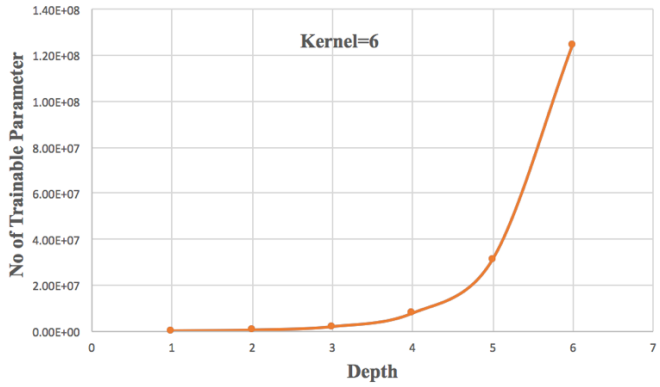


Figure 4.1: Trainable parameter number versus depth for kernel set to 6.

As shown in Figure (4.2), we can conclude that as training with more parameters reduces the loss.

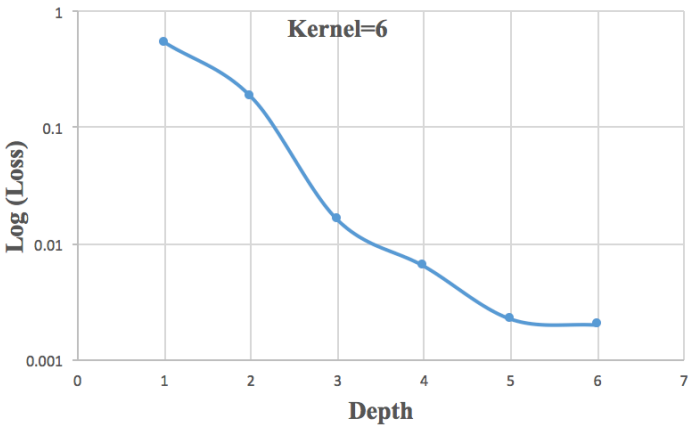
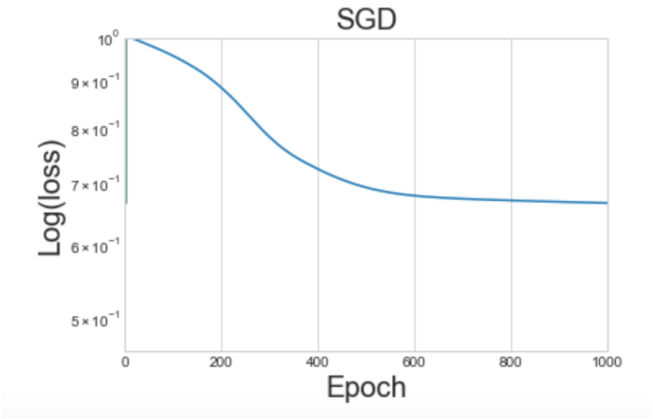


Figure 4.2: Loss vs. depth for kernel set to 6.

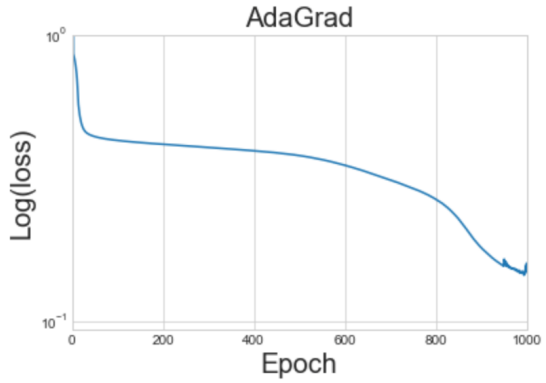
The decrease in loss with increase in kernel and depth means the model is converging and is able to learn from training data. This behavior of model confirms a better performance.

4.2 Assessment of the optimizer:

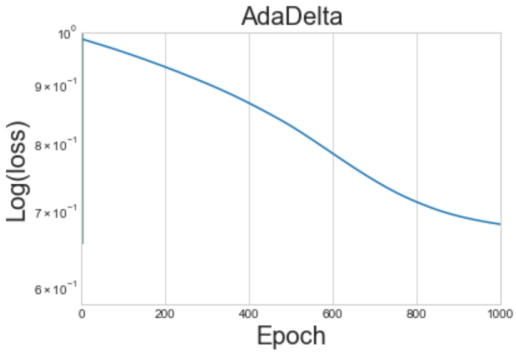
The choice of optimizer is another hyperparameter that affects model convergence. The optimizers used for testing were SGD, AdaGrad, AdaDelta, and Adam. A total of 1000 epochs were ran with each optimizer. Below are the graphs and table of Log(loss) vs. epochs for the four optimizers:



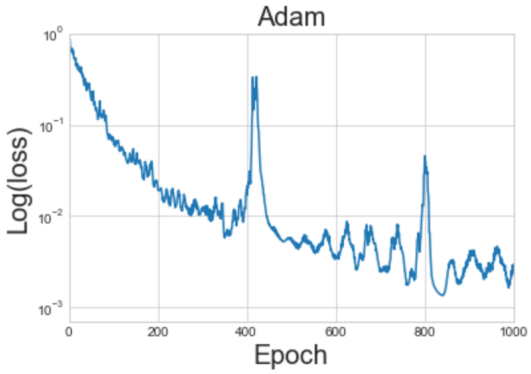
(a) SGD



(b) AdaGrad



(c): AdaDelta



(d) Adam

Figure 4.3: Loss vs. epoch obtained during training with depth 6, kernel 6 with various optimizers

Table 4.3: Loss obtained after 1000 epochs of training with various optimizers and depth 6, kernel 6.

Optimizer	Loss
SGD	$6.65X10^{-1}$
AdaGrad	$1.59X10^{-1}$
AdaDelta	$8.31X10^{-1}$
Adam	$2.93X10^{-3}$

The Adam optimizer achieved the lowest loss. AdaDelta looked like it was continuing to converge after 1000 epochs but another training run with 3000 epochs did not reduce the loss further. From the Figure(4.3) and Table (4.3) it was confirmed that Adam converged the fastest and to the lowest loss value. The noisy trend observed in Adam optimization is discussed in the next section.

4.3 Assessment of epochs:

The Adam optimizer achieved the lowest loss and so was used to assess the training for overfitting and underfitting. The training was performed for depth 6 and kernel 6 for 1000 epochs and the following loss was measured:

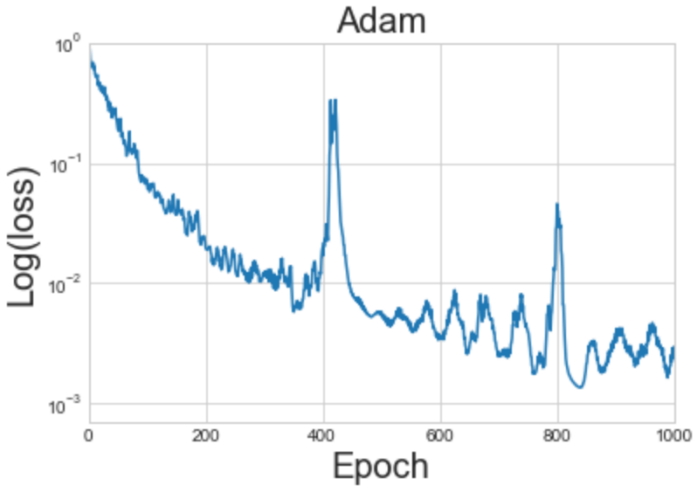


Figure 4.4: Loss observed with Adam over 1000 epochs

The increased loss near epochs 400 and 800 along with the various spikes are caused by the batch size parameter[58]. Jeffrey. M. Ede *et. al* [1], have studied this phenomenon and concluded that low batch number is the primary cause. Given the backpropagation gradient equation, they conclude that a change in loss is directly proportional to a change in parameters. Hence, if a high loss is observed by the optimizer, a large change in parameter is experienced, causing a disturbance in the learning process. At this point, the optimizer again has to re-train the parameter to achieve a better model. This is attained after few iterations of training. The loss then declines after the spike because the parameters are updated again.

