DEEP LEARNING FOR SEMI-AUTOMATED BRAIN CLAUSTRUM SEGMENTATION ON MAGNETIC RESONANCE (MR) IMAGES

A THESIS IN

Computer Science

Presented to the Faculty of the University
Of Missouri-Kansas City in partial fulfillment
Of the requirements for the degree

MASTER OF SCIENCE

By
Ahmed Awad H Albishri

B.S., King Abdulaziz University – Rabigh, Saudi Arabia, 2013

Kansas City, Missouri
2018

DEEP LEARNING FOR SEMI-AUTOMATED BRAIN CLAUSTRUM SEGMENTATION ON MAGNETIC RESONANCE (MR) IMAGES

Ahmed Awad Albishri, Candidate for the Master of Science Degree

University of Missouri-Kansas City, 2018

ABSTRACT

In recent years, Deep Learning (DL) has shown promising results with regard to conducting AI tasks such as computer vision and speech recognition. Specifically, DL demonstrated the state-of-the-art in computer vision tasks including image classification, segmentation, localization, and annotation. Convolutional Neural Network (CNN) models in DL have been applied to prevention, detection, and diagnosis in predictive medicine. Image segmentation plays a significant role in predictive medicine. However, there are huge challenges when performing DL-based automatic segmentation due to the nature of medical images such as heterogeneous modalities and formats, the very limited labeled training data, and the high-class imbalance in the labeled data. Furthermore, automatic segmentation becomes a challenging task, especially for Magnetic Resonance Images (MRI). In reality, it is a time- consuming procedure that requires trained biomedical experts to manually segment or annotate such MRI datasets. The need for automated segmentation or annotation is what motivates our work.

In this thesis, we propose a semi-automated approach that aims to segment the claustrum in brain MRI images. We recognize that the claustrum is an information

hub of human brains and can be used to find significant patterns from the segmentations. We applied a 2-Dimensional CNN model called U-net to segment the human brain dataset comprising 30 manually annotated subjects provided to us by the Department of Psychiatry at the University of Missouri-Kansas City. Our approach consisted of the following steps: (1) preprocessing, including converting, the data into Digital Imaging and Communications in Medicine (DICOM), re-sampling and selecting the claustrum slices, and applying an ROI selection; (2) building the claustrum model; (3) automatic segmentation; and (4) evaluation and validation. For the model validation, we used the cross-validation technique with n = 5. We administered the Dice coefficient index to evaluate the results and we achieved a Dice score of approximately 70%. A domain expert also evaluated the results.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Computing and Engineering, have examined a thesis titled “Deep Learning for Semi-Automated Brain Claustrum Segmentation on Magnetic Resonance (MR) Images” presented by Ahmed Awad Albishri, candidate for the Master of Science degree, and hereby certify that in their opinion, it is worthy of acceptance.

Supervisory Committee

Yugyung Lee, Ph.D., Committee Chair
Department of Computer Science Electrical Engineering

SEUNG SUK KANG, Ph.D.
Department of Biomedical Sciences

Zhu Li, Ph.D.
Department of Computer Science Electrical Engineering

Table of Contents

ILLUSTRATIONS

Figure Page

TABLES

ACKNOWLEDGEMENTS

I would like to FIRST thank my thesis advisor, Prof. Yugyung Lee for the amazing support and cooperation throughout my work on this thesis. The encouragement and motivation I received from her motivated me to learn more and work harder on this thesis; she was so helpful and knowledgeable whenever I faced a challenge or a problem. I am so glad she supervised my work.

I would like to thank my family, my wife, and my friends for the support and the encouragement I received from them throughout my master's program and especially this thesis. A very special thank go to my mom for her special love, motivation, and encouragement.

Finally, I wish to thank the members of my dissertation committee: Dr. Kang from the Department of Biomedical Sciences at the University of Missouri – Kansas City, for his help and cooperation in providing the dataset and for offering his time. Also, Dr. Zhu Li from the Department of Computer Science Electrical Engineering, for generously offering his time and support.

CHAPTER 1

INTRODUCTION

Since their invention, medical images in their deferent modalities such as magnetic resonance imaging (MRI), computed tomography (CT), ultrasound, and various other imaging techniques have all been estimable for doctors and radiologists to detect and diagnose diseases [1]. Every image stored in a computer saved in a format, these formats provide a standard way to store and retrieve the image information. Popular image formats like JPEG, PNG, TIF, and others represent the image in 2-dimensional arrays [6]. However, medical images are stored in different formats, as they consist of more than one image or slice, and represent the anatomical volume acquired from imaging machines [7]. These imaging technologies tremendously add to the knowledge of normal and diseased anatomies for both research and treatment plans [6].

In computer vision, convolutional neural networks (CNNs) are driving advances in recognition [6] with this success. They also dominate the detection, segmentation, and recognition of objects within images, with performance comparable to that of humans [6]. In all of these tasks, labeled data are used to teach the machine in a supervised learning manner. CNNs have been employed to solve different tasks in the field of medical imaging. Segmentation tasks play a significant role in delineating different anatomical structures and other regions [6]. Segmentation presents a more challenging task in the field of medical imaging due to the following reasons:

1. The nature of human anatomy shows major modes of variations.
2. There are different modalities and formats for images.

3. The labeled data are insufficient.

4. There is a high-class imbalance in the labeled data.

5. It requires a field expert to manually segment and validate the images.

6. Segmentation procedures can be tedious and time-consuming [7].

In brain anatomy, claustrum is a thin, irregular, vertical curved sheet of subcortical gray matter on each side of the head. It has been hypothesized that claustrum plays a vital role in consciousness [10] [11]. Figure 1 shows an illustration of the claustrum in the brain's anatomy.

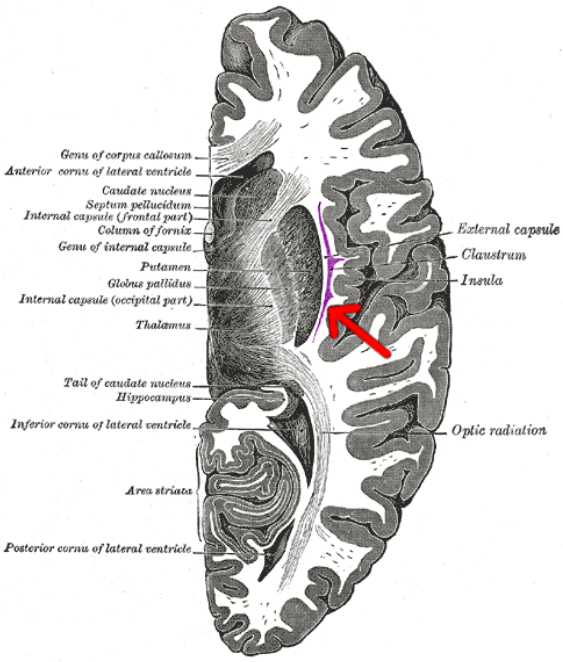

Figure 1 Shows Illustration of one side of the brain in an axial view; Claustrum is Indicated by the Arrow and Purple Color

1.1 Motivation

The increasing growth of medical imaging highlights the need for employing advancements in computer vision [6] to improve and ease the procedure of medical

image segmentation. Current work in this field shows remarkably amazing results using CNNs, from cancer detection [10] to brain tumor segmentation [11] and many other medical related problems. Hence, it is important to develop an automated method that employs the use of powerful algorithms like CNNs, for segmentation, detection, and classification in medical images. This method should tackle the problem faced by the practitioner and save the time.

1.2 Problem Statement

Manual segmentation of the claustrum on MR images is a time-consuming and challenging procedure that requires experts to do it. CNN algorithms' ability to segment natural and medical images is the current state of the art in computer vision. Thus, there is a need to automate the claustrum segmentation procedure, so it can be done in a much faster way, with results that are comparable to those of the experts or even better. Developing a tool to help the neurologist and doctors segment the claustrum in an automated way, is a process that involves facing and tackling different challenges from many aspects. These include the following: (1) understanding different medical imaging formats and their representation, as most of the machine learning and deep learning algorithms take the input data in a raw array-like format; (2) converting the image dataset into a suitable format; (3) up-sampling the dataset into the required image dimensions; (4) selecting Dicom slices where claustrum can be clearly viewed; (5) converting the dataset into NumPy arrays and applying the manual region of interest (ROI) to the claustrum surrounding; (6) preprocessing before model fitting and training; and (7) creating a web application to provide automated

segmentation. In this thesis, we will address all these problems, and then we will validate and evaluate our results with a domain expert.

CHAPTER 2

BACKGROUND

This chapter provides information about all the key terms related to deep learning, convolution neural networks, U-Net model, and deferent tools used in the implementation of the framework. We also discuss the related work to our problem.

2.1 Deep Learning

Deep learning (DL) is one of the machine learning classes. It is also known as deep structured learning because of learning data representation on hierarchal levels. By stacking multiple layers on multiple levels to extract features from data, in such types of architecture, every layer of output is the input for the successive layer. Deep learning depends on extensive neural networks that need large amounts of data to learn from and more powerful computers for faster computations. The ability to run deep learning algorithms on graphical processing units (GPUs) has accelerated the development in this field and reduced the longer time needed to train these algorithms, particularly by using parallel computations on multiple GPUs [6] [13]. In machine learning, there are two types of learning: supervised and unsupervised. In supervised learning, the algorithm trained on labeled input data examples, and this type of learning is the most common. Examples of supervised learning are classification, recognition, and annotation. In the second type of learning, unsupervised learning, unlabeled data are fed to the algorithm, and it tries to find a pattern on the input data. A famous example of unsupervised learning is clustering [6] [13].

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are the most popular type of algorithm in computer vision as they can learn from data by learning the internal feature representation [13]. The design of CNNs comprises multiple neurons these neurons have learnable weights and biases called the network parameters, the neurons at each layer of the network perform a mathematical function on the inputs. The network from a high-level overview gets an input image in raw pixels at the input layer and output a score for each class in case of classification, or a pixel level classification in case of segmentation. In between the input and output layer, we have the hidden layers, no limit to how many hidden layers can be added to the network. The hidden layers can be any of convolution layer, non-linearity layer, pooling layer, and fully connected layer. We will go into more detail about these layers below.

During the training phase the goal is to optimize these values to the best possible, the process of optimization uses algorithms like gradient descent, stochastic gradient descent, or its optimized variants. An important step to achieve this by applying the backpropagation technique [18] to propagate the error backward throughout the network to update the weights. While training we have multiple variables or what also called (hyperparameters) which controls the training properties, the learning rate, for example, control the speed of updating the parameters of the network during the training phase. Section 5.2.1 discusses the hyperparameters used in the thesis.

One of the CNNs abilities is that the learned features of an object can be used to detect the same object anywhere in the image. CNNs are considered the current state-of-the-art in computer vision for multiple tasks, they come in different architectures,

but the primary layers types in every CNN are (1) the convolution layers, (2) non-linearity layers, (3) pooling or sub-sampling layers, and (4) fully connected layers. CNNs differ in their architecture; each network can have its architecture, Figure 2 shows the architecture of LeNet-5 proposed by Yann LeCun [17].

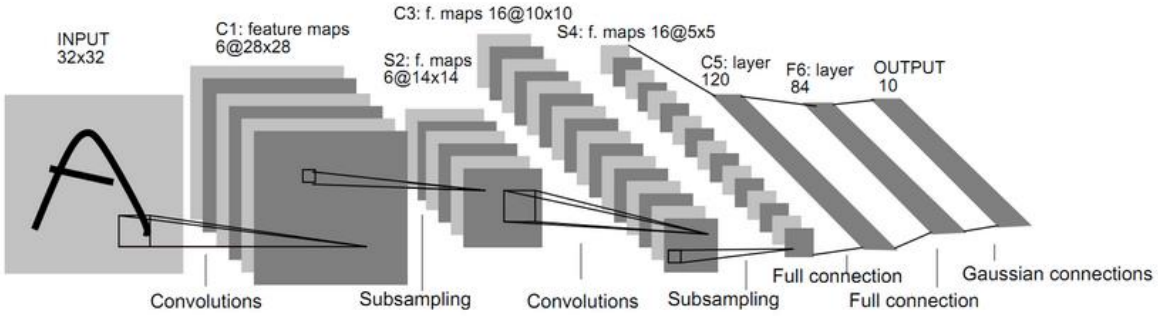

Figure 2 CNN architecture of LeNet-5 [17]

Convolutional layers:

The main components of the convolutional layer are the filters and feature maps. The filters, which are the neurons of the layer, have a spatial size, e.g., in Figure 3, the blue box shows a filter with a size 3x3 with weights for each pixel. The filter slides over every pixel in the input image and applies element-wise multiplication between the filter and its receptive field (see Figure 3 - the red box) in the image, and adding the multiplication output will produce a single element for the feature map as in Figure 3, the green box. The first layer, also called input layer, will take the image pixel values as input, whereas the next layer's input is the outputted feature map from their previous layers [13].

