SEGMENTATION OF THE GASTROINTESTINAL TRACT MRI USING DEEP LEARNING

J. Roy and A. Abdel-Dayem

Bahrti School of Engineering, Laurentian University, Sudbury, Ontario, Canada.

ABSTRACT

This paper proposes a deep learning-based model to segment gastrointestinal tract (GI) magnetic resonance images (MRI). The application of this model will be useful in potentially accelerating treatment times and possibly improve the quality of the treatments for the patients who must undergo radiation treatments in cancer centers. The proposed model employs the U-net architecture, which provides outstanding overall performance in medical image segmentation tasks. The model that was developed through this project has a score of 81.86% using a combination of the dice coefficient and the Hausdorff distance measures, rendering it highly accurate in segmenting and contouring organs in the gastrointestinal system.

KEYWORDS

Machine Learning, Cancer Diagnosis, Supervised Learning, Computer Vision, Semantic Segmentation.

1. INTRODUCTION

Cancer diagnosis and treatment is a challenging task for many healthcare professionals. Millions of people every year undergo many forms of treatments to help reduce the symptoms of the specific cancer they have, and in some cases, they are able to overcome the sickness altogether. A common form of cancer treatment is radiation therapy, where the patient receives radiation through a beam that targets the cancerous cells, while unfortunately also affecting a certain number of healthy cells around the cancer. Radiation oncologists and therapists make use of a highly advanced imaging system, Magnetic Resonance Imaging Guided Linear Accelerator (MR-Linac) [1], to get clear and accurate images of the position of the patient's organs and tumor. The imaging is done daily, and they must manually outline the organs in the dosage area so that they can plan and compute an effective angle and dosage of radiation to deliver to the tumor. This treatment therefore requires careful planning and organization to ensure the radiation can be safely and consistently delivered to the patient on a regular basis .[2]

It is estimated that 5 million people worldwide were diagnosed with cancer of the gastro-intestinal tract and of those 5 million, approximately half were treated with radiation treatments [3]. Radiation treatments of the GI tract require daily radiation doses from 1 – 6 weeks using the MRLinac scanner to scan slices of the patient's organs to plan an effective incident angle for the radiation (Figure 1).

There is always a possibility of the therapists making small errors when making these outlines, which can lead to radiation doses being delivered to healthy cells accidentally. Also, outlining each organ in each slice can be very time consuming and can generally add between 30 to 45 minutes to a patient's treatment, since the patients get scanned in multiple slices to simulate a 3-dimensional scan of their organs [4]. Due to the amount of time that it takes to take all these

DOI:10.5121/ijaia.2023.14404

scans and then make all these outlines, there is a higher probability that the patient waiting on the treatment table will move or slightly shift to make themselves more comfortable. Even slight movements from the patient can cause the organs to shift, which creates inaccuracies in the outlines made by the therapists. With these increased sources of error, the patient is more likely to experience side effects from the treatments and in some extreme cases, it can cause fatalities in patients that could have had a chance of surviving otherwise.

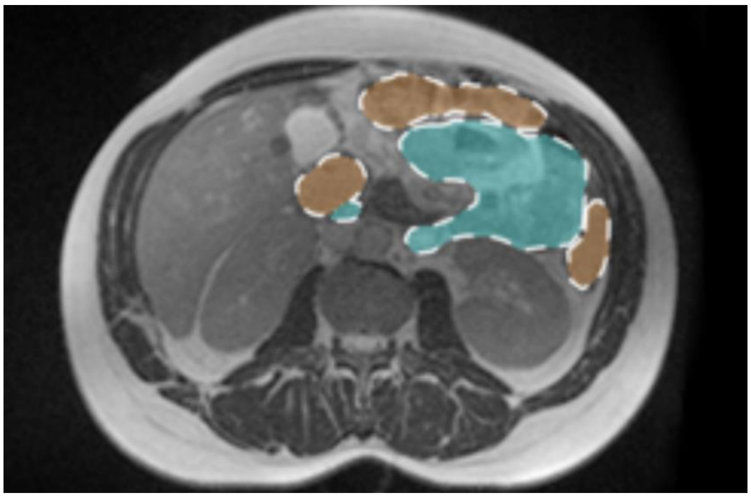

Figure 1. Segmentation of an MR-Linac scan.

Implementing a deep learning model to be able to make these outlines automatically can be a good idea to help radiation therapists deliver a more effective treatment. A computer can segment the organs much quicker and possibly more accurately, which would decrease treatment times for these patients. This could also mean that the therapists would be able to treat more patients per day, decreasing the waitlist for people who are waiting to get radiation treatments. It would also allow for more accurate treatments, meaning the dosage can be higher on the tumors themselves, while limiting the exposure to other healthy cells around the cancer. These benefits add up quickly, and can make for better cancer control, fewer side effects caused by radiation, shorter treatment times and an increase in the number of planned daily treatments for the radiation clinic. This paper proposed a deep learning-based system to analyze and segment the gastrointestinal tract magnetic resonance images (GI MRI). The proposed system focuses on cancers involving the gastrointestinal (GI) tract and aims to provide a second opinion that could help radiologists and therapists complete their duties more effectively. It automates the process of segmenting the organs to help radiation therapists contour organs of interest quickly and accurately.

The rest of this paper is organized as follows. Section 2 offers a preliminary introduction to machine learning principles and deep learning challenges. Section 3 presents the common network training and optimization techniques. Section 4 and Section 5 present the experimental setup and the obtained results, respectively. Finally, Section 6 offers the conclusions of this paper.