4.4 Dose prediction accuracy

Dose prediction was performed using the model trained with depth 6, kernel 6, MSE loss, Adam optimizer, 1000 epochs, and a batch size of 4. While training, it was determined that absolute dose prediction was not possible due to various limitations in the model as discussed in the conclusion section. Hence, the predicted dose was rescaled for gamma analysis to quantify accuracy. For rescaling, pixels where true dose exceeds 50% of the max true dose were located and the average true dose for those pixels was computed. These locations were then used to compute the corresponding average in the predicted dose. Then, the ratio of the average true dose to the average predicted dose was used to rescale the predicted dose before gamma evaluation. Accuracy of the renormalized predicted dose was quantified using gamma analysis with a 3mm, 3% dose tolerance. The gamma maps showed the regions where the model failed. The percentage of pixels passing the dose criteria is given in the table below

Table 4.4: Predicted vs. true dose gamma analysis pass rate for the training dataset.

Patient Number	Gamma Pass Percentage		
	+2cm from prescription position	At prescription position	-2 cm from prescription position
1	97.2	95.7	94.9
2	96.6	97.7	97.6
3	96.3	96.2	96.1
4	95.1	96.7	97.2
5	97.3	95.4	95.7
6	96.9	96.6	96.4
7	98.1	98.6	99.1
8	97.4	97.5	97.5
9	96.9	98.2	97.9
10	99.2	98.9	99.0
11	98.2	98.1	98.3
12	97.9	94.9	96.6
13	98.2	98.2	98.7
14	99.5	98.6	98.7
15	98.2	98.6	99.1
Overall Mean	97.5		
Standard Deviation	1.24		
Min, Max	94.9, 99.5		

From the training dataset result, we observe that the gamma pass percentage was relatively high (mean 97.5% and standard deviation of 1.24%). From this it can be concluded that the training process was successful as an almost perfect match of true dose and predicted dose was observed. Below are the gamma maps for the patients with the highest and lowest gamma pass percentages.

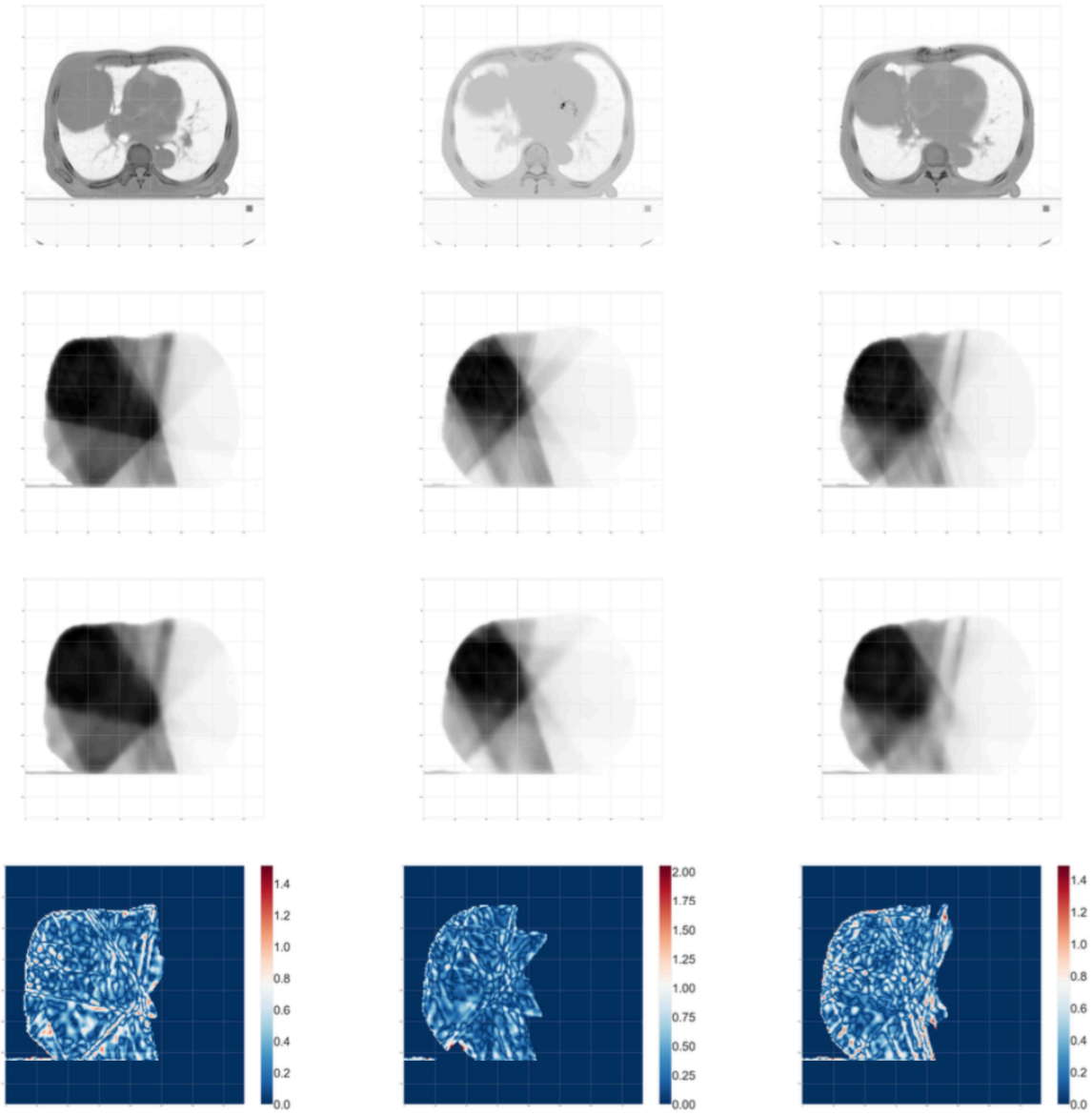


Figure 4.5(a): Gamma map with the highest gamma passing percentage of 99.0±1.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