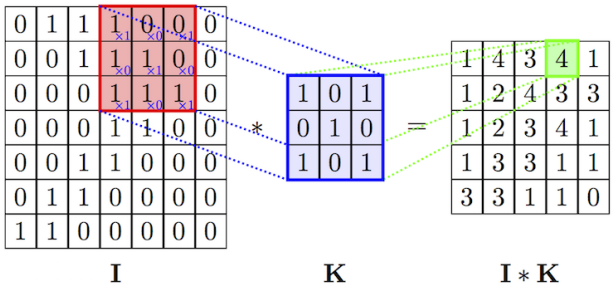

Figure 3 Feature Map Extraction

To see how the convolution layer works we have used one image from the dataset and run convolution layer on it; Figure 4 left image is the input and output images to the right. It is important to note that due to the different weight initialization repeating this test will provide different output images.

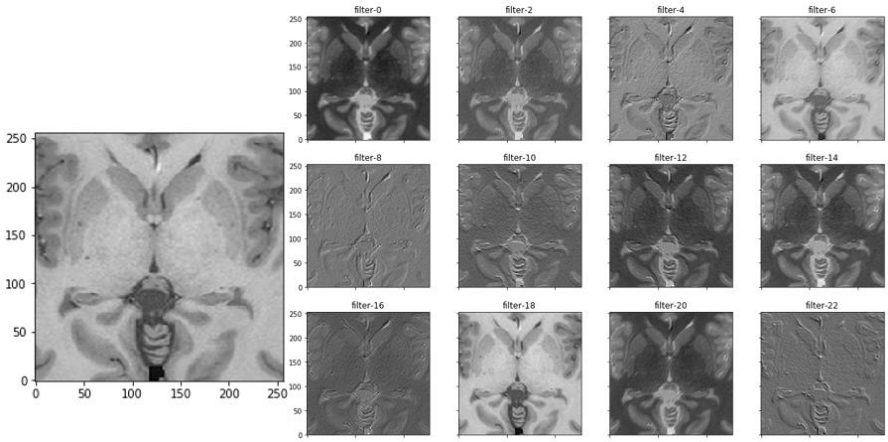

Figure 4 Conv. Layer Operation

Non-Linearity layers:

Non-Linearity layer also called activation function, to add a non-linearity to the model, there are multiple non-linearity functions. A widely used and effective non-linearity function is ReLU (Rectified Linear Unit) whose function is (x) = max (0, x). This function applies (element-wise) to the output of the previous layer in the CNNs by either a

convolution layer or a max-pooling layer. Simply put, this function will set all negative values to 0, and all other positive values will remain constant Figure 5 illustrates ReLU layer operation on a 4x4 feature map. Another function called sigmoid works by squashing the input value into a range between 0 and 1, the sigmoid function has the following equation:

$$\sigma(x) = 1 / (1 + exp(-x))$$

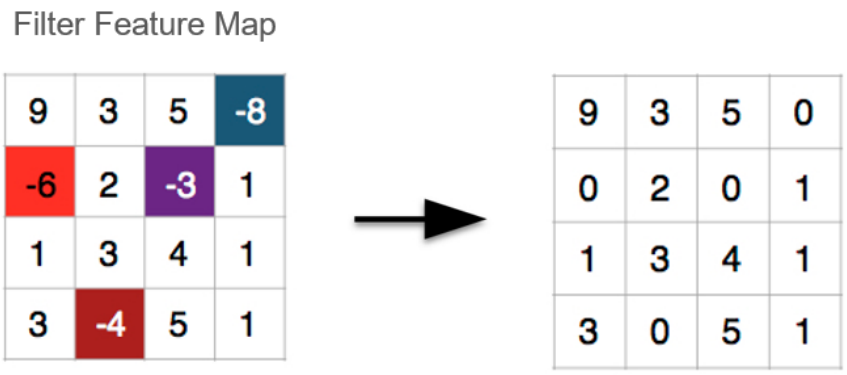

Figure 5 ReLU Layer

In Figure 6 we can see the result of applying ReLU function to after applying the convolution layer, some images in the results turned into mostly black pixels as ReLU function will change all the negative values to 0.

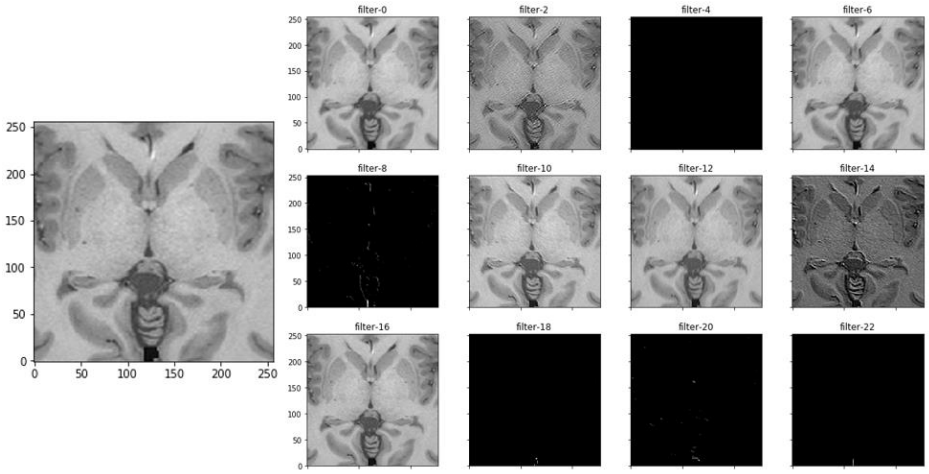

Figure 6 Conv. Layer Operation with ReLU Function

Pooling or subsampling layers:

Pooling or subsampling is a very effective layer that helps in reducing the dimensionality of the feature maps from the previous layer. A pooling layer can be applied in different ways such as taking the average value or maximum value. Max pooling is mostly applied is neural networks because of its good results [16]. In the case of max pooling, we can think about how the convolution layer is applied because max pooling is applied the same way, except there are no weights for the filter, and instead of multiplication, we only take the maximum value of from the receptive field. Furthermore, the pooling operation if applied with window size 2x2 will result in a half size dimension reduction to the image as in Figure 7 and Figure 8. Figure 7 demonstrates how the max and average pooling with window size 2x2 is applied.

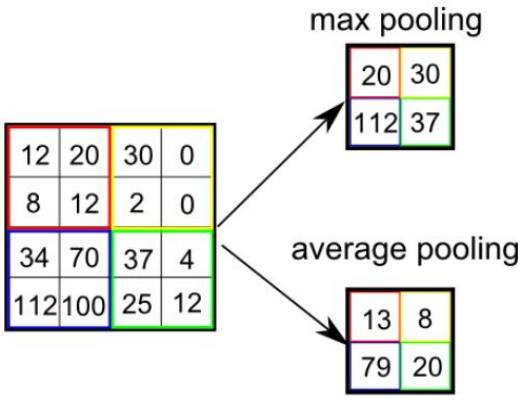

Figure 7 Example of Max, Average Pooling

We have applied the pooling operation on the input image to get more intuitive understanding; Figure 8 shows the results after applying convolution layer with filter size 3x3 and pooling window 2x2. The input image dimensions are 256x256, and the output dimension is the 128x128 half size the input image, the output image still clearly visible but after applying the pooling layer multiple times in the network the image resolution will get lower and lower.

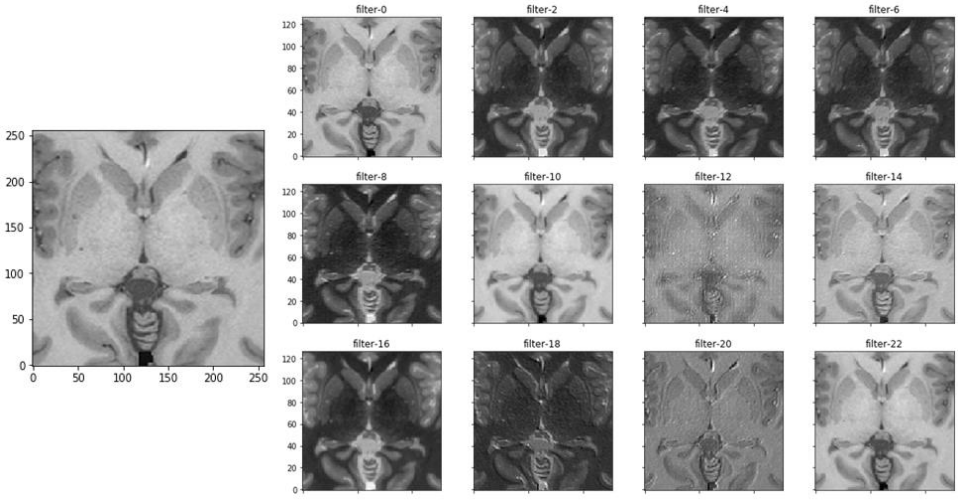

Figure 8 Conv. Layer with Pooling Operation. Left Input Image. Right Output Feature Maps

2.3 Segmentation

In computer vision, image segmentation is the process of partitioning the image into a set of pixels, based on the pixels' similarities, or another definition is to define the objects' boundaries in the image, this can be achieved by assigning a label or class to every pixel [17]. These sets of pixels represent objects or a boundary of the objects in the image. Further analysis of the image can benefit from the segmentation in multiple ways, e.g., finding the object size in the image.

There are multiple segmentation methods available such as thresholding, clustering, histogram based, and many others. Recently, CNN segmentation methods have achieved the state-of-art in image segmentation, and most of the current segmentation work in computer vision is done by employing CNNs [6]. Two popular CNN based methods are semantic segmentation and instance segmentation [6] [18]. We present both methods, in detail, with a brief history about them and a description of how each of these methods works in the following chapter.

2.3.1 Semantic Segmentation

Semantic segmentation is the understanding of image at pixel-wise level, and partitioning the image into semantic parts where each pixel in the image is assigned a semantic class. For this, a pixel-wise dense classification is needed to find the object boundaries. Semantic segmentation does not differentiate between instances of the same class. In Figure 9, the middle image segmented semantically and clearly shows that objects of the same class have correct segmentation but with no notion of object instances [6] [18]. In CNNs, when applying a pooling layer, the image is reduced

spatially, and this makes it difficult to output a clean segmentation with the same input size. To overcome this problem, a new approach was first introduced by Long, J. et al. [6]. They introduced skip connection from layers of higher resolution feature maps to improve the up-sampling of segmentation. Such architectures are called encoder-decoder architectures. The encoder part can be any CNN architecture that helps to understand the image, and the decoder part uses deconvolution or up-convolution to output the up-sampled high-resolution segmentation. SegNet, U-Net, and other architectures employ the same technique in different designs [6] [19] [20].

2.3.2 Instance Segmentation

Instance segmentation is based on a semantic segmentation with an approach of joining semantic and instance segmentation, where each pixel is assigned a semantic class and an instance label. This approach incorporates an object detector and Conditional Random Field (CRF) model to further distinguish between instances [18]. In addition to the segmentation output, this approach can count the number of instances in the image. Figure 9 shows how this method is able to distinguish between the same class instances clearly.

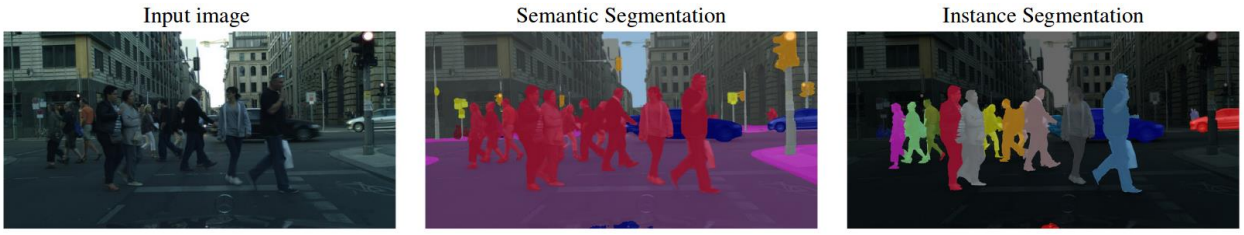

Figure 9 The Difference Between Semantic and Instance Segmentation [18]

2.4 Data Augmentation

In deep learning algorithms, the more data the algorithm trains on, the better results it can be [34]. Winners of famous competitions (e.g, ImageNet) in computer visions have utilized to increase the size of the training data by synthesizing new samples artificially from the existed data [22]. Data augmentation is also a common solution to reduce overfitting on image data [23]. Different methods can be applied to augment images like scaling, flipping, rotation, and translation. Employing the data augmentation technique in the medical image field is beneficial because labeled medical data are limited. Several research projects in medical imaging benefit from applying this method [20].

2.5 U-Net Model

Convolution network models dominate the field of computer vision, and every model comes with its own architecture and advantages. Ronneberger et al. (2015) proposed “U-Net: A Convolutional Network for Biomedical Image Segmentation” [20], U-net is a well-performing model in the field of medical imaging. U-Net model developed based on a popular architecture knows as the “Fully Convolutional Networks for Semantic Segmentation” by J. Long et al. [6]. The modifications made to U-Net enable the model to work with a smaller amount of training data while at the same time producing quality output segmentations.