2. BACKGROUND

2.1. Machine Learning Basics

Machine learning (ML) is a concept that encompasses many applications which include deep learning. A typical program involves explicit rules and instructions for the computer to follow in order to complete a task. ML can replace traditional programming where the data is too complex to be able to process using explicit rules. ML is the science of programming a computer to be able to look at the data, learn from it using statistical models and make predictions on new unseen data. This technique makes for an easier program to understand as well as to maintain for highly complex tasks [5][6].[7[

There are four types of learning models: supervised learning, unsupervised learning, semi-supervised learning, and reinforced learning. Supervised learning requires a set of data to come with the desired outputs (labels) so that a model can learn what the output of each instance should be. After training the model on the data with their labels, the model can then make predictions on a separate dataset called the test set to evaluate its performance. Unsupervised Learning is a form of learning without the use of labels. The model must learn independently the patterns of the data it is working with. There are, however, times where the data is mostly unlabeled with the exception of a few labeled instances. This is where semi-supervised learning techniques can be used to separate data into groups with similar features and inherit the labels of neighboring instances. Reinforced Learning is a form of learning where an “agent” performs actions and gets rewarded for good actions and penalized for bad ones. The agent learns on its own what actions give the greatest rewards. This paper focuses on building a supervised learning model for image segmentation tasks.

2.2. Problems and Challenges

There are many challenges when training models. For example, there is a possibility that the data used to train the model is considered bad data, or that there are not enough instances. Data can be considered bad if it contains many instances that fall outside the norm of the data (outliers), or if the data being used to train the model is not representative of the new data that will be used to test the model. This means that the underlying pattern in the data used for the training set does not coincide with the underlying pattern in the data used for the test set. Common as these issues may be, the most common problems faced when training a model come from overfitting and underfitting data.

Overfitting occurs when the model being used is too complex. The model attempts to find a very complex function to fit the training data perfectly so that it reduces the error for each of the instances’ predictions. The problem with this is that the model creates a very complex prediction function that only matches the training data it sees. So, the function can predict the training data very well, but when new data is presented, the function cannot make accurate predictions due to the overly complex function. For example, a model using a polynomial of degree four would not be able to make accurate predictions on data that behaves quadratically.

Underfitting is essentially the opposite of overfitting. When underestimating the complexity of the data and training a model using a function that is too simple, the model will likely underfit. This is because the model is limited to a certain degree of freedom when the data simply cannot be described with that limited freedom. A good example is a model using a linear function while the pattern within the data behaves like a polynomial function of degree three. Figure 2 demonstrates the difference between overfitting and underfitting.

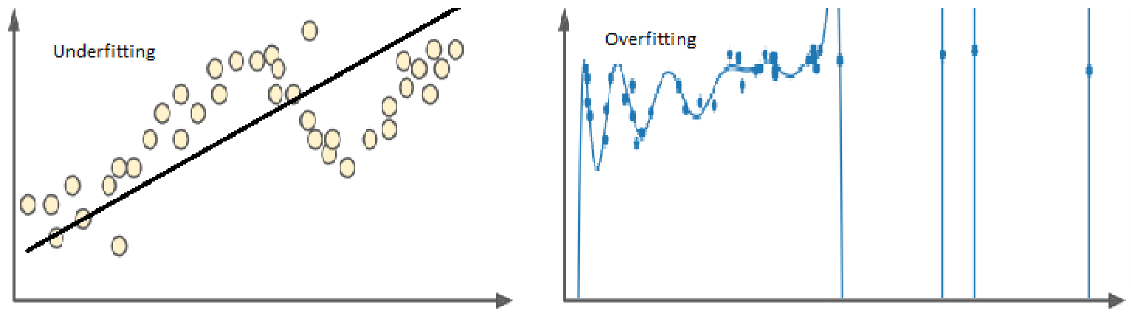

Figure 2. Underfitting vs. Overfitting.

For these problems, training data is separated into two sets. The first set is still considered the training dataset and the second one is considered the validation set, which can be used as a sanity check to ensure that neither overfitting nor underfitting occurs when training a model. If the model performs poorly on both, it is underfitting, whereas if it performs well on the training data but not on the validation data, it is overfitting. Tweaking the model until a balanced performance on both sets are achieved, will likely produce the most consistent and accurate model on the test set.

2.3. Regression vs. Classification

There are two ways a supervised model can make predictions on its data: regression and classification [8]. Regression is a kind of model that uses a set of input features and predicts what the numerical value should be, according to the features. An example of this would be a model that takes in a list of values describing a car's specifications and features to make a prediction on what the value of the car should be. The model takes in the inputs one by one and plugs them into a function, which then calculates the predicted value of the car. The functions used can vary from simple linear equations to complex polynomial functions to be able to detect the pattern within the data.

Classification models can be categorized into 3 different types: binary, multiclass and multilabel. Binary classification tasks require the model to make a simple prediction of whether an instance belongs to the class or not. For example, a model must look at a picture, and identify whether the picture is of an apple or not. Multiclass classification is used when there are many classes to choose from, and the model must predict which one the instance falls under. An example of this type is a model that predicts whether a picture contains an apple, a banana, or an orange. Multilabel classification refers to a classification task that can predict many different classes at the same time. For example, a model looks at a picture of an apple and is able to classify it as an apple, a round object, and a red object.

A supervised machine learning model uses a metric called the loss function to measure the accuracy of its predictions. It then makes corrections to the function to minimize the error between the labels and predictions. This can be done in one step, or many small iterative steps by running the data through multiple times. This is the fundamental way all supervised machine learning models detect patterns within the data and attempts to match the patterns as closely as possible. When using regression models, it is common to use the mean squared error between the predicted value and the label value as its metric for the model's performance. For classification tasks, it is common to use precision, recall, or a combination of both metrics.

2.4. Deep Learning

Deep learning is a subset of machine learning that was inspired by how the human brain works. The model consists of multiple layers of artificial neurons that take input parameters and processes the information layer by layer in order to compute a desired output. Deep learning models are very powerful models capable of learning incredibly complex patterns in datasets that simple machine learning models cannot detect. They are also very versatile, as they can be used for many different applications like tasks that require the use and analysis of written and spoken language, image analysis, text analysis and many more. The following section will introduce the basic concepts that encompass deep learning models.