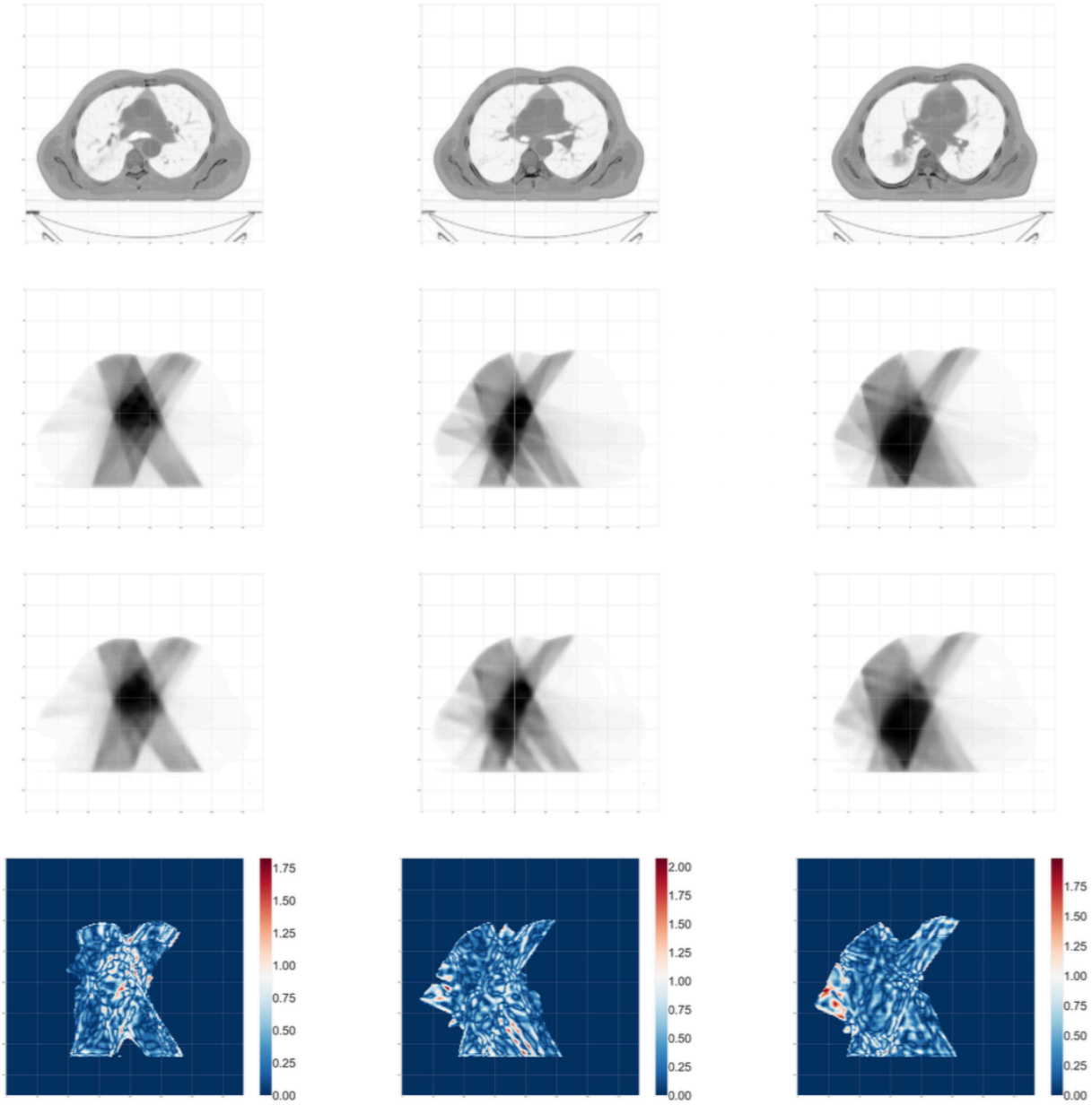


Figure 4.5(b): Gamma map with the lowest gamma passing percentage of 96.6±1.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

From the validation dataset, the predicted dose was not a good representation of the true dose. The table below shows the obtained gamma percentages. Further below are the dose grids and gamma maps.

Table 4.5:Gamma percentage for validation dataset with 3mm, 3% dose tolerance

Patient Number	Gamma Pass Percentage		
	+2cm from prescription position	At prescription position	-2 cm from prescription position
16	11.6	9.86	9.07
17	14.4	14.2	11.9
18	17.7	22.8	16.6
19	19.4	18.6	14.6
20	7.62	4.78	61.4
21	18.9	15.9	15.8
22	18.5	15.1	13.7
Mean	14.2		
Standard Deviation	4.69		
Min,Max	18.9 ,4.78		

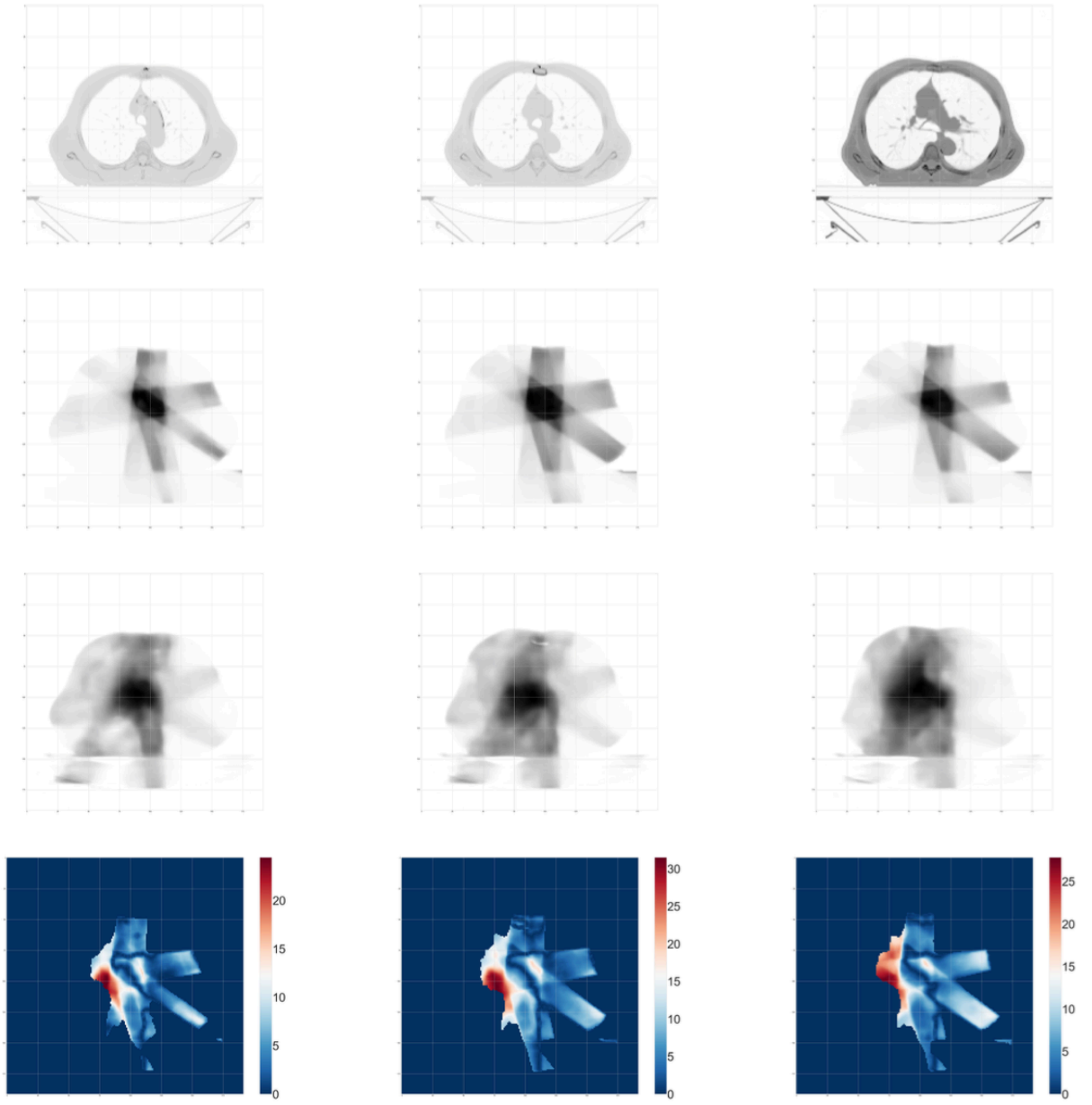


Figure 4.6(a): Gamma map of patient number 16 with gamma passing percentage of 10.2±1.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