U-Net architecture as described in the original paper by Ronneberger et al. (2015) [20] “The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.” Figure 10 illustrates how

U-Net is designed like an encoder-decoder model; the left side of the model is the encoder portion (contracting path), and the right side is the decoder portion (expanding path). This architecture has concatenation connections between the encoder at the left side and the decoder at the right side at every stage or level; this allows the decoder side to learn the needed features that are lost on the decoder side after the pooling laye; this helps the expanding path to output quality segmentations.

The U-Net model does a pixel-wise segmentation in which each pixel in the input image is given a class depending on the problem. The U-Net model has won several competitions, like the ISBI cell tracking challenge 2015, which only contained 35 annotated training images. It also achieved state-of-the-art results on the EM Stacks dataset that only contained 30 medical images.

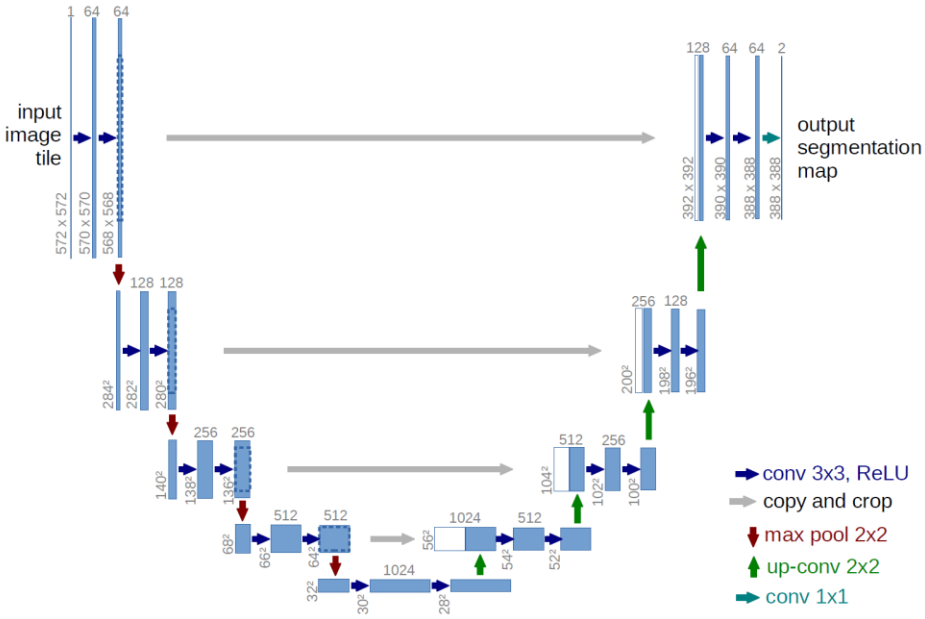

Figure 10 U-Net Architecture [20]

2.6 Medical Image Data

Medical images come in different modalities and formats, therefore, to analyze or process medical data, an understanding of the different formats used in this field is necessary. Two major file formats in medical imaging are Neuroimaging Informatics Technology Initiative (Nifti), and Digital Imaging and Communications in Medicine (Dicom) [7]. In this thesis, the Nifty and Dicom formats are the main formats we interacted with throughout processing and preparing the data.

The main difference between Dicom and Nifti is that the raw image data in Nifti is saved as a 3D image, whereas in Dicom, raw image data are 2D slices. Dicom is a standard format created by the National Electrical Manufacturers Association (NEMA), Dicom provides a standard on how to handle, store, print, and transmit medical image information. A Dicom file has two main components: the Dicom header and image data, the Dicom file format uses “. dcm” extension. Figure 11 shows the header information for one of the Dicom files from this thesis dataset, Figure 12 shows Dicom files for one subject.

```
[(0008, 0008) Image Type                         CS: ['ORIGINAL', 'PRIMARY', 'AXIAL']
(0008, 0016) SOP Class UID                       UI: CT Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.826.0.1.3680043.2.1125.1.196677519762579019139220545004074 18
(0008, 0020) Study Date                          DA: '20060101'
(0008, 0030) Study Time                          TM: '010100.000000'
(0008, 0050) Accession Number                    SH: '1'
(0008, 0060) Modality                            CS: 'CT'
(0008, 0070) Manufacturer                        LO: 'Unknown manufacturer'
(0008, 0090) Referring Physician's Name          PN: 'Unknown'
(0008, 1030) Study Description                   LO: 'None'
(0008, 103e) Series Description                  LO: 'No series description'
(0008, 1090) Manufacturer's Model Name           LO: 'Unknown model'
(0010, 0010) Patient's Name                      PN: 'Anonymous'
(0010, 0020) Patient ID                          LO: '123456'
(0010, 0030) Patient's Birth Date                DA: '20060101'
(0010, 0032) Patient's Birth Time                TM: '010100.000000'
(0010, 0040) Patient's Sex                       CS: 'M'
(0010, 4000) Patient Comments                    LT: 'None'
(0018, 0050) Slice Thickness                     DS: '1.0'
(0018, 5100) Patient Position                    CS: 'HFS'
(0020, 000d) Study Instance UID                  UI: 1.2.826.0.1.3680043.2.1125.1.6844135398866787373629426835486432 4
(0020, 000e) Series Instance UID                 UI: 1.2.826.0.1.3680043.2.1125.1.9722445008720507385152693060309295 8
(0020, 0010) Study ID                            SH: 'SLICER10001'
(0020, 0011) Series Number                       IS: '1'
(0020, 0013) Instance Number                     IS: '1'
(0020, 0032) Image Position (Patient)            DS: ['-90.75', '108.5', '-90.5']
(0020, 0037) Image Orientation (Patient)         DS: ['1', '0', '0', '0', '-1', '0']
(0020, 0052) Frame of Reference UID              UI: 1.2.826.0.1.3680043.2.1125.1.8062573707990557247363409855443222 2
(0020, 1040) Position Reference Indicator        LO: 'SN'
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 218
(0028, 0011) Columns                             US: 364
(0028, 0030) Pixel Spacing                       DS: ['1', '0.5']
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 16
(0028, 0102) High Bit                            US: 15
(0028, 0103) Pixel Representation                US: 1
(0028, 1052) Rescale Intercept                   DS: '0'
(0028, 1053) Rescale Slope                       DS: '1'
(0028, 1054) Rescale Type                        LO: 'US'
(0032, 4000) Study Comments                      LT: 'None'
(7fe0, 0010) Pixel Data                          OB or OW: Array of 158704 bytes]
<type 'list'>
```

Figure 11 A Dicom header Example from Dataset

Figure 12 Dicom Files for One Subject from the Dataset with (.dcm) Extension

2.7 Software Configuration

This section provides general knowledge and terminologies about the software tools and libraries that we used to implement this work. The used software and libraries:

Table 1. Software and Libraries Used for Implementation

3D-Slicer	NumPy
TensorFlow	Pydicom
Keras	Matplotlib

2.7.1 3D-Slicer

A 3D Slicer is an open source software platform for medical image informatics, image processing, and three-dimensional visualization. Built over two decades through support from the National Institute of Health and a worldwide developer community, Slicer brings free, powerful, cross-platform processing tools to physicians, researchers, and the public. We used this software to convert our MRI dataset from the Nifti to Dicom format. Moreover, this software was helpful while selecting the claustrum slices from the Dicom files. Figure 13 displays a 3D-Slicer application interface showing one of the subjects from the dataset.

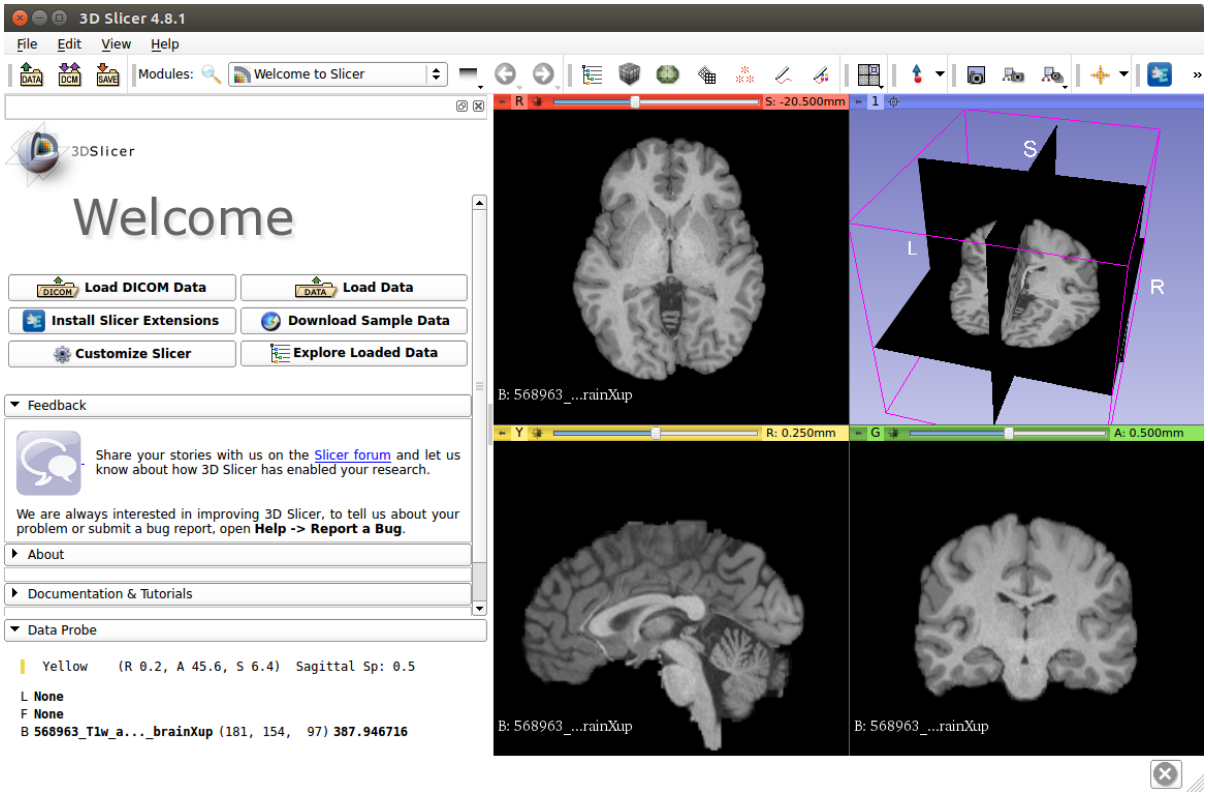

Figure 13 A 3D-Slicer Software View Displaying MRI from Dataset

2.7.2 TensorFlow

TensorFlow is an open source deep learning platform and library developed by a Google Brain team for high-performance numerical computations. TensorFlow was designed to have a flexible architecture to help developers deploy their models across a variety of platforms and more importantly, the ability to run them on GPUs for faster training. Because TensorFlow is a cross-platform library, it can be used across multiple platforms in desktops and mobiles. We have used TensorFlow GPU as a backend deep learning platform in this thesis.

2.7.3 Keras

Keras is an open source, high level, neural network library written in Python. Keras comes with the flexibility to run on top of different deep learning platforms like TensorFlow, Theano, and Microsoft Cognitive Toolkit. It enables easy and fast prototyping, with support for major CNN types and the ability to run a model easily on CPUs or GPUs. Keras was developed to give a best practice user experience and produce less effort through simple API usage and a minimal number of user actions for common cases.

2.7.4 NumPy

NumPy is the fundamental package for scientific computing in the Python programming language. It provides a powerful N-dimensional array object, linear algebra, and random number capabilities. NumPy can also be used as an efficient multi-dimensional container of generic data, this allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2.7.5 Pydicom

Pydicom is an open source Python package for working with the Dicom medical image format. This library makes it possible and easy for the developers to read and interact with Dicom files in simple python programming style, the library provides an easy to use features that give the user the ability to modify and write again to Dicom format.

2.7.6 Matplotlib

Matplotlib is one of the top Python 2D plotting libraries, can be used with any python application development as well as other python environments. Matplotlib generates high-quality images that can be used in publications. Furthermore, matplotlib developer's goal is to make this library simple and easy to use. With matplotlib, it is possible to generate plots, histogram, bar charts, and many others with just a few lines of code. All the visualization used in this thesis by this fantastic library.

CHAPTER 3

RELATED WORK

3.1 Medical Image Analysis

Medical Images can be defined as the representation of the human body's internal structure or a function of an anatomical region [7]. In natural or medical images, the two-dimensional image comes in the form of a two-dimensional array with elements called pixels; a three-dimensional image comes in the form of a three-dimensional array with picture elements called voxels. The pixel or voxel value represents its intensity, which describes its brightness and/or its color [25].

Medical images in their different formats share common properties such as Pixel Depth, which is the number of bits needed to encode the information of each pixel in the image. Photometric Interpretation helps to correctly display the image as it specifies how to read the image pixels. Metadata, which can be found in most of the images, describe the image, and medical imaging metadata can contain far more helpful information than natural images. For example, Dicom format metadata show the patient and the study information along with technical information related to the scanner and the format properties [7] [6]. Medical images come from different diagnostic modalities, in magnetic resonance images (MRI), and computed tomography (CT) images come with a grayscale interpretation [7].