2.4.1. The Threshold Logic Unit

The threshold logic unit (Figure 3) is the most basic building block of any neural network architecture. The TLU assigns weights to each input and computes the sum of the inputs. It then uses an activation function called the step function capable of outputting a 1 or a 0.

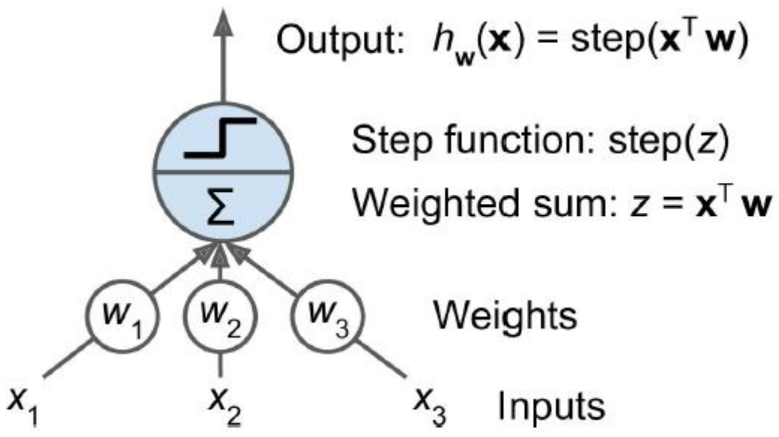
Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{x}^T \mathbf{w})$
Step function: step(z)
Weighted sum: $z = \mathbf{x}^T \mathbf{w}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Figure 3. The threshold logic unit [7].

The TLU can perform linear binary classification by computing the sum of the inputs, and classifying them using a linear separation. It outputs the positive class if it exceeds the threshold or the negative class otherwise. Training a TLU is as simple as adjusting the weights of the inputs.

2.4.2. The Perceptron

The perceptron is composed of a single layer of TLUs with their own weights and step functions, all fully connected to the inputs and the output layer. The model also uses a loss function on its final predictions for training purposes. A fully connected layer is more commonly known as a dense layer. Perceptron models are typically not very powerful as learning models, but they can be stacked in multiple dense layers to build a model capable of learning much more complex tasks. Multi-layer perceptron models (commonly referred to as artificial neural networks) also often contain a bias term to help shift activation functions based on applications and desired range of outputs (Figure 4).

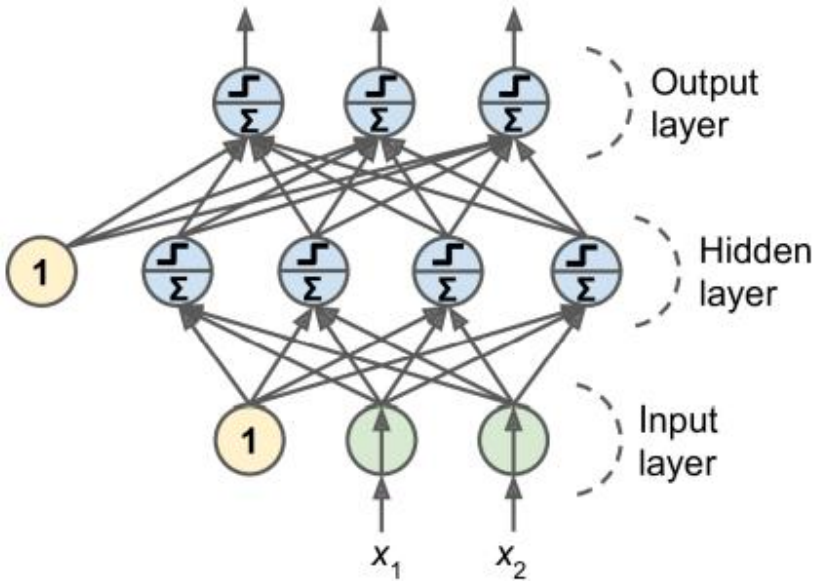

Figure 4. Multi-layer perceptron with bias term [7],

Multi-layer perceptron models (MLP) generally consist of an input layer, one or more dense layers in the middle called hidden layers, and an output layer. Another important distinction between the perceptron and MLPs is that instead of using a step function as the activation function, they generally use a function with a well-defined slope to be able to learn more complex patterns. It is very important to carefully consider the activation function to use for a given model (Figure 5).

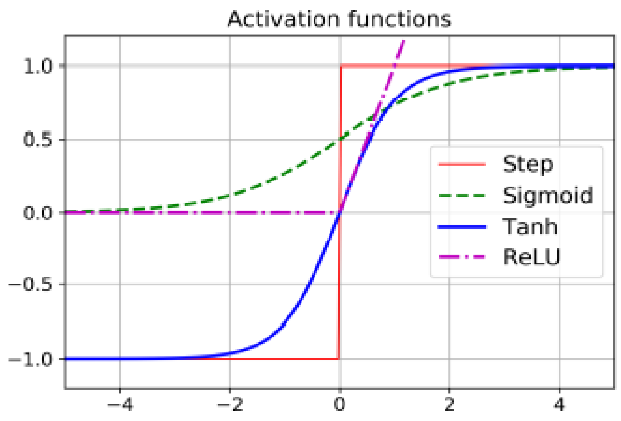

Figure 5. Activation functions for neurons in a neural network.

Choosing the right activation function depends on what kind of model is being trained, and on what kind of application it is being used. Each of these functions uses their own equations to deal with the value obtained from summing all weighted inputs of a neuron.

2.4.3. Deep Learning Networks

When an artificial neural network (ANN) has a deep enough stack of dense layers, it is referred to as a deep neural network (DNN). ANNs and DNNs are trained using a technique referred to as back propagation, which is essentially a gradient descent technique used after running on a subset of the data through the network. To start, the weights of all input connections are initialized randomly. After making the predictions on the subset using the randomized weights, the model can compute the gradient of the model's error with respect to each input weight of each neuron. Using these gradients, the model can compute the adjustments needed for each weight of each neuron moving backwards in the network, layer by layer. Then, it adjusts the input weights of the connections between the neurons accordingly. Also, a learning rate determines the size of each adjustment made in the input weights. This process happens iteratively until it reaches the end of the dataset, which completes an epoch. Then the model starts another epoch and continues improving until the network converges to a solution, or until it reaches the maximum allowed number of epochs.