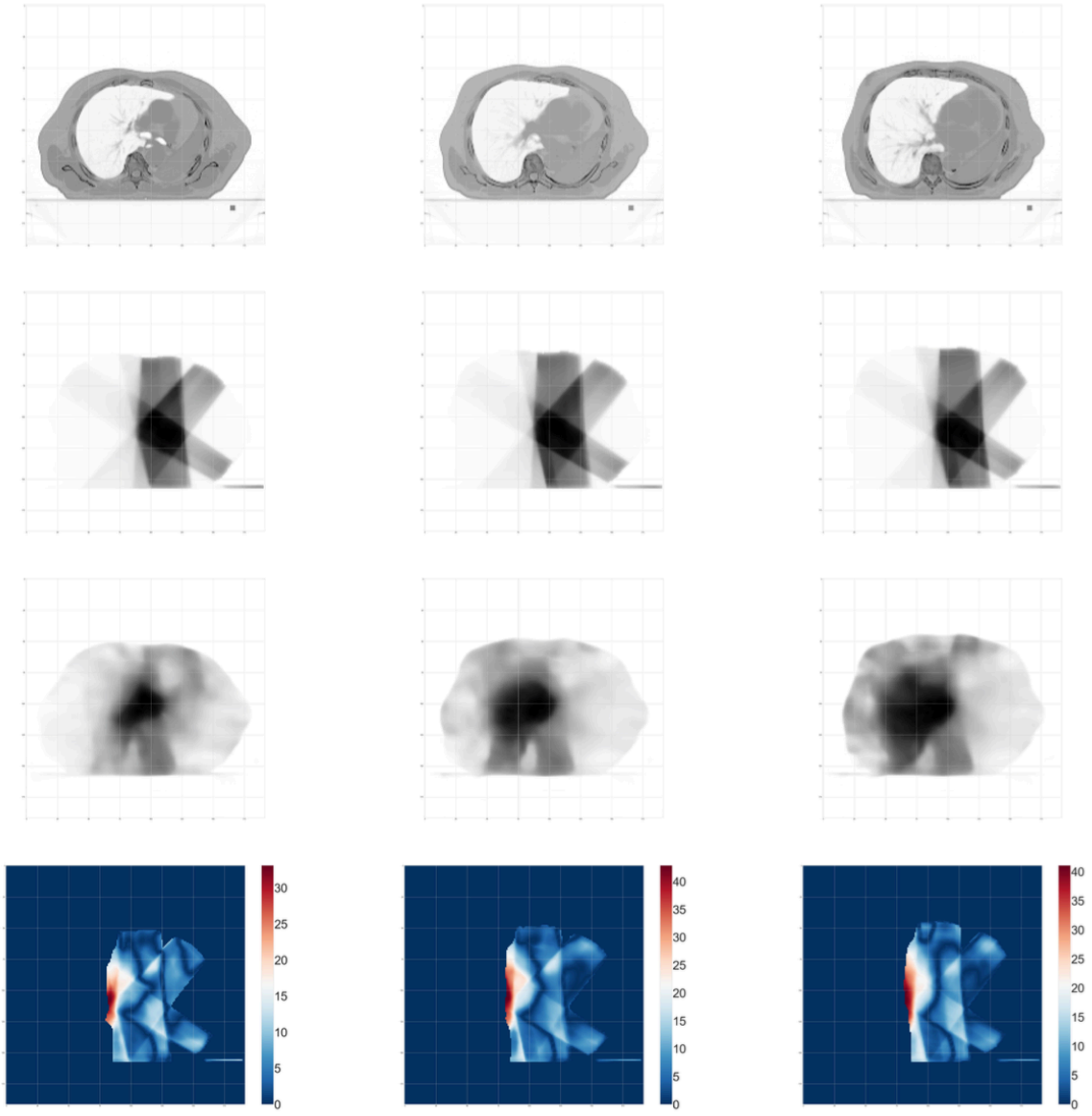


Figure 4.6(b): Gamma map of patient number 17 with gamma passing percentage of 13.5±1.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

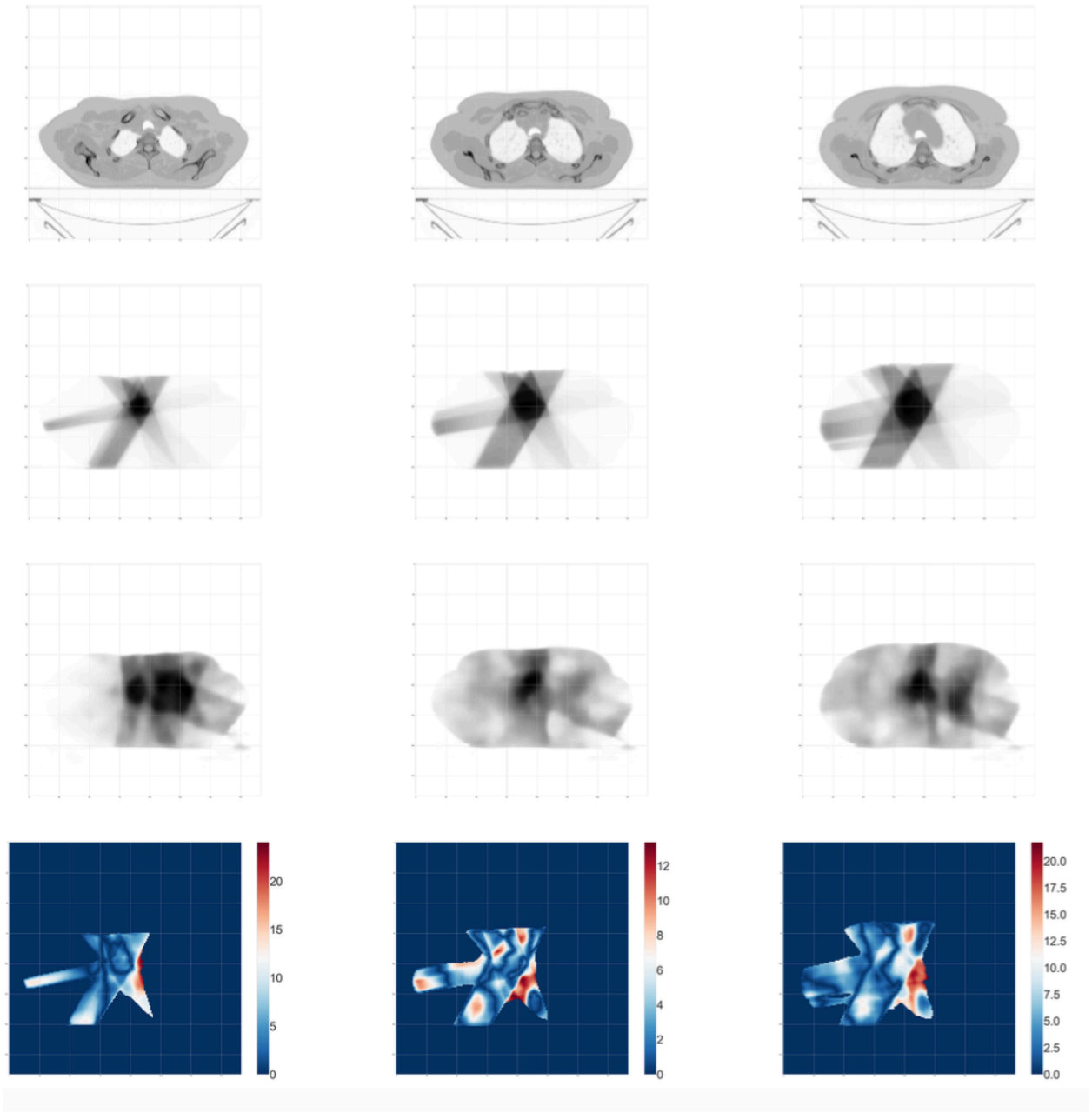


Figure 4.6(c): Gamma map of patient number 18 with gamma passing percentage of 19.0±3.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