A massive amount of work has been proposed on medical image analysis, and to narrow it down, we will discuss the related work on medical image segmentation using CNNs.

Table 2 summarizes some of the related work in this field, and we provide a detailed discussion about this work below.

Table 2. Related Work Summary

Architecture	Dataset	Type / Organ	Evaluation
U-Net, 2015 [20]	ISBI Cell Tracking 35 annotated training images	Electronic microscopic images / Cells	IOU: 92%
CNN, 2017 [26], Similar to U-net	BRATS 2013 challenge, 274 images	3D - MRI / Brain Tumor	Dice: 0.87
CNN, 2017 [27], combines spatial and Conv. features	MICCAI 2012 - 35 T1-w MRI volumes.	MRI - Subcortical Brain structure	Dice: 0.86
Cascade-FCN 2017	3DIRCAD - 20 CT volume	CT - Liver	Dice: 0.93

U-Net [20] architecture was already explained in 0. This architecture won the ISBI challenge for the cell tracking challenge 2015 and outperformed many other methods on the segmentation of neuronal structures in electron microscopy stacks from the ISBI challenge. The authors used an evaluation metric to calculate the accuracy of the segmentation called intersection over union (IOU). This metric's range was from 0 to 1 and was measured by dividing the area of overlap by the area of the union. The dataset in this challenge contained 35 partially annotated images, and the authors reported the benefit of data augmentation and how it helped to generate more synthetic data, especially the use of the elastic deformation technique.

In a paper by Kayalibay et al. on “CNN-based Segmentation of Medical Imaging Data” [26], the authors proposed a U-net similar architecture with two modifications. The first modification combined multiple feature maps from different scales because the original architecture only merged the copied feature maps at the same scale. The second modification made use of element-wise summation instead of concatenation when combining feature maps. The authors indicated the challenges and problems they faced by the medical data, especially the class imbalance problem and data scarcity. Two datasets were reported in their paper, one for hand MRI data and the second for a brain MRI tumor. The second dataset for the brain tumor contains four classes with a high-class imbalance of 0.96 for the background class. Both datasets are 3D-volume data. Finally, the authors reported that the second modification produced slightly worse results, whereas, the first modification did not add to the result but helped in speeding up the network convergence.

In the paper by Kushibar at al. on “Automated Sub-cortical Brain Structure Segmentation Combining Spatial and Deep Convolutional Features” [27], the authors introduced a novel CNN-based approach to segment the brain’s sub-cortical structure, by combining the convolutional features and the prior spatial features to improve the results. The spatial features were extracted from a structurally probabilistic atlas called the Harvard-Oxford atlas template. Because 3D data are memory and computationally expensive, the authors trained their network on 2.5D batches from the orthogonal views (Axial, Coronal, and Sagittal). According to the authors, incorporating atlas-based spatial features can improve the network performance significantly and based on their experiment, the atlas helped the network in segmenting difficult areas.

Furthermore, the restricted sample selection approach proposed by the authors to help the network to segment the difficult areas like the borders of the structure, the restricted sample selection idea is to train the network on the extracted sub-cortical background (negative) boundaries only.

In the paper, “Automatic Liver and Tumor Segmentation of CT and MRI Volumes Using Cascaded Fully Convolutional Neural Networks” [37] by Christ et al., the proposed model called cascaded fully CNNs (CFCNs). The authors in this model trained the model to segment the liver and lesion sequentially; this cascaded model produced a better quality segmentation. Both cascaded networks in this paper are U-Net architectures. The first network segment the liver, then the liver ROI output from the first network is passed to the second network to segment the lesion. The authors compared the segmentation accuracy between AlexNet and U-net and found that the first one’s lesion Dice score was 0.24, whereas, the second one’s lesion Dice score was 0.53. The skip connections in U-Net were powerful and proved to output quality segmentation.

3.2 Deep Learning Convolutional Neural Networks

In section 2.2, we introduced CNNs with some technical details: convolutional neural network CNNs are biologically-inspired variants of Multilayer Perceptron (MLPs). It is arguable as to whether or not they are the most popular deep learning architecture because of their effectiveness [6].

The interest in CNN started with AlexNet in 2012, and it has grown exponentially ever since. In just three years, researchers progressed from an 8-layer AlexNet to a 152-layer ResNet [29]. A true advantage of CNNs discovered from

previous works is the ability to automatically extract the key features giving CNNs the flexibility to solve multiple tasks in computer vision such as

- Classification:
    - Given an input image, CNNs will output class portability for the image. A popular example on an MNIST dataset are the handwritten digits. CNNs output the class of the digit and the classes are numbered from 0 to 9.
    - Enormous amounts of network architectures can do classification and the most popular architectures are GoogLeNet [30], VGGNet [31], and ResNet [29].
- Localization:
    - The CNNs' task is to localize the object in the image and draw a bounding box around it. CNNs first classify the object and then localize it.
    - R-CNN, Fast R-CNN, Faster R-CNN, YOLO, and Mask R-CNN are the current state-of-the-art in object localization and Detection.
- Segmentation:
    - Discussed in detail in Section 2.3. A related work discussion is below.

3.2.1 Image Segmentation

Segmentation of images in computer vision is the process of partitioning the image into a set of pixels, based on the pixels' similarities and another definition is to

define the object boundaries in the image. This is achieved by assigning a label or class to every pixel [17]. These sets of pixels represent objects or a boundary of the objects in the image. “Fully Convolutional Networks for Semantic Segmentation” [6] by Long et al. is considered the pioneering work in image segmentation with CNNs. In their work, the authors employed the method of upsampling the feature maps to output a higher segmentation resolution. To tackle the coarse segmentation maps from the upsampled layer, the authors introduced skip connections. Figure 14 illustrates the FCN for semantic segmentation architecture.

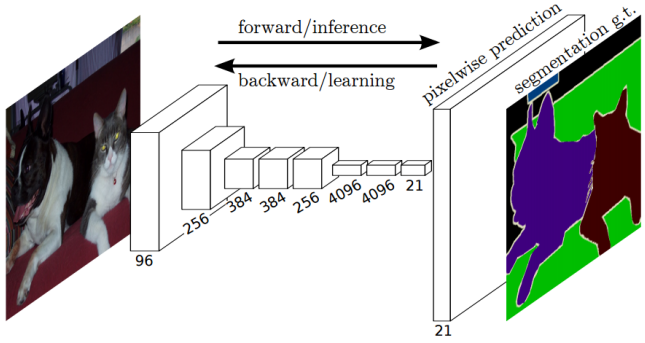

Figure 14 FCN for Semantic Segmentation Architecture

“SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation” [19] by Badrinarayanan et al. SegNet is an encoder-decoder based architecture like FCN [6]. In SegNet, the authors introduced more skip connections than in FCN, and instead of sending the left side (encoder) feature maps, the SegNet sent or copied the indices from max pooling to the right side (decoder) to produce a better segmentation. The authors of SegNet claim that sending the max pooling

indices is more memory efficient compared to FCN. Figure 15 illustrates the SegNet architecture.

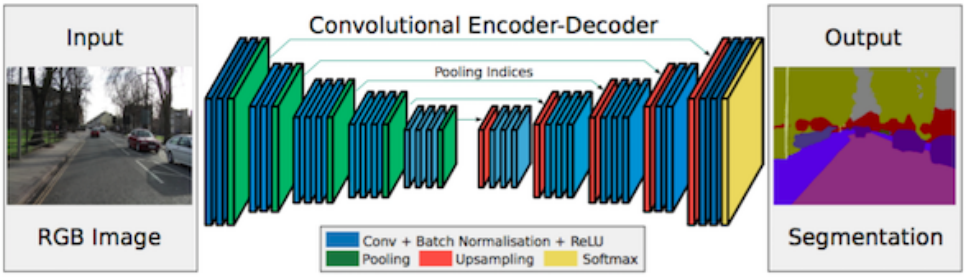

Figure 15 SegNet Architecture [19]

3.2.2 Object Localization and Annotation

The ability of CNNs in computer vision shows amazing results because CNNs can localize an object anywhere in the image and annotate the object and or the image. This challenging technique has a tremendous amount of benefit, in robotics and medicine for example. Many CNN models use this ability to enhance the model performance by localizing an object in the first step and then classifying or segmenting that region only.

R-CNN (Regions-CNN) [32] by Girshick et al. is one of the early models to come up with object detection. The goal in object detection is to find the objects in an image and classify them. An R-CNN model takes an input image and output bounding boxes and labels for each object in the image see Figure 17. R-CNN generates the set of bounding boxes for the image using a method called selective search, then takes the images in the bounding boxes to the pre-trained modified version of AlexNet. R-CNN has a Support Vector Machine (SVM) in the last layer to classify the objects. R-CNN's

main drawback is its slow speed because of its architectural design. Figure 16 shows the steps taken by R-CNN for object detection.

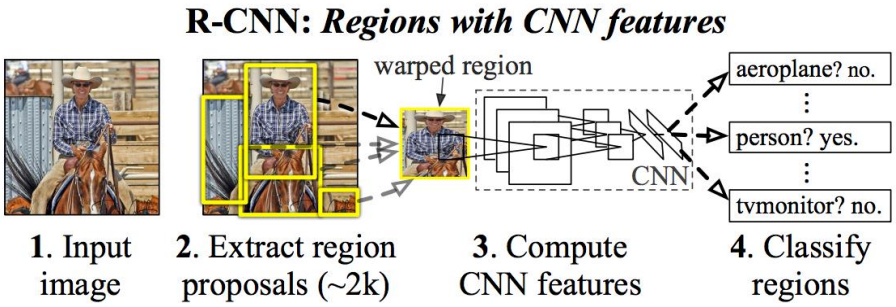

Figure 16 R-CNN Model Steps

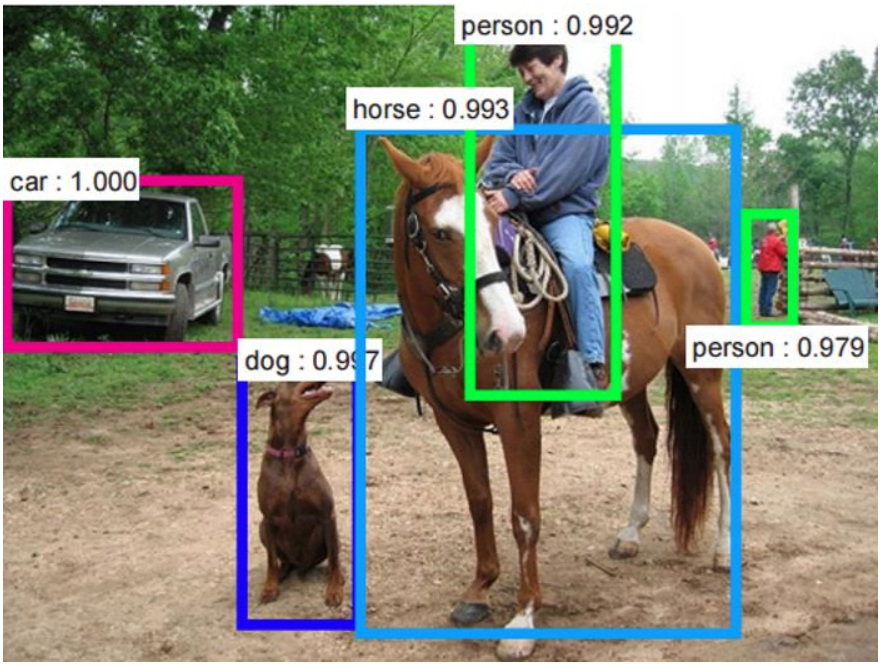

Figure 17 Faster R-CNN Model Output Showing Bounding Boxes Around Objects with Classification Accuracy [21]

Shortly after proposing R-CNN [32] many other papers published enhanced models, Fast R-CNN 2015 [33], Faster R-CNN 2016 [21], and Mask R-CNN 2017 [35]. All these papers show amazing results in localizing the objects in the image. Figure 17 illustrates the ability of Faster R-CNN model in localizing the objects in the image, the other models incorporate the same idea by using the object location or region of interest (ROI) in the image then segment or classify the ROI only.

CHAPTER 4

PROPOSED SOLUTION

4.1 Architecture

We propose a deep learning framework based on the CNNs to semi-automatically segment the claustrum on MR images. Our framework is based on the elegant U-Net architecture, the proposed solution combines different components that work together to achieve the desired goal.