DNNs can be used for regression, simply by making one output neuron per value to predict. Loss functions used for regression are typically the mean squared error and the mean absolute error, while classification typically uses loss functions like cross-entropy loss. Generally, the output neurons for regression do not use any activation functions so that they can output any range of values. However, the ReLU (rectified linear unit) activation function can be used if the task only requires positive values. The sigmoid activation function is useful for tasks that require only a range of values. A very common loss function used to perform binary and multiclass classification tasks in a DDN is cross-entropy loss function. For a binary classification task, a single output neuron using the logistic function can be used, and for multilabel classification tasks, one neuron per class with the logistic function. For multiclass classification, one neuron per class using the softmax activation function can be used.

2.4.4. Convolutional Neural Networks

Convolutional neural networks (CNN) [9] are common in various applications like voice recognition, natural language processing, and object detection. While DNNs do have the ability to perform incredibly complex tasks, there is a drawback to this kind of model. When implementing a DNN, a single dense layer of size N neurons connected with another dense layer of size M neurons will have N x M connections when fully connected. Building a DNN with multiple dense layers can have a very large computational cost that becomes very computationally expensive for training models. Even a shallow network can have a very large computational cost if the dataset contains a large number of parameters. This is especially true for tasks that involve processing and analyzing images. This is the motivation for creating a different kind of layer that does not need to be fully connected. The neurons in a convolutional network are only connected to a few neighboring neurons within their receptive field, limiting the number of connections necessary between layers. To understand this, it is important to understand what a convolution layer is doing in the background (Figure 6). Convolutional layers initialize an array of small filters and slide each of them throughout the entire image to look for different patterns called feature maps. The filters slide through the image to detect various patterns in the image. It is common to pad the image for the filter to be able to reach the edges of the image without causing border violation issues.

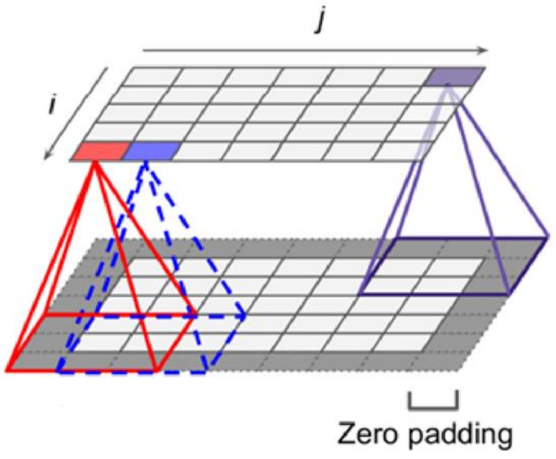

Figure 6. The filters slide through the image to detect various patterns in the image. It is common to pad the image with 0s for the filter to be able to reach the edges of the image without going outside the borders of the image.

The number of filters used will determine the number of output feature maps by the convolution layer. The logic behind this design is that different sections (from left to right) of the layer will be able to detect different patterns in each image. Higher-level layers will only detect simple patterns while the lower-level layers will be able to build on the simple patterns from previous layers' feature maps to detect much more complex patterns. This architecture is often combined with one or more dense layers after the convolution layers, so that the model can combine the information to make a prediction based on these patterns.

2.4.5. The Pooling Layer

A large part of computer vision tasks is handled by convolutional layers, but there is more to this task than simply chaining convolutional layers together. For instance, there are times when the image size is still too large and causes performance problems. In this case, a pooling layer can down-sample the image size by applying filters to the image and only keeping a certain value within the filter (Figure 7).

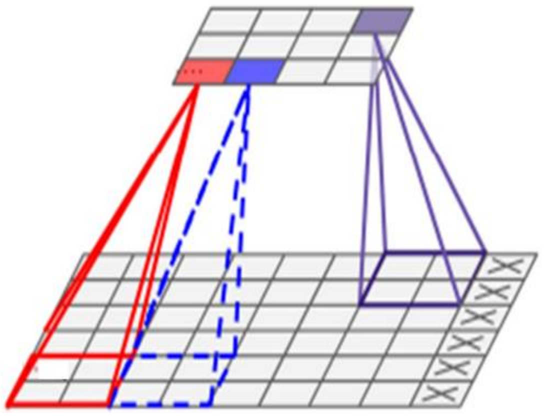

Figure 7. The pooling layer.

It is possible to implement a pooling layer that picks the average value within the filter, the minimum value, maximum value, or other values. Computational performance is not the only advantage of using this layer. Using pooling layers can correct some of the small rotational and positional discrepancies found in the images that may affect the performance of the model. A negative effect of this layer, however, is that a lot of information can be lost since it compresses the information of the image. To counteract this, up sampling can be implemented to bring back some of the lost information when processing the image. This is done using a transposed convolution layer, which stretches the image by adding rows and columns of zeros between each pixel and then applying regular convolution to the image.

2.4.6. Semantic Segmentation and the U-Net Model

Semantic segmentation is a special object detection model that not only detects objects in an image, but also detects the location of the object and outlines it. The U-Net model is a popular semantic segmentation model used in the medical field for segmentation in medical images [10]. The U-Net model is an example of a Fully Convolutional Network (FCN). This means that the model makes use of only convolutional layers to perform the segmentation and does not contain any dense layer within its architecture. For the purposes of this article, the objects to be detected are the large intestine, the small intestine, and the stomach. This will aid the therapists to outline the areas of healthy cells to be avoided by the radiation.