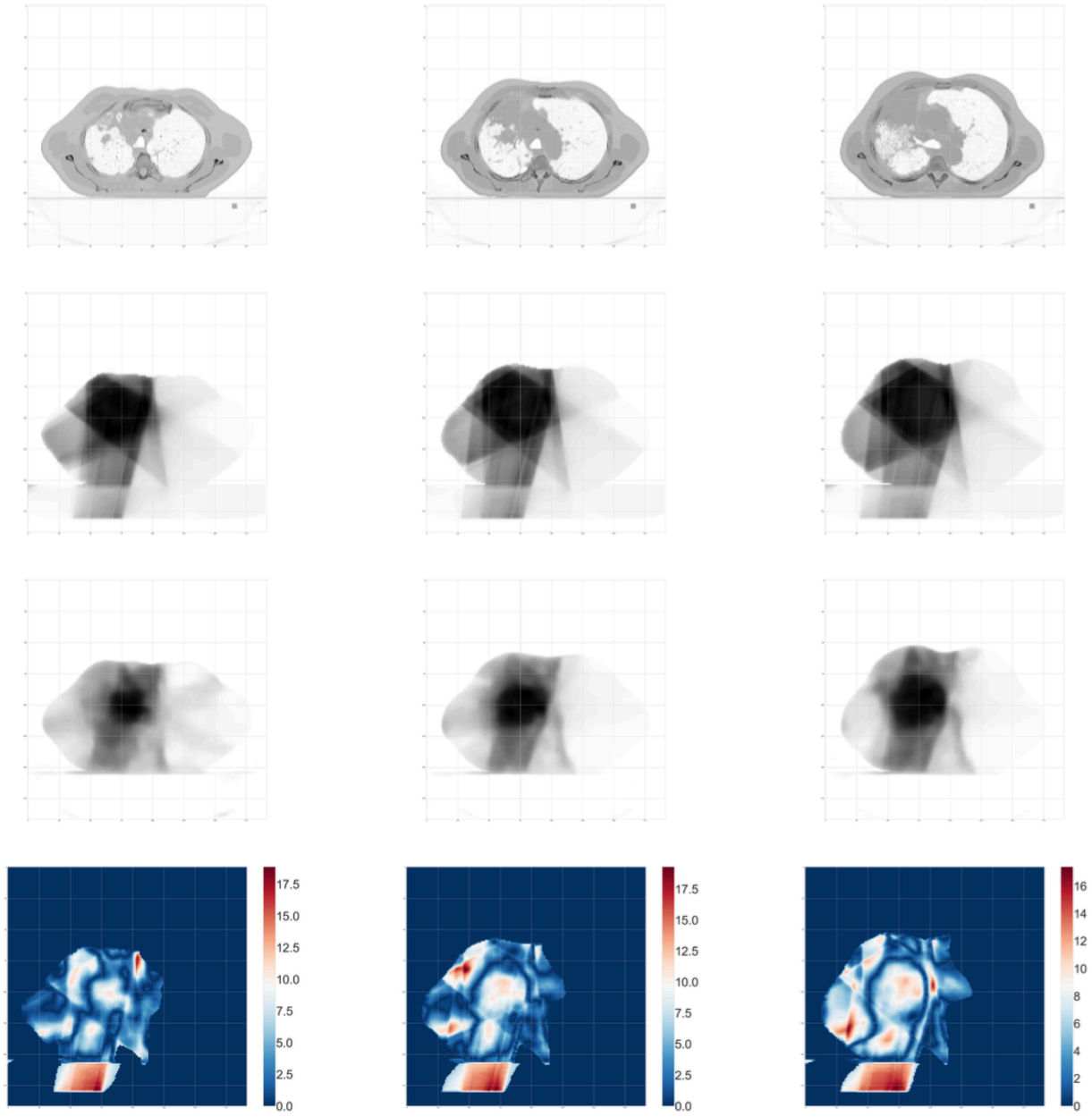


Figure 4.6(d): Gamma map of patient number 19 with gamma passing percentage of 17.5±2.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