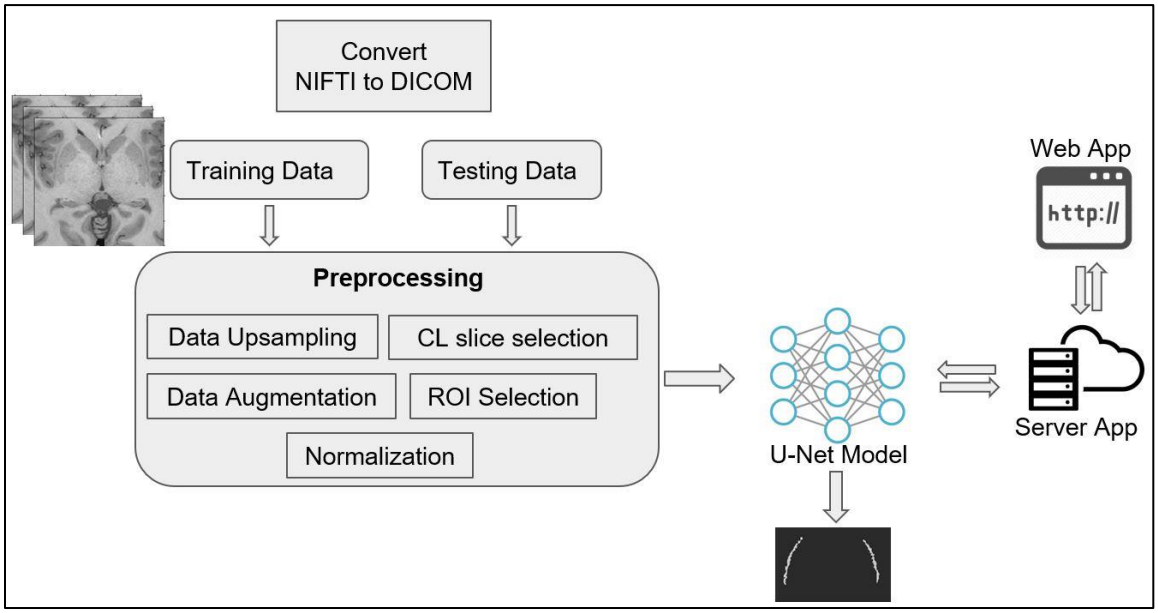

Figure 18 Proposed Solution Architecture

Figure 18 shows our proposed solution architecture; this architecture involves different steps to achieve the final goal. We describe all the steps in detail in the next sections.

4.2 Proposed U-Net Models

We used the original U-Net model in our first applied experiment, and we also created two more modified versions of the original architecture: a simpler and deeper model. All these models shared the same original design except the simpler model had a smaller number of layers, and the deeper model had an extra layer with a skip connection. Because of our training data ROI dimension 256x256 half the size used in the original U-Net paper, we changed the number of feature maps in all the models. The number of feature maps in the original U-Net start from 64 feature maps at first hidden layer and increases by a factor of 2 for each successive layer until the last layer in the contracting path. Figure 19 shows all the models' architectures from left to right. Their order is as follows: Original model, Deeper model, and Simpler model.

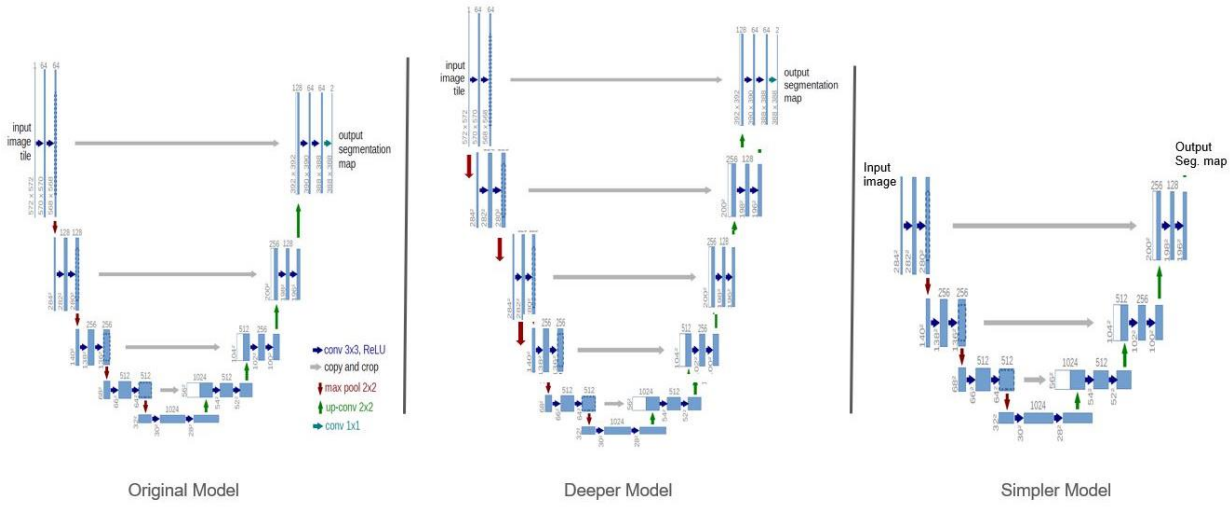

Figure 19 Experiment with Different U-Net Models Architectures, Original U-Net model image take from [20]

The authors of U-net [20] have used the cross-entropy as loss function for the model, with the soft-max function over the final feature map. Soft-max function applied to reflect the probability of a pixel being a positive or negative class. Moreover, their

network trained with stochastic gradient descent and using Caffe deep learning framework.

In this thesis, the binary cross-entropy loss function applied with ReLU activation function. Over the last feature map, we used the Sigmoid function, to output the probability of a pixel being a claustrum or not. Only pixels with a probability higher than 0.5 considered claustrum and their values changed to 1, and all other pixels changed to 0. Also, we trained the network with Adam optimizer [28] implementation of the TensorFlow framework.

4.3 Proposed Solution Steps

In this section, we describe each of the steps taken in order to achieve the final model output. The steps below are in the order of their implementation:

1. Converting the dataset into Dicom format
2. Data up-sampling and Claustrum slices selection
3. Data augmentation
4. Region of interest (ROI) selection
5. Data normalization

1. Converting dataset into Dicom format:

   This is the first step taken in our proposed solution. The original dataset format is in Nifti format, which, by default, saved the raw image data in 3-dimensions. For this step, we used the 3D-Slicer software that provides excellent features used by medical experts and researchers. The conversion of the 30 MRI Nifti files was done manually for each subject, and 29 out of 30 Nifti files converted successfully. Upon

converting the files into the Dicom format, the Slicer software gave us the ability to add and update the files' metadata. Depending on the pixel spacing (isotropic voxel resolution), the number of Dicom files (slices) for each Nifti file can be different. For 23 Nifti files, we got 182 Dicom (.dcm) files. Each file was a slice from the 3D image in the Nifti format because they had a 0.7 mm isotropic voxel resolution. The remaining 7 Nifti files resulted in 250 Dicom files due to their different isotropic voxel resolutions. The same conversion applied to the label data. Figure 20 shows the difference between Nifti and Dicom file formats for the same subject.

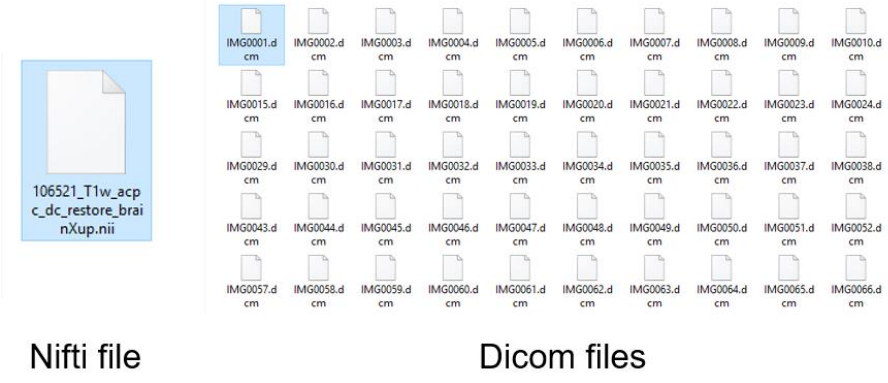

Figure 20 One Subject in Two Different Formats

2. Data up-sampling and Claustrum slices selection:

   In this step, we accomplished two things. First, by up-sampling the image data in all the slices for each subject into a new dimension (512x512), we used the same Pydicom library to read all the Dicom files for one subject at a time. By using the files' metadata to get the pixel spacing between all the slices we up-sampled all the slices into the required dimension by using the function provided by the SciPy

library (scipy.ndimage.interpolation.zoom). This zoom function took an image array as input and output the zoomed array; the implementation of this function was based on the cubic spline interpolation. This spline function is accurate, stable, and used widely [36]. The equation for this spline interpolation is

$$f(x,y) = \sum_{k=k_1}^{(l_1+k-1)} \sum_{l=l_1}^{(l_1+k-1)} c(k,l)\beta^n(x-k)\beta^n(y-l)$$

To match the 7 different Dicom subjects with the other 23 subjects in terms of the slice counts, we had to add an extra step to resample these files and change their isotropic voxel resolution. This allowed each subject to have 182 slices instead of 250. Second, from the 182 Dicom slices, the claustrum could be clearly viewed in around ~20 to ~30 slices, depending on the subject. Therefore, instead of pushing all the 182 slices, we had to select where the claustrum could be clearly viewed on the slices. For this to occur, we had to do this step for each subject, because due to the variety between the subjects, we could not select the same slice numbers for all of them. We chose, on average, 13 slices from each subject. *Figure 21* shows an example of the claustrum view selection for one subject from the dataset overlaid with ground truth label; this example shows 64 out of the 182 slices. For illustration, the presented slices start from slice 40, and we skip one slice in between. The red bounding box is to show the slices with clear claustrum. Only slices of all the new claustrum were stacked together and saved as a NumPy array with an (.npy) file extension.

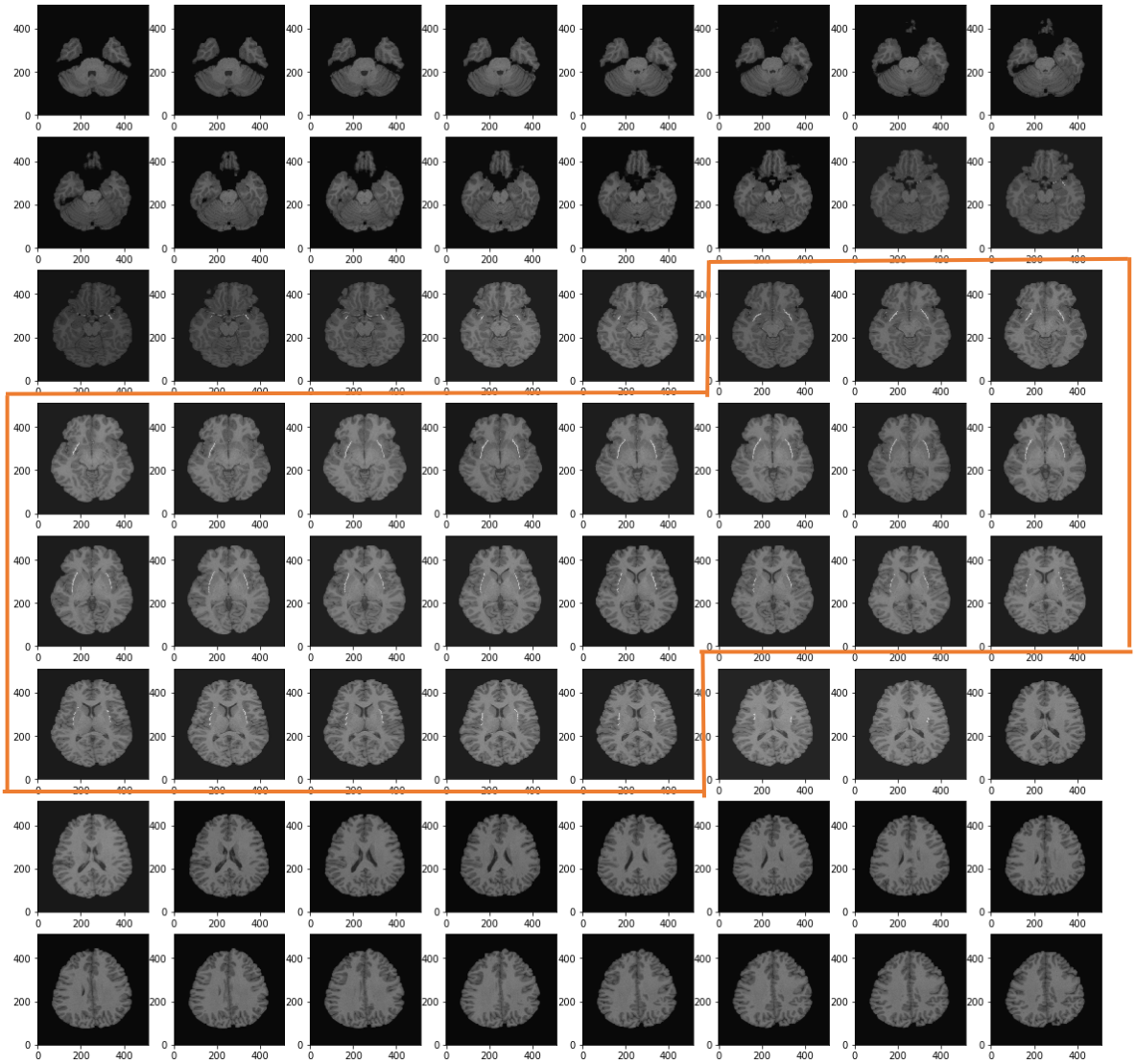

Figure 21 Claustrum Slice Selection

3. Data Augmentation.

   Due to the limited dataset size, we applied some image augmentation techniques such as: image rotation, width and height shifting, and horizontal flip. There are other augmentation techniques available, but we chose these techniques for their popularity and ease of use. The augmentation was done by using the ImageDataGenerator class from the Keras library. Since we are dealing with a

dataset that contains images and labels, it is important to set a random seed to apply the same augmentation technique for the image and its label. We applied the augmentation by a factor of 3 for each image and its label in the dataset, and the augmentation technique was chosen randomly for each image. Figure 22 shows some of the augmented images overlay with the labels, the augmentation is only applied to the selected region of interest (ROI). The next section discusses the ROI step and the reason to apply it.