The way U-Nets work is through a series of multiple encoder blocks followed by a series of decoder blocks (Figure 8). There is an equal number of encoder blocks and decoder blocks, and they are organized in a symmetric fashion. The encoder blocks consist of two consecutive 3 x 3 convolution layers followed by a 2 x 2 max pooling layer. The decoder blocks consist of a 2 x 2 transpose convolution layer followed by another two 3 x 3 convolution layers.

The model starts with four encoder blocks that down-samples the image, with the number of filters in the convolution layers doubling for every block. This is where the model extracts spatial features in each color channel in order to detect the objects or patterns in the image. Then the model performs a double 3 x 3 convolution to bridge the encoder blocks to the decoder blocks, where the model up-samples the image and constructs the segmentation mask. There are four decoder blocks and for each block, the filters are halved. The final decoder block uses the same number of filters as the initial encoder block, and it is generally followed by a 1x1 convolution layer with a sigmoid function for the output [11].

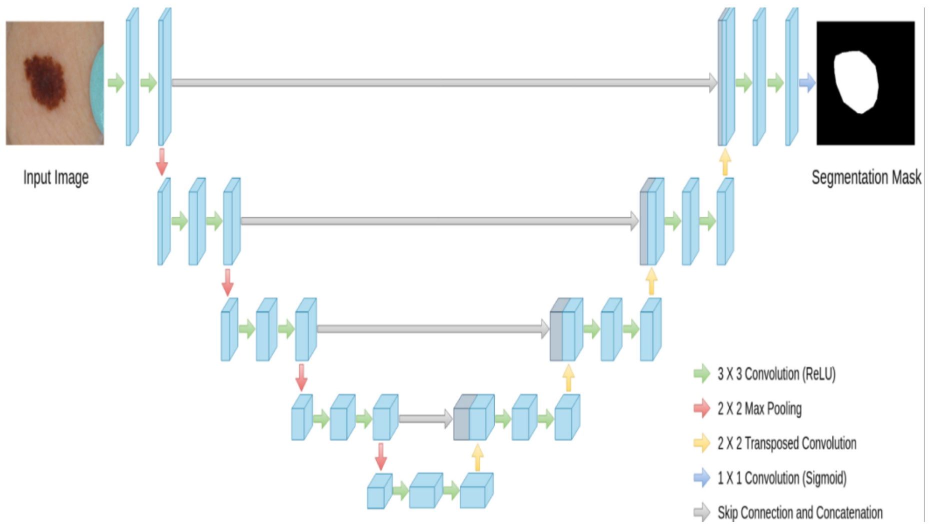
Input Image
Segmentation Mask
3 X 3 Convolution (ReLU)
2 X 2 Max Pooling
2 X 2 Transposed Convolution
1 X 1 Convolution (Sigmoid)
Skip Connection and Concatenation

Figure 8.Visual Representation of the U-Net model [11].

An important aspect of the U-Net model is the use of skip connections between the encoder and decoder blocks. Before the down sampling of each encoder block, the outputs of the double convolutions are taken and passed on to be used as skip connections in the decoder block after the up sampling in their respective layers. This step allows the model to regain some of the information that was lost during the down sampling processes in the encoder blocks.

3. OPTIMIZATION OF THE NETWORK

Most of the effort involved in training tasks using DNNs and CNNs alike will go towards optimizing them to get the best performance possible. There are many techniques that help boost the performance of a neural network, however there is a limit to how powerful the model should be. Training a model that is too powerful will likely result in a model that overfits the training data, therefore there is much to consider when attempting to improve the performance of a given model. The following section will dive into some of the options to fine-tune a model in ways that may generalize better new data that the model has not encountered.

3.1. Performance Improvements

3.1.1. Hyperparameter Tuning

There are various ways to tune the model's hyperparameters to increase the performance of a model. For example, increasing the number of layers and the number of neurons per layer can increase the ability of the model to detect increasingly complex patterns, but it also greatly increases the probability of overfitting. Another method to improve the performance is to increase the learning rate so the model converges to a solution faster. However, increasing the learning rate too much may cause the model to be unable to settle on a good solution. There is a simple way to remedy this by implementing a learning rate scheduler. As the model goes through epochs, changing the learning rate to smaller and smaller values will help the model quickly learn in the initial epochs and approach better solutions more closely on later epochs.

Another way to improve performance is to increase the number of epochs and the batch size that the model accepts to make its forward and backward propagation steps while training on the dataset. Of course, the amount of computational resources will be a limiting factor on how big the batch size can be for training. However, the number of epochs during training will let the model approach closer to a good solution without worrying about running out of resources, but it will take more time to train. Increasing the number of epochs also increases the chances of the model overfitting .

3.1.2. Improved Optimizers

An easy way to get faster convergence of a model is to simply use a faster optimizer. Gradient descent works well for some tasks and does get models to converge on good solutions, but there are much better and faster optimizers that can help improve a model's overall performance. A common method for optimizers to speed up the training of a model is to implement a momentum variable of sorts along with the gradient of the error. This concept is used by many optimizers like the Nesterov Accelerated Gradient [13], AdaGrad [14], RMS Prop and Adam optimizers [15]. The Adam optimizer (adaptive moment estimation) and the RMS Prop use an average of the previous gradients in order to compute the steps towards a better performing algorithm. This makes the learning rate adaptive and means the initial epochs can learn quickly while the later epochs can take smaller steps and converge closer to a maximum.

3.1.3. Initializers

When a model starts its first epoch, the layers in the model must have values for the weights in order for the model to make predictions and calculate the gradients of the weights based on the loss function. It can be as simple as randomly initializing the weights and start training, but that can cause the model to either find poor local maxima or even diverge and never come to a decent solution. Therefore, it can be useful to use a strategy to initialize the model weights that will help the model train to a better solution. One strategy is to ensure that the variance of the inputs in a layer is the same as the variance of its output. Glorot, He and LeCun are examples of initializers that use this concept to initialize the weights with random values within a range that will ensure that the model can converge to an acceptable solution [16].