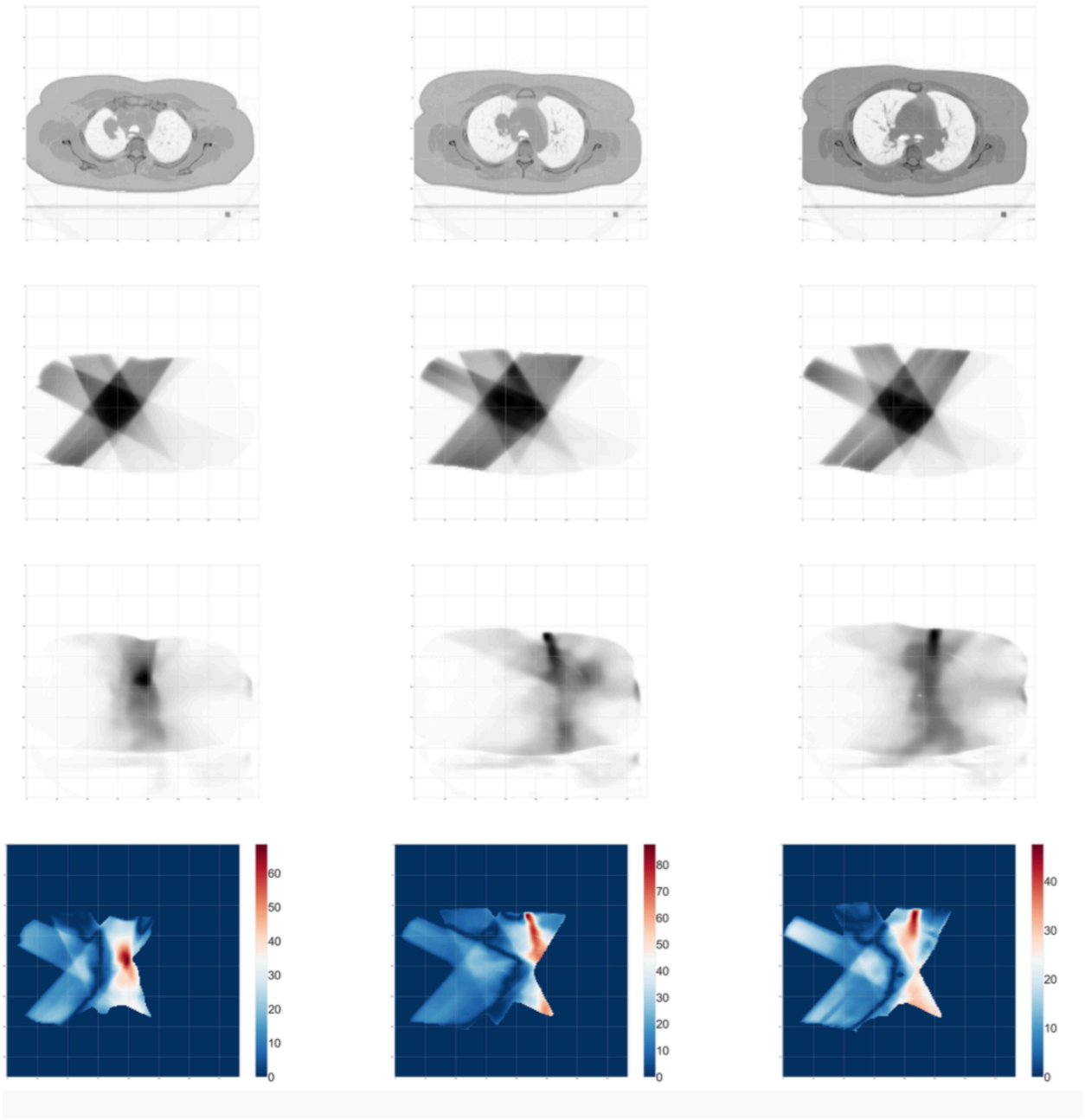


Figure 4.6(e): Gamma map of patient number 20 with gamma passing percentage of 6.18±1.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

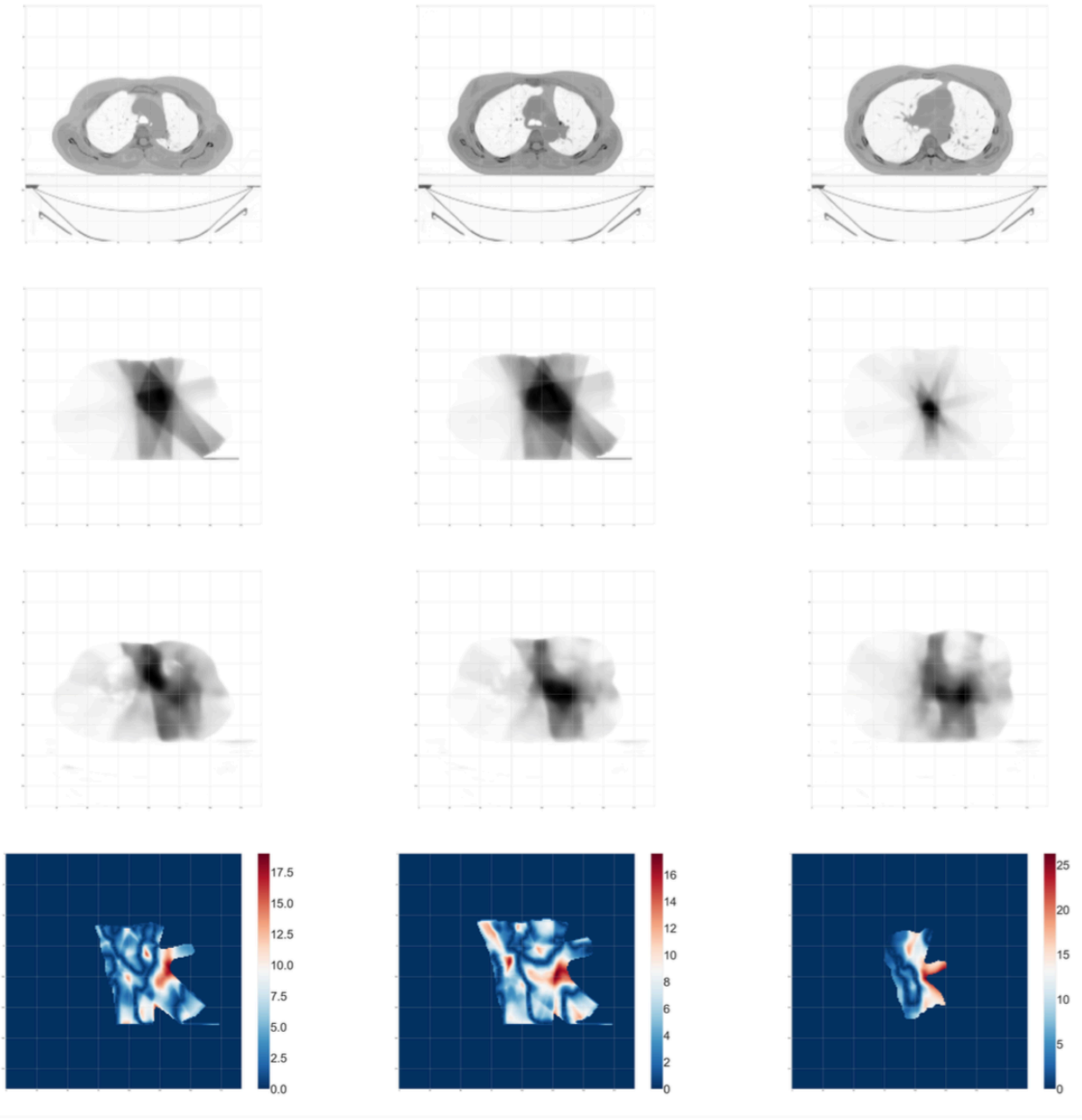


Figure 4.6(f): Gamma map of patient number 21 with gamma passing percentage of 16.8±2.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