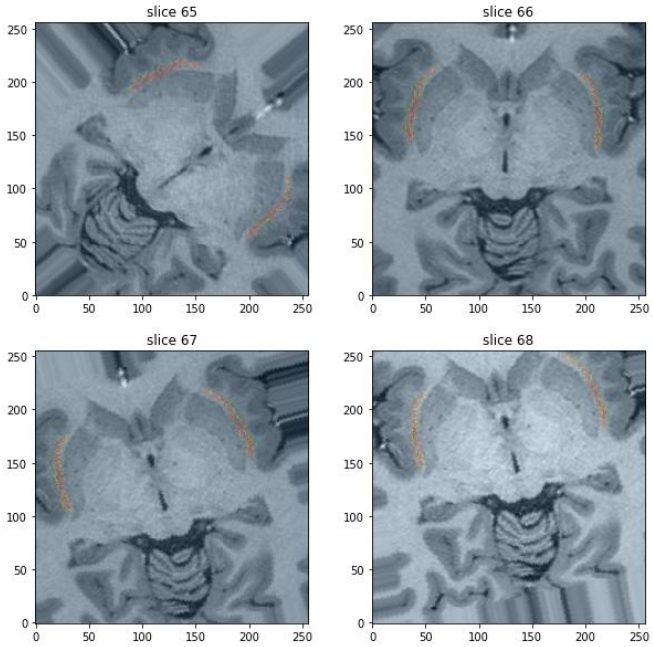

Figure 22 Augmented Images and Labels

4. Region of interest (ROI) selection:

   As per its name, only select or look into the region that interests you. This step is a significant one because it helps tackle the class imbalance problem, and the dataset we have suffers from a huge class imbalance. Class imbalance is a classification problem where classes are not equally represented. In this dataset,

we have two classes: class 0 for the background and class 1 for the claustrum. The image and label dimensions are (512x512) and by counting the pixels on one of the labels we got (0: 261466, 1: 678), which is a massive difference between the two classes. By trying to fit the U-Net model without applying ROI, we got a very high accuracy rate of 99%. This was because the model correctly classified the background class as background and this class percentage is 99% of the whole image. We decided to apply ROI with (256x256) dimensions that only show the claustrum and parts of its surrounding structure as shown in Figure 24. After applying this ROI into the 512x512 image, we got a new count for the pixel classes (0: 64858, 1: 678), which is about a ~75% decrease for the background class only. However, the claustrum was still the underrepresented class, but this helped the model to work and output very good results. Figure 23 below demonstrates the ROI selection on one of the dataset labels.

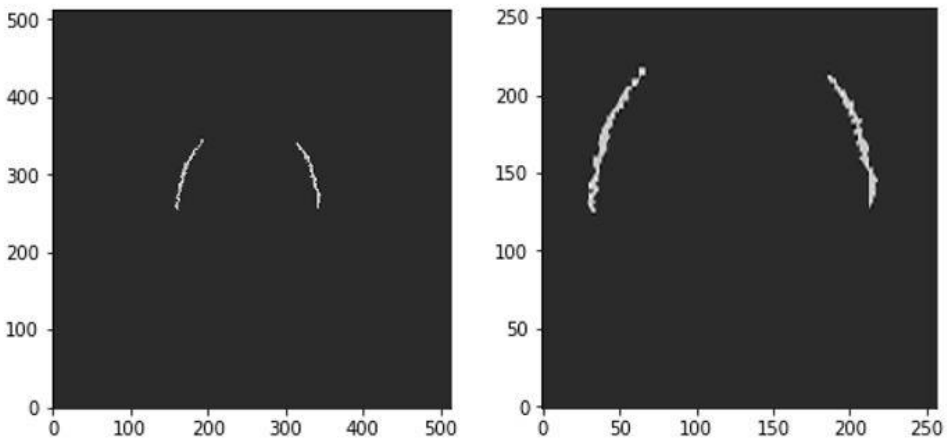

Figure 23 ROI Selection on One of the Labels

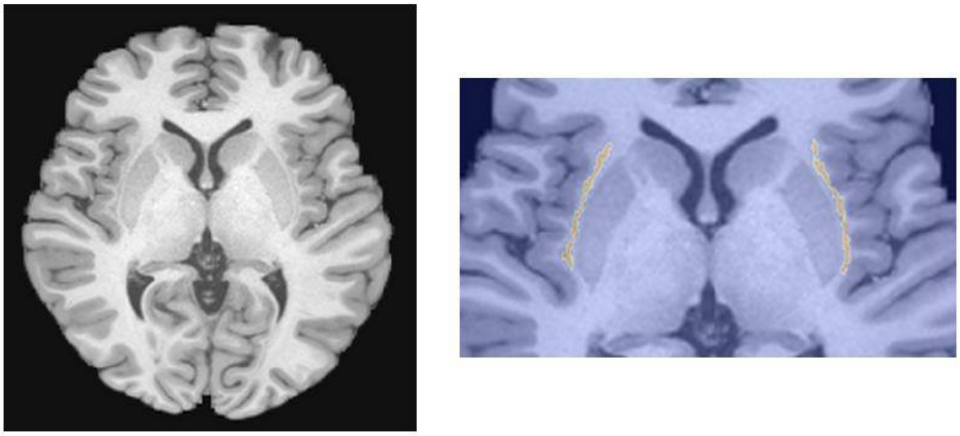

Figure 24 ROI Selection on One of the MR Images

5. Data normalization:

   Normalization, or feature scaling, is an effective popular technique in data preparation for machine learning algorithms. The normalization step only standardizes or rescales the range of data features, and in our case, the images' pixel intensities to the common and most used range of 0 to 1. This step is helpful in training a deep neural network, because having values with a large size affects the model results; hence, standardizing all the values of the pixels intensities help the model train perform better and faster, optimize the model weights while training, and remove the irrelevant aspect of the data [38] [39]. The dataset in this thesis has a huge difference in pixel intensities between slices in each subject, and a different maximum and minimum value for each slice so there is no fixed or standardized range for the pixel intensities. This why we had to normalize all the slices to a new range from 0 to 1. Figure 25 left side illustrates the distribution of pixel intensities for one of the slices and with max. and min. pixel values for the same slice. The applied method to normalize or standardize the pixels' intensities is determined by subtracting the minimum pixel value in the image array from the

image array, and dividing this by the result of the image array maximum value subtracting the image array minimum value. Figure 25 right side shows the result of the applied method on the same slice on the left side of the figure.

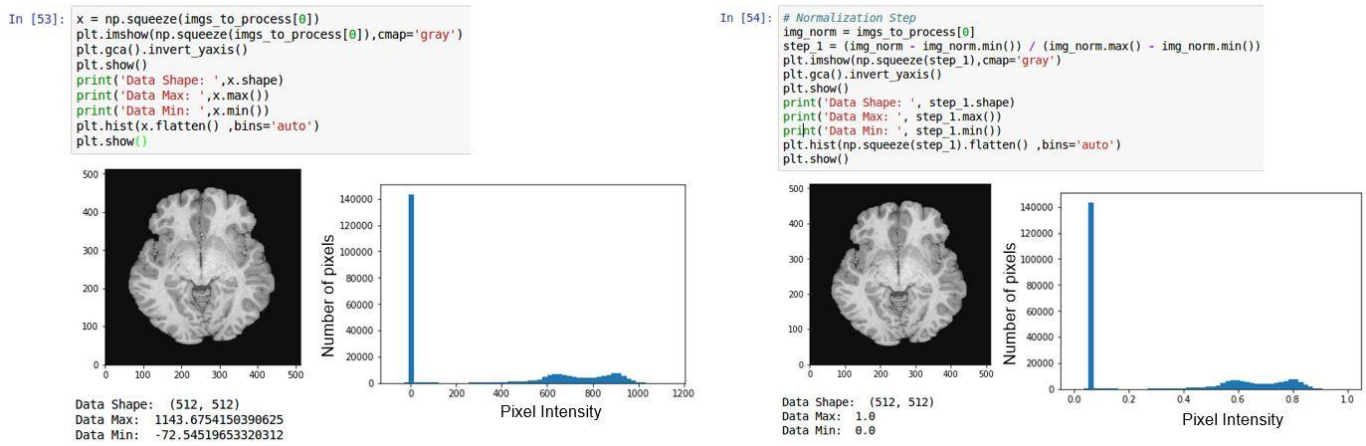

Figure 25 Left Side Before Applying the Normalization Method, Right Side After Normalization Method Applied on Left Side Image

CHAPTER 5

RESULTS and EVALUATION

5.1 Dataset

The dataset used in this thesis comprises 30 manually annotated healthy subjects, the dataset's original modality was MR, and the provided labels were only for the T1-weighed type of scans. The Department of Psychiatry at the University of Missouri-Kansas City provided us with the dataset and the original source of the dataset was the human connectome project. The team in the Psychiatry Department started by applying some processing (such as skull stripping) and used a well-known open source medical software called FreeSurfer. After the processing step, the team manually annotated all 30 MRI subjects using the 3D-Slicer software. In addition, the original dataset format was the Nifti 3D format. This dataset's range of ages was between 22 and 35 with about an equal gender distribution. Table 1 shows the gender and age metadata for the subjects. Furthermore, Table 1 shows the gender and age metadata for the subjects, Furthermore, Figure 26 illustrates the variation between subjects on the same slice's number.

Table 3. MRI Dataset Metadata

Subject	Gender	Age	Subject	Gender	Age	Subject	Gender	Age
105923	F	31-35	191033	F	26-30	162026	F	31-35
106521	F	26-30	192641	F	31-35	162935	M	22-25
108323	F	26-30	198653	M	22-25	164636	M	22-25
109123	M	31-35	204521	F	31-35	175237	F	31-35
113922	M	31-35	353740	M	22-25	185442	M	22-25
133019	F	26-30	358144	F	26-30	707749	M	31-35
140117	F	26-30	406836	F	31-35	725751	M	26-30
149741	M	26-30	568963	F	31-35	891667	M	26-30
156334	F	26-30	679770	M	26-30			
257845	M	26-30	898176	M	31-35			

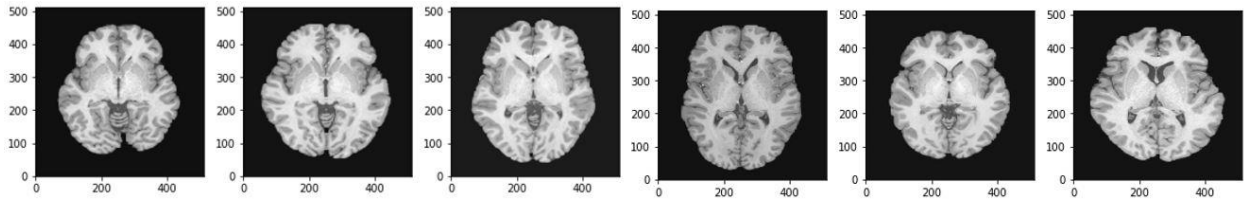

Figure 26 Dataset Slices from Different Subjects to Show the Variations Between the Subjects

5.2 Hardware Configuration

Our proposed method has been implemented using the following system configuration:

- Operating System: Ubuntu 16.04
- Memory: 32 GB RAM
- Processor: Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz
- Graphics Card: GeForce GTX 1080 Ti 11 Gb

5.3 Experiments

5.3.1 Network Hyperparameters

In machine learning, (similarly deep learning), we have a model parameter and hyperparameters. Model parameters are the learnable parameters during the training. However, we want to focus here on the model hyperparameters that need to be a fixed set before the training step as they cannot be learned during the training. These hyperparameters help in tuning the model to fit the training data. There is no specific method or way of setting these hyperparameters, but there are two ways to find the best or the optimal hyperparameters for the model you work with. First, you can run a

grid search to find your optimal parameters from a list of hyperparameters that you create. Second, a random search, which can be defined as a trial and test method, is performed. We next discuss some of the main hyperparameters in detail and what values we set for them.