3.1.4. Batch Normalization

Batch normalization [17] is a form of scaling applied to the data throughout the training period, and it is very commonly used in artificial networks for increasing the accuracy of models. The process begins by “0 centering” the features and then applying a scaling algorithm to get an even distribution of all features. This is generally done between each layer of the model and can cause the model to converge to an acceptable solution in fewer epochs. While the added computational cost of the several batch normalization layers added to the model can cause the model to take longer to complete a single epoch, the reduced number of required epochs generally counteracts this and sometimes makes the model converge to a better solution more quickly. It also naturally adds some form of protection from overfitting to the model and allows for a higher learning rate, which then increases the convergence rate of the model even further, increasingly helping mitigate the added cost of the normalization layers.

3.2. Regularization Techniques

Regularization [18] is the practice of counter-balancing an algorithm’s tendency to overfit the data. For instance, if a model is learning on a dataset that has an underlying pattern that is relatively simple but the model has a tendency of overfitting the data, regularization can help reduce or eliminate the overfitting problem.

3.2.1. Dropout

While dropout [19] is an effective technique to add regularization to the model, it increases the required number of epochs for the model to converge. The idea behind dropout is to drop a random subset of neurons at every training step, forcing the remaining neurons to increase their influence on the final prediction. The end goal is to create a network of neurons that are less reliant on each other, and more independent when making predictions on new data. This concept has been proven to be an excellent regularization technique and has even been shown to boost the overall performance of the model. For this reason, it is one of the most common regularization techniques used in deep learning.

3.2.2. Early Stopping and Callback

Early stopping and callbacks go hand in hand regarding deep learning models. When training a model, it can be worth keeping track of the error on the validation set to have an estimate on the model’s ability to generalize to new data. Early stopping involves stopping the training process when the validation error reaches a minimum. During training, the error iteratively decreases and as soon as an increase in error is detected, the training stops. The problem with this method is that it is possible that the model stops at a suboptimal local maximum. The remedy for this is to add a patience variable to the stopping criteria in order for the model to get past local minimums. This means that even if an increase in the validation set error is detected, the model continues training for a number of iterations determined by the patience variable. If there is a better solution, the model has a chance to get past this hump and keep training towards that solution. If the error keeps increasing for the given patience variable number of iterations, a callback is invoked. The input weights of the model are returned to the values that gave the minimum error.

4. Experimental Setup

4.1. The Dataset

The dataset used for training in this research was downloaded from the "UW-Madison GI Tract Image Segmentation" on Kaggle website [2]. It has a total of 38,496 MRI scans of actual cancer patients who required daily scans for their radiation treatments. These images vary in size, but the majority of them have a resolution of 266x266 pixels. All images were encoded as 16-bit grayscale format (1 color channel). The images were organized and labeled by their case number, day of treatment, and the number of the slice. There are many slices per day, each one slightly offset from the last, and together the scans represent a 3-dimensional image of the patient's organs. There is also a dataset given for this project, which contains a list of IDs that correspond with the names of the files of all the images. The dataset also contains a class column that specifies which organ the segmentation represents, and the segmentation column itself contains the mask that corresponds to the class, if it happens that the organ clas appears in the image. The segmentations are all RLE encoded to reduce the size of the pandas data frame. In total, the dataset has 115,488 instances in the training set. Figure 9 shows an image from the database together with its segmentation mask.

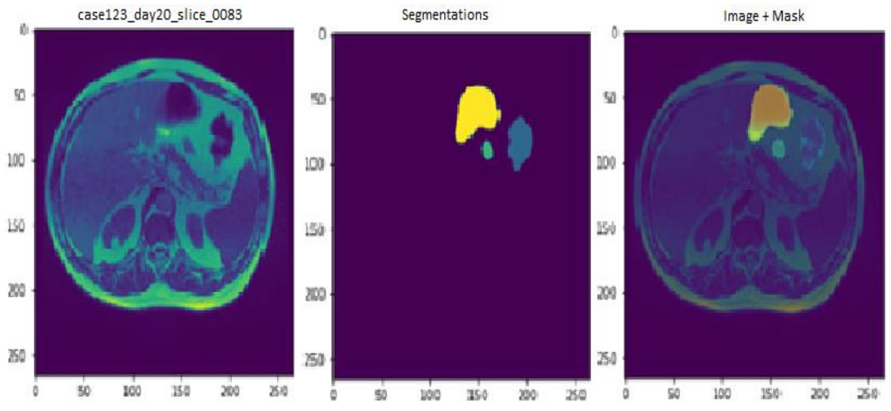

Figure 9. A sample image with its segmentation mask.

In this example, the different colours of the segmentation masks represent the 3 different classes found in the dataset. This is simply for visual purposes to be able to separate the segmentation results into various classes. The yellow mask represents the stomach, the blue mask represents the large intestine, and the green one represents the small intestine.

In order for the model to be able to use the images for training, various preprocessing steps were considered to set up the data for the model. A few of the instances were incorrectly labeled, therefore these instances were considered bad data and were filtered out to give the model the best chance to detect the correct patterns in the dataset. The image paths were imported and placed with their respective IDs in the data frame. Then, the segmentations were split into 3 separate columns for their respective classes for each image to reduce the number of duplicate image IDs. The data was then fed into a keras data generator where the images were loaded in batches of size 80 images when training began. The images were all resized to 272x272 pixels (for U-Net implementation reasons that will be described shortly) and the information from the RLE codes from each segmentation field were extracted and formatted into a proper mask as the model trained on each batch.

4.2. The U-Net Implementation

Based on the dataset, it is best to implement the model in a way that it predicts the segmentation of the desired class separately since there is a possibility that the image does not contain all classes, or even any of the classes at all. The model should be able to output all three of the segmentations independently based on what organs are found in the image. This section presents an overview of the model used in this research.