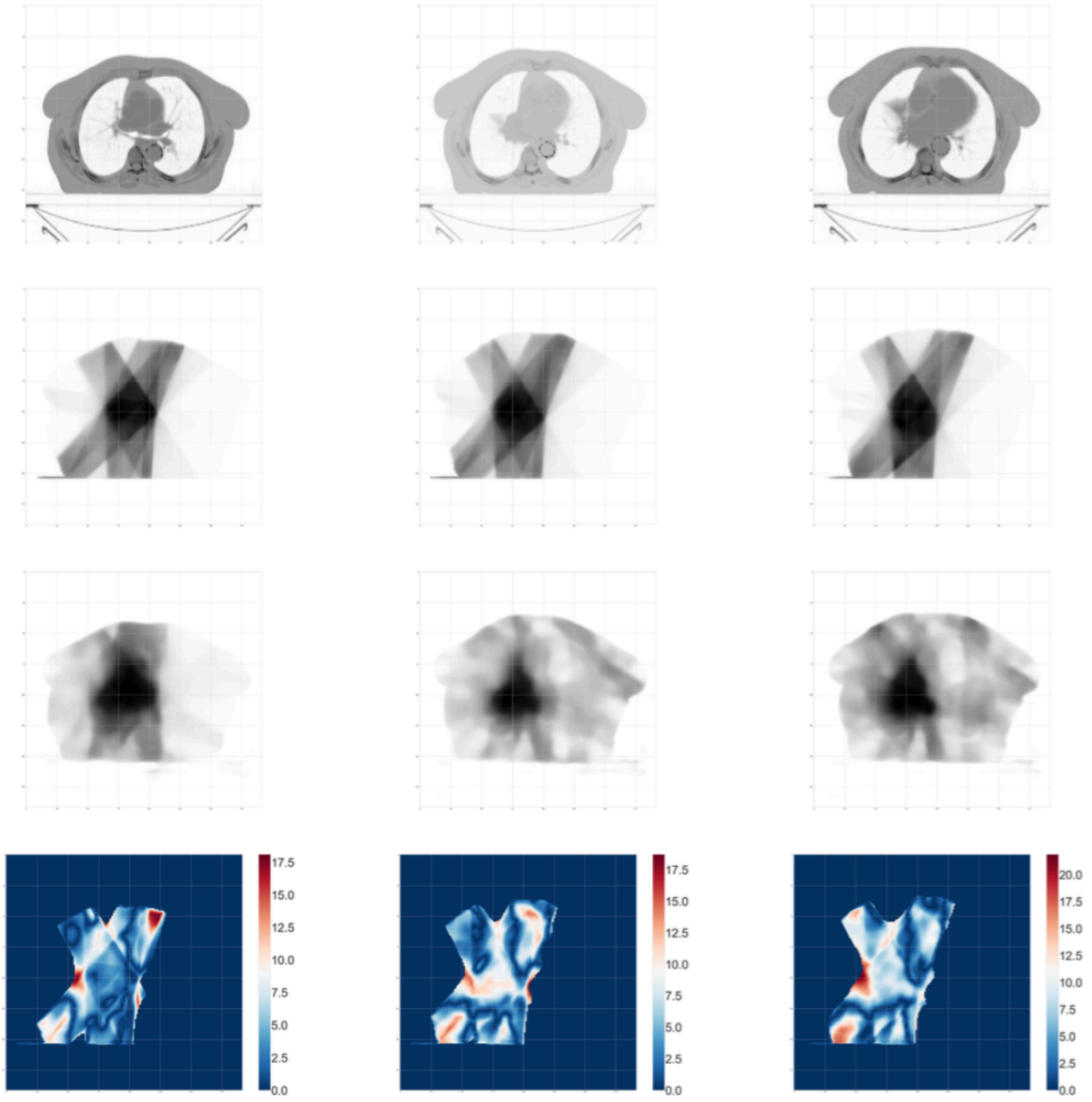


Figure 4.6(g): Gamma map of patient number 22 with gamma passing percentage of 15.7±2.0. The CT scans, true dose, predicted dose and gamma map are in row 1,2,3,4 respectively. The slices of true dose +2cm from prescription position, at prescription position and -2cm from prescription position are in column 1,2 and 3 respectively.

The percentage of pixels passing the dose criteria is very low for the validation dataset. Visually, many hotspots are observed on the gamma map. These are the regions where the dose prediction has failed. From current gamma map, the dose prediction is not acceptable for clinical purpose, however it is promising as the model was able to predict the approximate highest dose region. The main reason could be that more datasets are needed to learn all possible image patterns and generalize for better predictions on new datasets.

To test the the level of discrepancy between the predicted and true dose in the validation set, a gamma analysis with a 5mm, 5% dose tolerance was performed. The table below shows the obtained gamma percentages.

Table 4.6:Gamma percentage for validation dataset with 5mm, 5% dose tolerance

Patient Number	Gamma Pass Percentage		
	+2cm from prescription position	At prescription position	-2 cm from prescription position
16	22.9	21.0	17.2
17	27.4	25.8	24.6
18	32.1	39.8	29.1
19	34.9	32.9	26.9
20	14.4	11.3	13.6
21	34.4	29.7	26.7
22	32.7	26.9	24.9
Mean	26.2		
Standard Deviation	7.47		
Max,Min	11.3,39.9		

From Table (4.6) the percentage of pixels passing the dose criteria for validation is slightly better than the validation dataset with 3mm, 3% dose tolerance. This demonstrates the discrepancy between the predicted and true dose in the validation set is more than at the level of 5% and 5mm.

4.5 Conclusions:

A dose prediction model for treatment planning of advanced stage lung cancer patients was generated using the U-Net architecture. The gamma distribution map was created to observe the regions where dose distributions failed (gamma > 1). With the U-Net architecture with depth=6 and kernel=6, the feature map was reduced from (192X192) to (6X6) which helped extract the low level features. The obtained gamma percentage values on the training set were acceptable, however the gamma map values obtained on the validation dataset were not in agreement within the distance and threshold. Visually we observe a dose distribution very similar to true dose on training set of the data and a relative dose obtained was within the true dose range. However, for validation set, the highest dose region was located, but the overall performance was unacceptable.

There are several reasons for the poor performance of the model with the validation set. The training model was limited to depth/kernel=6. If depth=7 was possible, the feature map would have been reduced to (3X3), which would have allowed extraction of dose distribution at the core level of the true dose, giving a better prediction of the dose distribution. The increase of kernel would have also helped to extract a more accurate dose result. The data was trained only on 45 training samples, representative of 15 patients (3 slices of each image). The number of datasets was very small, limited to what was easily available during the work. It was promising however to see that the model was trained sufficiently to be able to extract accurate dose for the training set itself. It is quite possible that additional training cases would have improved the overall performance of the model. Another limitation was the reduction of the original image data from (512X512) to (192X192) to reduce RAM requirements and improve computational speed. Using the full data may have improved the dose prediction resolution. Another method to improve results

would have been to provide the model with additional input such as contours from the treatment planning system as was done in the work of Nyugen *et. al*[22].

Finally, the dose prediction results may be improved with the use of other network architectures. For example, the GAN (Generative Adversarial Network) has shown outstanding results in producing precise images from arbitrary inputs[59,60,61].

References

[1] High, Peter. "Carnegie Mellon Dean Of Computer Science On The Future Of AI." *Forbes*, Forbes Magazine, 30 Oct. 2017, www.forbes.com/sites/peterhigh/2017/10/30/carnegie-mellon-dean-of-computer-science-on-the-future-of-ai/#343cfb022197.

[2] Brownlee, Jason. "What Is Deep Learning?" *Machine Learning Mastery*, 19 Dec. 2019, machinelearningmastery.com/what-is-deep-learning/.

[3] *Learning Deep Architectures for AI - Université De Montréal*. www.iro.umontreal.ca/~lisa/pointeurs/TR1312.pdf.

[4] Nilsson, Nils J. *Artificial Intelligence: a New Synthesis*. Kaufmann, 2003.

[5] Jackson, Philip C. *Introduction to Artificial Intelligence*. Dover Publications, Inc., 2019.

[6] Aizenberg, Igor N., et al. *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Springer, 2011.

[7] "Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference?" *Sumo Logic*, www.sumologic.com/blog/machine-learning-deep-learning/.

[8] "Neural Network Terminology¶." *Terms*, www.cs.toronto.edu/~lczhang/360/lec/w02/terms.html.

[9] Lecun, Y., et al. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324., doi:10.1109/5.726791.

[10] Krizhevsky, Alex E., et al. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105., doi:http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] "Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)." *ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)*, www.image-net.org/challenges/LSVRC/2012/.

[12] Hao, Karen. "We Analyzed 16,625 Papers to Figure out Where AI Is Headed Next." *MIT Technology Review*, MIT Technology Review, 2 Apr. 2020, www.technologyreview.com/2019/01/25/1436/we-analyzed-16625-papers-to-figure-out-where-ai-is-headed-next/.

[13] *TensorFlow: Large-Scale Machine Learning on Heterogeneous* ...download.tensorflow.org/paper/whitepaper2015.pdf.

[14] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, 'Automatic differerentiation in pytorch', in Conference on Neural Information Processing System, 2017.

[15] Seide, Frank, and Amit Agarwal. "Cntk." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, doi:10.1145/2939672.2945397.

[16] Team, Keras. "Keras Documentation: Developer Guides." *Keras*, keras.io/guides/.

[17]"CUDA C Programming Guide." *NVIDIA Developer Documentation*, docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[18] Stevens, Eli. *Deep Learning with Pytorch*. Oreilly Media, 2019.