The network hyperparameters applied in this study are as follows:

- Network Optimizer
- Learning Rate
- Loss Function
- Batch Size
- Number of Epoch
- Dropout

Network Optimizer:

Deep learning typically uses Stochastic Gradient Descent (SGD) to train the network model and update the network weights on iterative base, there are other variation of the SGD: Adam, Adagrad, RMSProp, and many other optimizers. These optimizers help the model to train or converge faster and achieve good results. In our experiments we have mainly relied on Adaptive Moment Estimation (Adam) [28] because it is an effective widely adopted optimizer that can achieve better results in a faster way. In “Adam: A Method for Stochastic Optimization” [28] by Kingma and Ba 2015 the authors have combined the advantages of two other optimizers Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to come up with Adam optimizer.

Learning Rate:

The learning rate is one of the most important hyperparameters in machine learning. It is very effective in tuning the network in order to train the model in deep learning neural networks. The learning rate helps the network optimizer to determine the speed in which to update the network weights, because deep learning models use stochastic gradient descent for network training or other variations, e.g., Adam, Adagrad, and RMSProp optimizers. All these optimizers use a default learning rate in case you don't set one [40]. When using a low learning rate, the optimization of the model will take a longer time to reach the local minimum of the used loss function because the steps towards that point are small. On the other hand, by using a high learning rate, the model will use higher steps to find the local minimum of the loss function. However, because of the higher steps, the model may not converge, and the loss function might get worse. In most of our experiments, we used an Adam optimizer with a learning rate of (0.0001). We found that by using this learning rate, the model learns and trains faster with very good results.

Loss Function:

Loss function is a function defined on a data point, and the ground truth and model prediction are used to measure how well the model is performing. The loss function indicates the model's magnitude of error when classifying input; in our case, this is the classification of each pixel as claustrum or background. Thus, the loss functions help you to know how your model is doing on the training data with respect to the applied

parameter. Because we are training a model with SGD, or any of its variants, we want to find the local minimum of the used loss function. There are several loss functions widely used in deep learning: Cross Entropy Loss, Mean Squared Error (MSE), Mean Absolute Error (MAE), and others. Since we are dealing with a classification problem, the cross-entropy loss is a well-suited loss function for our problem because the output of the model has a probability of between 0 to 1 that a pixel is a claustrum. The cross-entropy loss comes from the probabilistic interpretation of the binary classification. Figure 27 shows how loss decreases for training and validation data with training for many epochs.

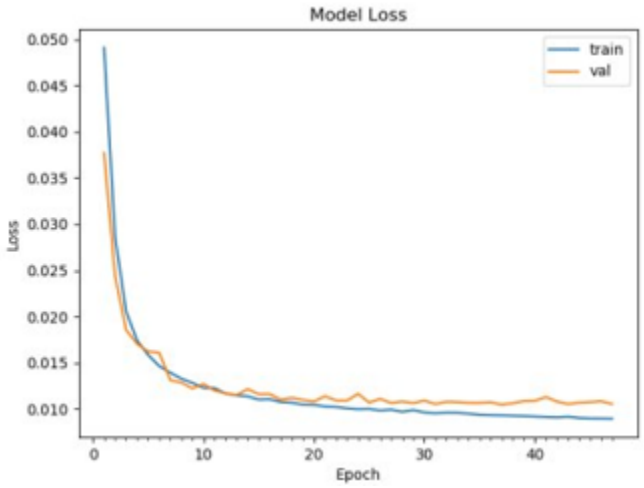

Figure 27 Binary Cross Entropy Loss function

Batch Size:

Batch size is the number of training samples from training images that is given to the network after each iteration or update to the parameters. Three popular options for the batch size are shown below.

- The batch mode — if the batch size is equal to the total training size.

- The mini-batch mode — if the batch size is larger than 1 and less than the total training size.
- The stochastic mode — if the batch size is equal to one.

The U-Net [6] original work uses the stochastic mode in which the gradient and the network weights are updated after each sample. We applied the batch sizes 1 and 4 because they gave the best results.

Number of Epoch:

The number of epoch is the total number of times the dataset is shown to the network. We can also define it as the number of cycles on the entire training data. Generally, there is no method to determine the optimal number of epochs, but a good practice is to use a high number of epochs and apply the Early stopping technique, or model checkpoint, to prevent the overfitting problem. The Early stopping idea is simple: you give a parameter called patience, which, for example, takes an integer number 2, and Early stopping then tracks the loss function. If the function does not improve for two epochs, Early stopping terminates the training process. We adopted the Early stopping technique with different patience levels ranging between 2 and 5.

Dropout:

In deep learning, the goal is to a have more generalized model that can fit your training data and predict the test data correctly. Overfitting is one of the biggest problems in machine learning, and it happens when your model is more complex to learn how to models the training data correctly. There is no unique method to quantify the model

complexity; a popular method is that a model having more parameters is complex. The overfitting problem negatively impacts the model's ability on the test data, and therefore, the dropout technique is applied to reduce the overfitting during the training phase.

The dropout role is to ignore some neurons randomly during training. Technically, a dropout percentage is given to some layers, and during training, only layers with the dropout will ignore some neurons based on the given dropout percentage. There is no optimal dropout percentage, because the learned features differ from a dataset to a dataset. Dropout helps to reduce interdependent learning neurons and this forces the model to learn more features [16]. In this thesis, we applied the dropout technique in two layers from the left side (contracting path) of a U-Net model with a 0.5 percentage. Figure 28 illustrates how the dropout technique works in neural networks. The figure is taken from "Dropout: a Simple Way To Prevent Neural Networks Overfitting." By Srivastava et al., JMLR 2014 [16].

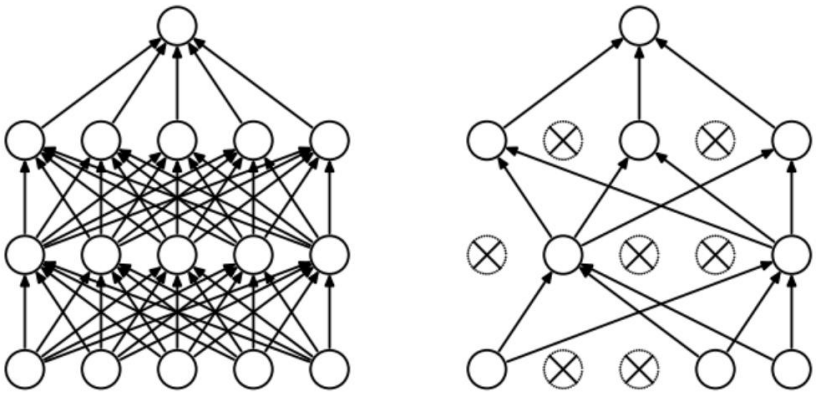

Figure 28 Dropout in Neural Networks Figure Taken from [16]

5.4 Validation and Evaluation

Machine learning validation is a technique to get the error rate of the model. To validate a model, there are a couple of well-known validation methods. We now discuss two popular methods that we employed in this thesis:

- Hold-out Method
- K-Fold Cross Validation

Hold-out method:

In this method, we split the dataset into training and testing, so we have two separate datasets; the training dataset and testing dataset. The choice of the percentage in which we split the data depends on the size of the dataset. Generally, it is better to train models that have more robust features on a larger training dataset. To evaluate the model during training, a ratio of the training dataset is assigned as a validation dataset. This validation dataset is also used to fine-tune your model before using the testing dataset. Technically, splitting the dataset between training and testing data can be 80/20, 70/30, 60/40, or what works best for your model. The validation ratio needs to be assigned before the training phase with a parameter that holds the ratio or the percentage. In this thesis, out of the 29 subjects, we held out 2 subjects for testing and we further split the training in 80/20 training and validation. We randomly selected the testing subjects:

Both subjects 106521 and 358144 shared the same metadata (F: 26-30).

Figure 29 shows an illustration of the hold-out method.

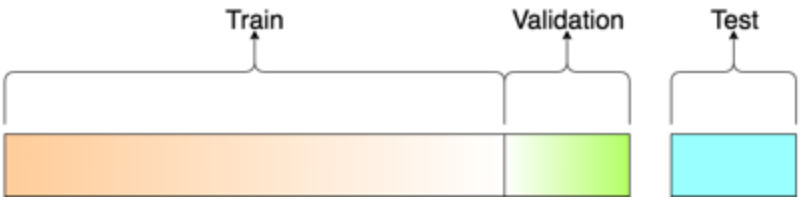

Figure 29 Hold-out Validation Method

K-Fold Cross Validation:

Cross-validation is a method to evaluate the model performance by iterating over the dataset. This method uses the whole dataset for training and testing, the idea is to use a K-fold number, e.g., k=5 and split the dataset into five folds, the training will be repeated five times each time a new fold is used as a testing data. Finally, the error rate is the average of all runs. Figure 30 illustrates the k-for cross validation method with k=5.

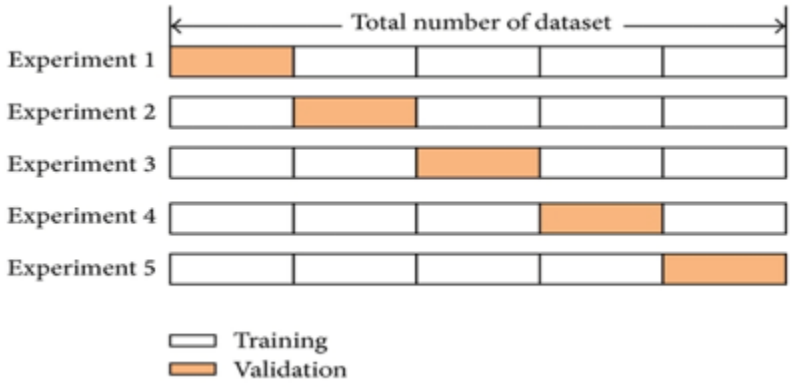

Figure 30 K-Fold Cross Validation Example

The K-fold cross validation method takes a long time because we will test on all the datasets and depending on the choice of K, the larger the K, the longer the training time. We used K=5 in this thesis. We discuss the results of this validation method in the next section.

- Model Evaluation:

The accuracy metric in machine learning is a very common for classification tasks. It measures the percentage of correct prediction, and a larger value indicates a better accuracy. This equation calculates the accuracy of a model:

$$Accuracy = \frac{True\ Postive + True\ Negative}{True\ Postive + False\ Positive + False\ Negative + True\ Negative}$$

In our segmentation case, we need to use a metric that measures the accuracy of the claustrum segmentation. We tried the accuracy metric with the high-class imbalance we have in the dataset, and the model accuracy achieved 99%. Because the model correctly classified the background (negative) pixels. To overcome this problem, we used a popular segmentation metric reported by many research papers in this field. This metric called Dice score or Dice similarity coefficient [41], the Dice score metric measures the overlapping or the intersection between the model output and ground truth label. Dice score equation:

$$Dice\ score = \frac{2\ x\ (P \cap T)}{(P + T)}$$

P = the predicted segmentation, T = ground truth segmentation. The Dice score range from 0 to 1, where 1 means the model segmentation match the ground truth. Figure 31 illustrates Dice score in a more meaningful way.

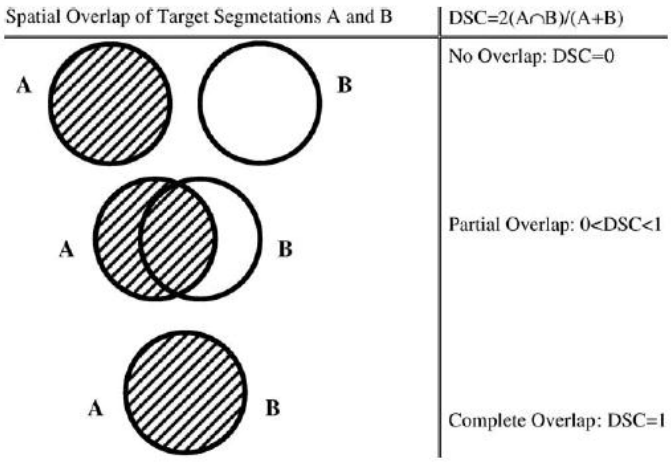

Figure 31 Dice Score

5.5 Model Results

This section discusses the results from the different U-Net models we experimented with. We trained the models on two types of datasets:

1- Not-Augmented Dataset:

1) This dataset contains the original images and labels from the dataset with no augmented images or labels.
2) The total size of the whole dataset before the split was 279 images with 279 labels.
3) We kept two subjects from the total size for testing only. One with a size of 23 images and the other with a size of 23 labels.
4) Before the training phase started, 20% of the training samples were reserved as validation samples. Therefore, the training samples size was 204 and the validation samples size was 52.

2- Augmented Dataset:

- This dataset contains the original images and labels from the dataset plus the augmented images and labels.
- The total size of the whole augmented dataset is 1536 augmented images with 1536 augmented labels.
- We used the same test subjects for the augmented dataset, and separated the test data from the dataset before we applied the augmentation method.
- Before the training phase started, we reserved 20% of the training samples as validation samples, and therefore, the training samples' size was 1228 and the validation samples size was 308.