For implementation of the U-Net model for this dataset, a double convolution layer block is used throughout it to reduce the length of the code. This layer consists of two back-to-back 3x3 convolution layers with ‘ReLU’ as the activation function and zero padding around the image to preserve the dimensions of the image. The initializer used is the default one of the convolution layer, which is the Glorot uniform initializer. The encoder block starts with a batch normalization layer followed by a double convolution layer, a 2x2 max pooling layer and a dropout layer at 30% dropout rate. The decoder block has a 2x2 transpose convolution layer with zero padding and Glorot uniform initializer and concatenates the output with skip connections to be used as an input for another double convolution layer. This is also followed by a dropout layer with 30% dropout.

The building blocks previously mentioned are combined in a manner consistent with the general format of the U-Net model. The model has four encoder blocks that down-samples the images, starting with 32 filters and multiplying the number by two after every block, and then a double convolution block with 512 filters. This is followed by 4 decoder blocks that up-samples the images, and the number of filters is halved after every block until the number of filters returns to 32. The output layer consists of a 1x1 convolution layer with zero padding and a ‘sigmoid’ activation function. The number of filters is 3, but since the convolution is 1x1, the filters simply output 3 masks that correlate with each class. Figure 10 demonstrates the structure of the model described above.

It is worth mentioning that the data fed through the model must be resized to 272x272 pixels for the model to behave correctly when down sampling and up sampling. The reason for this comes from the varying image sizes found in the data set, but more specifically the actual dimension of some of the images given. Most of the images have a size of 266x266 pixels, which is a dimension that causes errors for any segmentation models in the max pooling and concatenation steps. In the max pooling step, the 2x2 stride in combination with the 2x2 filter means that the image dimensions (266 height and 266 width) get halved to 133x133. This step in particular does not cause the error, but the following max pooling is what triggers the error. The down sample still happens with no error, but since it is an integer division, the floor of the result is taken and passed to the next step (66x66). When the other side of the network up samples, it will take 66x66x1, and up sample it to 132x132 pixels and attempt to concatenate it with the correlating input of 133x133 pixels and cause a mismatch error. This is the reason why the images going into the model get resized into a value that is a multiple of 16, since it must be able to be divided up to 4 times to avoid odd numbered dimensions, given that there are 4 max pooling layers.

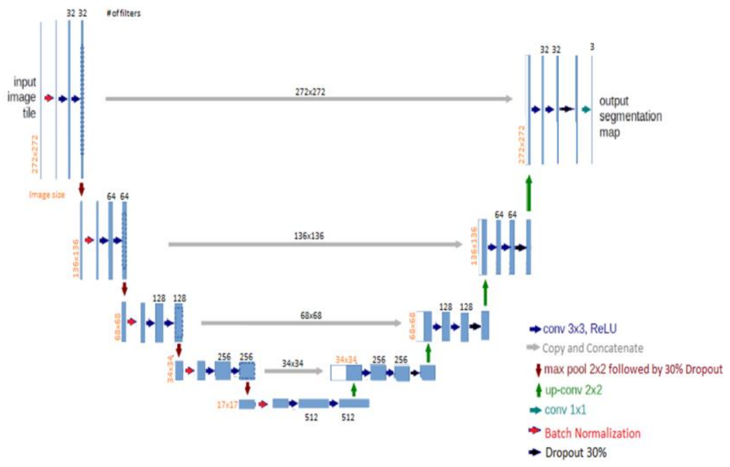

Figure 10. The U-Net implementation for the semantic segmentation model.

4.3. Performance Measure

For most binary classification applications, the accuracy measure of a model is sufficient to understand how well a model is performing on the data. Accuracy is measured by how many classes were correctly labeled as belonging in the positive class. However, the accuracy measure does not take into consideration the number of false positives. A 100% accuracy can be achieved by assigning all pixels in an image as a positive finding. Therefore, the accuracy measure by itself might be misleading if used alone to train the model.

The dice coefficient [20] is a performance metric that uses both concepts of precision and recall to measure performance. Precision is a metric that keeps track of the true positives and the false positive, so the model can track how well it is predicting the positive class. Recall keeps track of the true positives and the false negatives. This combination of recall and precision is also called the F1 score and it helps get a better estimate of the performance of the model.

$$Dicecoefficient = \frac{2 \times the\ area\ of\ overlap\ between\ predicted\ segmentation\ and\ ground\ trouth}{thetotal\ number\ of\ pixels\ in\ both\ predicted\ segmentation\ and\ ground\ truth}$$

Another way for the model to gain information on its predictions is to use a loss function to keep track of the correct and incorrect predictions. Binary cross entropy (BCE) [20], commonly referred to as the log loss, measures the dissimilarity between the true label and predicted probability distribution over two classes, and can be defined as:

$$BCE(y, \hat{y}) = -(y \times log \log \ (\hat{y}) + (1 - y) \times log\ log\ (1 - \hat{y}))$$

where $\hat{y}$ is the prediction value by the model.

4.4. Compilation of the Model

The model was compiled using the Adam optimizer with a learning rate of 0.002, the binary cross entropy loss function, and the dice coefficient. An early stop and callback was also implemented with a patience of 3, meaning the model will stop and call back its best weights if there is no improvement to the validation error after 3 epochs. Finally, the epochs are set to 40 to give plenty of iterations for the model to find a good solution and the callback with a value of 4 epochs will ensure the model does not overfit the data. A learning rate scheduler was also implemented with a factor of 0.3 and a patience of 2, meaning the model's learning rate is multiplied by 0.3 when there is no improvement for 2 epochs so that it can approach the local maximum more closely.