[19] "PyTorch Documentation¶." *PyTorch Documentation - PyTorch 1.6.0 Documentation*, pytorch.org/docs/stable/index.html.

[20] Roach, Michael C et al. "Optimizing radiation dose and fractionation for the definitive treatment of locally advanced non-small cell lung cancer." *Journal of thoracic disease* vol. 10,Suppl 21 (2018): S2465-S2473. doi:10.21037/jtd.2018.01.153

[21] Diamant, A., Chatterjee, A., Vallières, M. *et al.* Deep learning in head & neck cancer outcome prediction. *Sci Rep* **9,** 2764 (2019). https://doi.org/10.1038/s41598-019-39206-1

[22] Nguyen, Dan, et al. "A Feasibility Study for Predicting Optimal Radiation Therapy Dose Distributions of Prostate Cancer Patients from Patient Anatomy Using Deep Learning." *Scientific Reports*, vol. 9, no. 1, 2019, doi:10.1038/s41598-018-37741-x.

[23] "Data Science Bowl 2017." *Kaggle*, www.kaggle.com/c/data-science-bowl-2017/overview/description.

[24] Li, Yun et al. *Beijing da xue xue bao. Yi xue ban = Journal of Peking University. Health sciences* vol. 43,3 (2011): 450-4.

[25] *Forecasting Lung Cancer Diagnoses with Deep Learning*. raw.githubusercontent.com/dhammack/DSB2017/master/dsb_2017_daniel_hammack.pdf.

[26] Nielsen, Michael A. "Neural Networks and Deep Learning." *Neural Networks and Deep Learning*, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/.

[27] Haykin, Simon S. *Neural Networks and Learning Machines*. Pearson, 2016.

[28] Lecun, Y., et al. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324., doi:10.1109/5.726791.

[29] Goodfellow, Ian, et al. *Deep Learning*. MIT Press, 2016. url{http://www.deeplearningbook.org

[30] Islam, M. M. Manjurul, and Jong-Myon Kim. "Vision-Based Autonomous Crack Detection of Concrete Structures Using a Fully Convolutional Encoder–Decoder Network." *Sensors*, vol. 19, no. 19, 2019, p. 4251., doi:10.3390/s19194251.

[31] "Conv2d¶." *Conv2d - PyTorch 1.6.0 Documentation*, pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

[32] "Building a Neural Network." *Data Warehousing, BI and Data Science*, 13 July 2020, dwbi1.wordpress.com/2018/02/07/building-a-neural-network/.

[33] Nwankpa, Chigozie et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning." *ArXiv* abs/1811.03378 (2018): n. pag.

[34] *Recti Er Nonlinearities Improve Neural Network Acoustic Models*. ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[35] "Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer." *Algorithms*, vol. 11, no. 3, 2018, p. 28., doi:10.3390/a11030028.

[36] "Softmax Function." *DeepAI*, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[37]Pokhrel, Sabina. "Beginners Guide to Understanding Convolutional Neural Networks." *Medium*, Towards Data Science, 20 Sept. 2019, towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural- networks-ae9ed58bb17d.

[38] Mahmood, Hamza. "Gradient Descent." *Medium*, Towards Data Science, 3 Jan. 2019, towardsdatascience.com/gradient-descent-3a7db7520711.

[39] "Gradient Descent¶." *Gradient Descent - ML Glossary Documentation*, ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html.

[40] Duchi, J, et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2121–2159.

[41] Zeiler, D. Matthew. "Adadelta: An adaptive learning rate method." *CoRR*, vol. 1212, no. 5701, Dec. 2012, pp. 5701–5707.

[42] Hinton, Geoffrey, et al. *Http://Www.cs.toronto.edu/~Tijmen/csc321/Slides/lecture_slides_lec6.Pdf*.

[43] Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *ArXiv.org*, 30 Jan. 2017, arxiv.org/abs/1412.6980.

[44] Sebastian Ruder. "An Overview of Gradient Descent Optimization Algorithms." *Sebastian Ruder*, Sebastian Ruder, 20 Mar. 2020, ruder.io/optimizing-gradient-descent/.

[45] TORRES.AI, Jordi. "Learning Process of a Neural Network." *Medium*, Towards Data Science, 28 Apr. 2020, towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7.

[46] Ioannou, Yani. "Backpropagation Derivation - Delta Rule." *A Shallow Blog about Deep Learning*, 16 Mar. 2018, blog.yani.io/deltarule/.

[47] Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[48] Chazareix, Arnault. "About Convolutional Layer and Convolution Kernel." *Sicara*, www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel.

[49] Ronneberger, Olaf, et al. "U-Net: Convolutional Networks for Biomedical Image Segmentation." *Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015, pp. 234–241., doi:10.1007/978-3-319-24574-4_28.

[50] "ISBI Challenge: Segmentation of Neuronal Structures in EM Stacks." *About the 2D EM Segmentation Challenge | ISBI Challenge: Segmentation of Neuronal Structures in EM Stacks*, brainiac2.mit.edu/isbi_challenge/.

[51] Gal, Michal and Rubinfeld, Daniel L., Data Standardization (June 2019). 94 NYU Law Review (2019) Forthcoming, NYU Law and Economics Research Paper No. 19-17, Available at SSRN: https://ssrn.com/abstract=3326377 or http://dx.doi.org/10.2139/ssrn.3326377

[52] Hancock, Matt. *Batch Updates for Simple Statistics*, notmatthancock.github.io/2017/03/23/simple-batch-stat-updates.html.

[53] Jvanvugt. "Jvanvugt/Pytorch-Unet." *GitHub*, 18 Apr. 2019, github.com/jvanvugt/pytorch-unet/blob/master/unet.py.

[54] "Individual Edition." *Anaconda*, www.anaconda.com/products/individual.

[55] Liu, Rui, et al. "Understanding and Optimizing Packed Neural Network Training for Hyper-Parameter Tuning." *NASA/ADS*, ui.adsabs.harvard.edu/abs/2020arXiv200202885L/abstract

[56] Low, Daniel A., and James F. Dempsey. "Evaluation of the Gamma Dose Distribution Comparison Method." *Medical Physics*, vol. 30, no. 9, 2003, pp. 2455–2464., doi:10.1118/1.1598711.

[57] Christopherpoole. "Christopherpoole/Pygamma." *GitHub*, github.com/christopherpoole/pygamma.

[58]Ede, Jeffrey M, and Richard Beanland. "Adaptive Learning Rate Clipping Stabilizes Learning." *Machine Learning: Science and Technology*, vol. 1, no. 1, 2020, p. 015011., doi:10.1088/2632-2153/ab81e2.

[59] Rmahmood. "Automated Treatment Planning in Radiation Therapy Using Generative Adversarial Networks." *GroundAI*, GroundAI, 17 July 2018, www.groundai.com/project/automated-treatment-planning-in-radiation-therapy-using-generative-adversarial-networks/1.

[60] Kearney, Vasant, et al. "DoseGAN: a Generative Adversarial Network for Synthetic Dose Prediction Using Attention-Gated Discrimination and Generation." *Scientific Reports*, vol. 10, no. 1, 2020, doi:10.1038/s41598-020-68062-7.

[61] Babier, Aaron, et al. "Knowledge-Based Automated Planning with Three-Dimensional Generative Adversarial Networks." *Medical Physics*, vol. 47, no. 2, 2019, pp. 297–306., doi:10.1002/mp.13896.