5. RESULTS AND ANALYSIS

The model was trained using the dataset downloaded from Kaggle website [2]The model's . predictions were analyzed using a combination of the dice coefficient and the Hausdorff distance[21][22]. The Hausdorff distance uses the distance between the furthest point of the predicted segmentation and the closest point of the actual segmentation. It repeats this calculation at interval distances to get a good understanding of the distance between both segmentations. The evaluation of the results uses this concept on a 3-dimensional level by combining the segmentations of each slice from a single day to create a 3-dimensional volume and applying the Hausdorff distance between the created volumes. Therefore, the evaluation not only looks at each individual slice, but also looks at other surrounding slices to measure the accuracy of the model. The model's score uses 40% of the value of the dice coefficient added with 60% of the value obtained by the Hausdorff distance. Using this performance measure, the proposed model managed to get a score of 81.86%.

6. CONCLUSIONS AND FUTURE WORK

The model described in this paper managed to achieve a promising performance that could prove useful for helping radiation therapists in segmenting organs in the scans taken of patients on a daily basis. The benefits of this model will not only affect the lives of the radiation therapists, but also the patients they treat by improving wait times and quality of the treatment itself. Finally, with the reduced treatment times for each patient, cancer clinics will naturally be able to accept a higher volume of patients per day and people diagnosed with cancers in the GI tract will be able to get treatment more quickly. All these benefits mean that people diagnosed with this kind of cancer can have a higher chance of survival with fewer side effects.

In the Future, the model will be further tuned to provide a better performance score. This tuning phase will consider different network layouts as well as hyperparameters tuning. Furthermore, the model could be used as a stepping stone towards creating more models to segment other organs of the body to help therapists with treating other kinds of cancer.

REFERENCES

[1] Sunnybrook Health Sciences Centre,Toronto, Canada, https://sunnybrook.ca/content/?page=occ-radonc-cancer-mr-linac

[2] University of Wisconsin (2020). *UW-Madison GI Tract Image Segmentation*. Kaggle. https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation

[3] Melina Arnold, Christian C. Abnet, Rachel E. Neale, Jerome Vignat, Edward L. Giovannucci, Katherine A. McGlynn, and Freddie Bray, (2020) "*Global Burden of 5 Major Types of Gastrointestinal Cancer*", Gastroenterology Vol. 159, No. 1, pp 335 – 349.

[4] Kevin McGuinness and Noel E O'connor, (2010) "*A comparative evaluation of interactive segmentation algorithms*", Pattern Recognition, Vol. 43, No. 2, pp 434 – 444.

[5] Ian Goodfellow, YoshuaBengio, Aaron Courville, (2016) "*Deep Learning*", The MIT Press.

[6] Francois Chollet, (2017) "*Deep Learning with Python*", Manning.

[7] Géron, A., (2022) "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems",* 3rd edition, O'Reilly Media, Inc.

[8] Norman Matloff, (2017) "*Statistical Regression and Classification: From Linear Models to Machine Learning*", 1st edition, Chapman and Hall/CRC.

[9] Yamashita, R., Nishio, M., Do, R.K.G., Togashi K., (2018) "Convolutional neural networks: an overview and application in radiology", Insights into Imaging, Springer, Vol. (9), pp. 611–629. https://doi.org/10.1007/s13244-018-0639-9.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox., (2015) "U-Net: Convolutional Networks for Biomedical Image Segmentation", Computer Vision and Pattern Recognition. http://arxiv.org/abs/1505.04597.

[11] Ibtehaz, N., and Rahman, M. S., (2020) "*MultiResUNet : Rethinking the U-Net architecture for multimodal biomedical image segmentation*", Neural Networks, ELSEVIER, Vol. (121), pp. 74-87.

[12] CejudoGrano de Oro, J. E., Koch, P. J., Krois, J., Garcia Cantu Ros, A., Patel, J., Meyer-Lueckel, H., &Schwendicke, F., (2022) "Hyperparameter Tuning and Automatic Image Augmentation for Deep Learning-Based Angle Classification on Intraoral Photographs—A Retrospective Study", *Diagnostics*, *12*(7). https://doi.org/10.3390/diagnostics12071526

[13] Y. Nesterov, (1983) "A method of solving a convex programming problem with convergence rate O(1/k2)", In Soviet Mathematics Doklady, Vol. 27, pp. 372–376.

[14] Duchi J., Hazan E., Singer Y., (2011) "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", The Journal of Machine Learning Research, Vol. (12), pp. 2121−2159.

[15] Diederik P. Kingma, Jimmy Ba, (2017) "Adam: A Method for Stochastic Optimization", arXiv:1412.6980v9, https://doi.org/10.48550/arXiv.1412.6980

[16] Li, H., Krček, M., Perin, G., (2020). "A Comparison of Weight Initializers in Deep Learning-Based Side-Channel Analysis", Applied Cryptography and Network Security Workshops ACNS, Lecture Notes in Computer Science, Springer, Vol (12418), pp.126-143, https://doi.org/10.1007/978-3-030-61638-0_8

[17] Ioffe S., Szegedy C., (2015) "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv:1502.03167v3, https://doi.org/10.48550/arXiv.1502.03167

[18] Kukačka J., Golkov V., Cremers D., (2017) "Regularization for Deep Learning: A Taxonomy", arXiv:1710.10686v1, https://doi.org/10.48550/arXiv.1710.10686

[19] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., (2014) "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research, Vol. (15), pp. 1929-1958.

[20] Shruti J., (2020) "A survey of loss functions for semantic segmentation", IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 1–7. doi: 10.1109/CIBCB48159.2020.9277638.

[21] Rockafellar, R. Tyrrell; Wets, Roger J-B, (2005) "*Variational Analysis*", Springer-Verlag. pp. 117.

[22] Birsan, T., Tiba, D., (2006) "*One Hundred Years Since the Introduction of the Set Distance by DimitriePompeiu*" In: Ceragioli, F., Dontchev, A., Futura, H., Marti, K., Pandolfi, L. (eds) System Modeling and Optimization. CSMO 2005. IFIP International Federation for Information Processing, Vol. (199), pp. 35–39, Springer, Boston, MA. https://doi.org/10.1007/0-387-33006-2_4