Generally, all the models showed very good and comparable segmentation results. However, the deeper model performed worse on the not-augmented dataset. Next, we explain the best results from each model:

1- Original Model
    1) Not-Augmented Dataset
    2) Augmented Dataset
2- Simpler Model
    1) Not-Augmented Dataset
    2) Augmented Dataset
3- Deeper Model
    1) Augmented Dataset

<u>1- Original Model:</u>

1) Not-Augmented Dataset:

With this dataset, we discuss two experiments where both of them shared the same hyperparameters, except for batch sizes 1 and 4 for the first and second experiments. Table 4 shows the applied hyperparameters for the two experiments.

Table 4. Hyperparameter for the Not-Augmented Dataset

Optimizer	Adam
Learning Rate	0.0001
Batch Size	1,4
Epoch	40
Validation Set	20%
Dropout – two layers	0.5

Batch Size 1:

The training time was 300 seconds with a total number of 7,760,069 parameters. Because the number of parameters is high, we calculated it with this formula:

Total number of parameters =

*(filter height * filter width * input image channels + 1) * number of filters*

We set the height and width for all filters to (3*3),. The input image channel for the first layer (the input layer) was 1 because we dealt with grayscale images. Successive layers used the same number of filters as from the previous layers and as the input image channels. Finally, the number of filters differed at each layer. In this experiment, the number of filters started at 32 filters for the first layer and then multiplied by 2 for

the next layers, Therefore, at the end, the number of filters at each layer in the contracting path was 32-64-128-256-512. The results with these hyperparameters were the highest in terms of the Dice score; the model achieved a 0.68 Dice score. The model with the batch size of one suffered from the overfitting problem as the loss function started to plateau for the validation set and decreased for the training set as shown in Figure 32. Figure 33 shows the segmentation results from the models in the red color that overlaid on top of the ground truth manual segmentation. The Early stopping parameter was not applied in this experiment because we wanted to see how the model would perform with these hyperparameters.

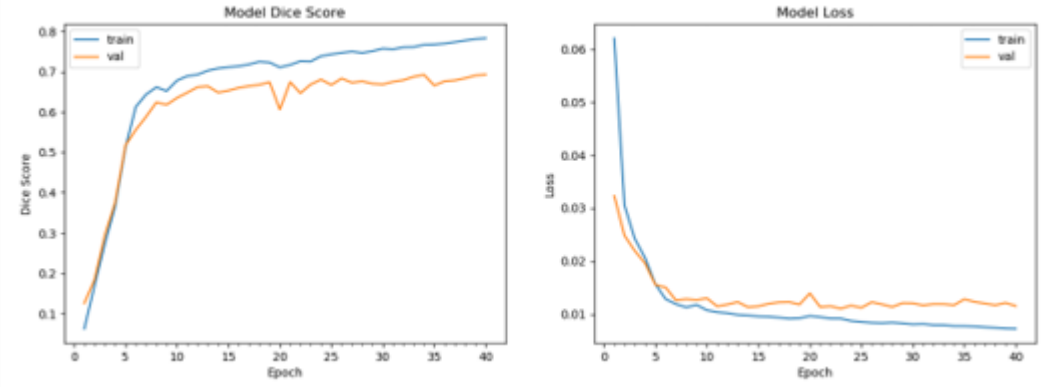

Figure 32 Model Learning for Loss Function and Dice Index for Not-Augmented Dataset with Batch Size 1

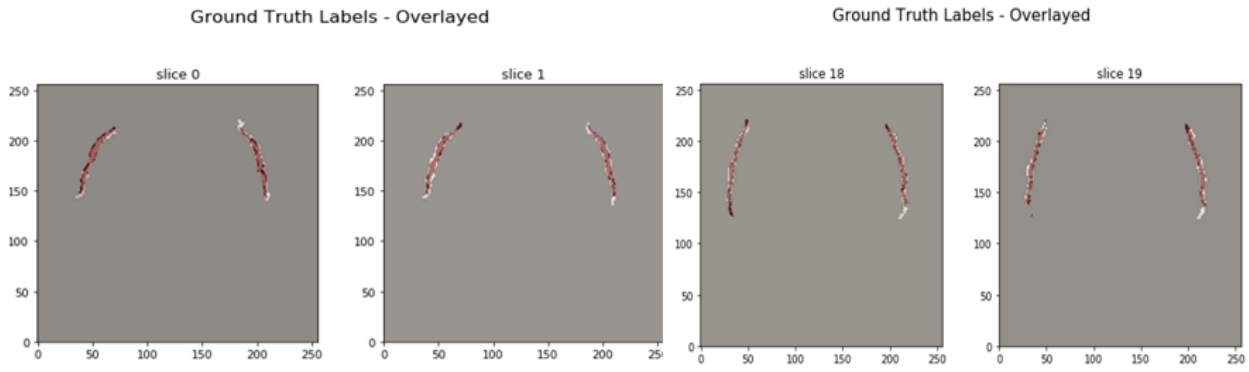

Figure 33 Model Segmentation Results for Not-Augmented Dataset with Batch Size 1. Red Foreground is model segmentation, and Beige Background is Ground Truth segmentation

Batch Size 4:

The batch size was set to 4 because we wanted to how the U-Net model performed by increasing the batch size. The model loss function performed better compared to the batch size of 1 (Figure 32). In terms of the Dice score, the model was only able to achieve 0.48 and the training time was 200 seconds. The segmentation results, shown in Figure 35, were still comparable to the results from batch size 1, and because the Dice score penalizes for false positive, this may have been the reason for getting a lower Dice score. Therefore, a further review to the manual segmentation is required to validate the model output. Furthermore, if we compare the segmentation between the two models at Figure 33 and Figure 35 the model with batch size 4 at Figure 35 tried to identify some small pixels around the claustrum as a claustrum thus, the model with batch size 1 output better quality segmentation results.

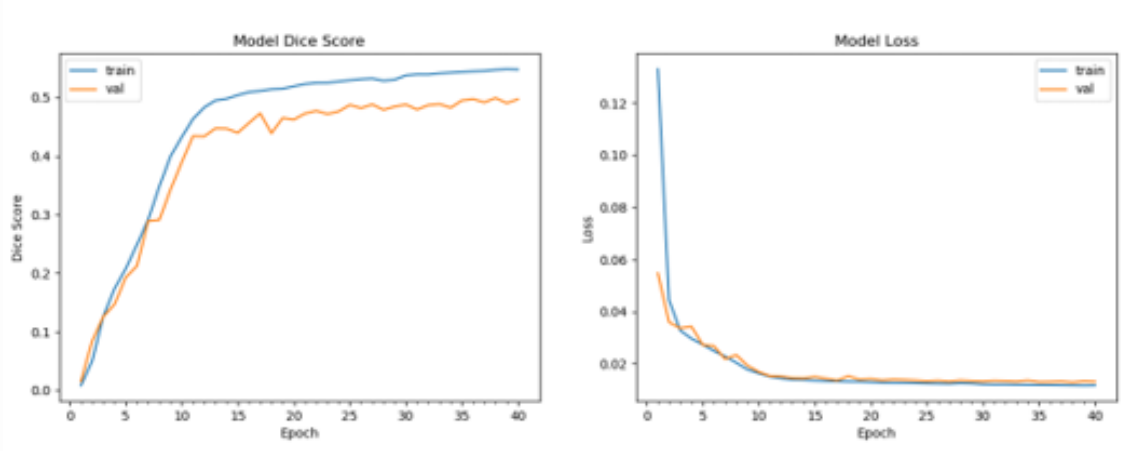

Figure 34 Model Learning for Loss Function and Dice Index for Not-Augmented Dataset with Batch Size 4

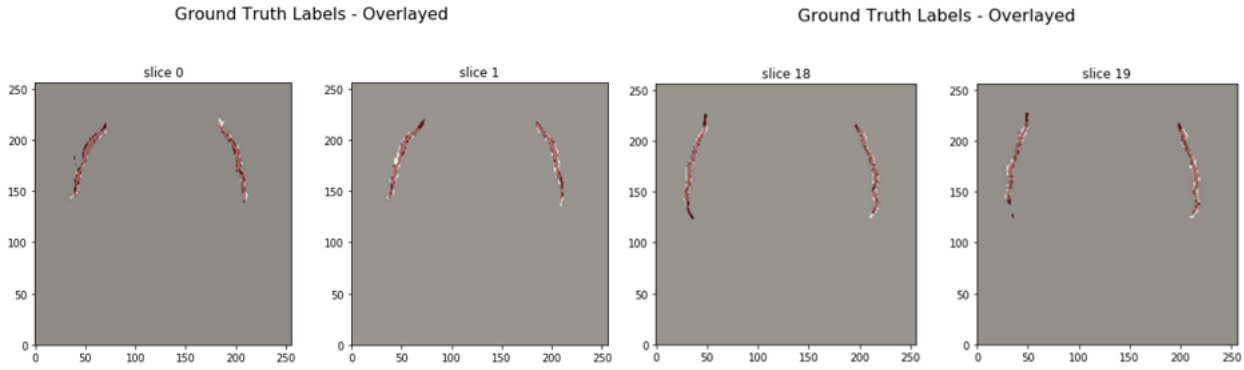

Figure 35 Model Segmentation Results for Not-Augmented Dataset with Batch Size 4. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

2) Augmented Dataset:

We followed the same experiment that we applied on the previous not augmented dataset. All of the hyperparameters were the same and we tried batch sizes 1 and 4.

Batch Size 1:

The training time was 856 seconds, more than double the time of the previous not augmented dataset and due to the differences in dataset size. The number of parameters was the same because we used the same filter numbers and sizes. The

model achieved a Dice score of 0.66 at epoch 19 and stopped the training because the Early stopping method was applied on both the loss functions and Dice score. Figure 36 shows the model learning. If we compare the Dice score at the 5th epoch in this figure and Figure 32, we see that the augmented dataset model achieved more than a 0.6 Dice score. On the other hand, with the not augmented dataset, the model did not achieve more than a 0.4 Dice score. Hence, the U-Net model achieved a higher Dice score with a smaller number of epochs when trained with more data.

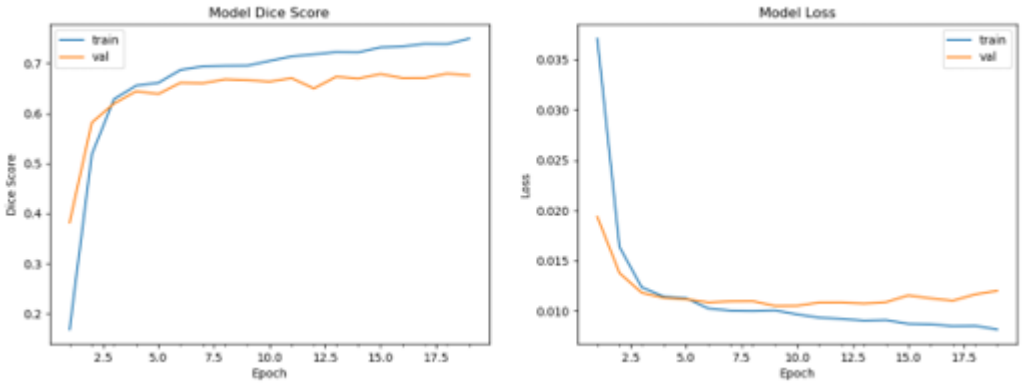

Figure 36 Model Learning for Loss Function and Dice Index for Augmented Dataset with Batch Size 1

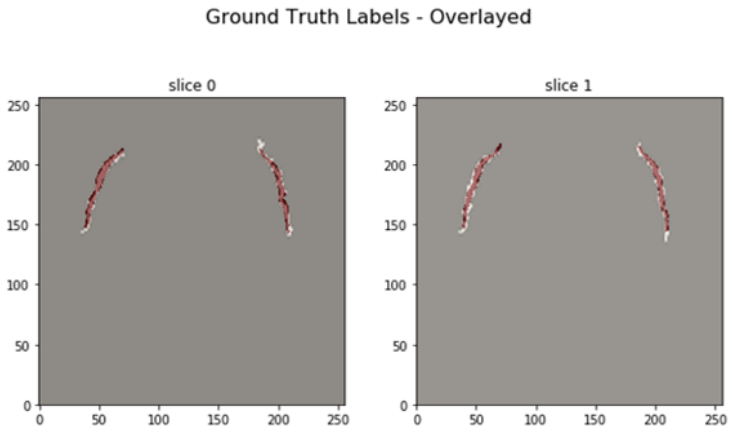

Figure 37 Model Segmentation Results for Augmented Dataset with 1 Batch Size. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

Figure 37, the segmentation output of this model is very comparable to the same model without augmentation. Both models got higher than a 0.6 Dice score with only a small margin between them.

Batch Size 4:

Increasing the batch size from 1 to 4 with the augmented dataset reduced the training time to 787 with a 69 second difference. Similar to the not-augmented dataset, increasing the batch size reduced the overfitting problem for loss function. Figure 38 shows the model learning that compared this model to the previous one. Figure 36, with a batch size 1, shows how the overfitting problem reduced the performance of the previous model as the model stopped training at an earlier epoch where this model stopped at epoch 27. If we compare the segmentation results in Figure 39 and Figure 37, we can identify that slice 0 at Figure 39 is a better-quality segmentation because

the model with the batch size of 4 tried to identify some small left parts near the claustrum.

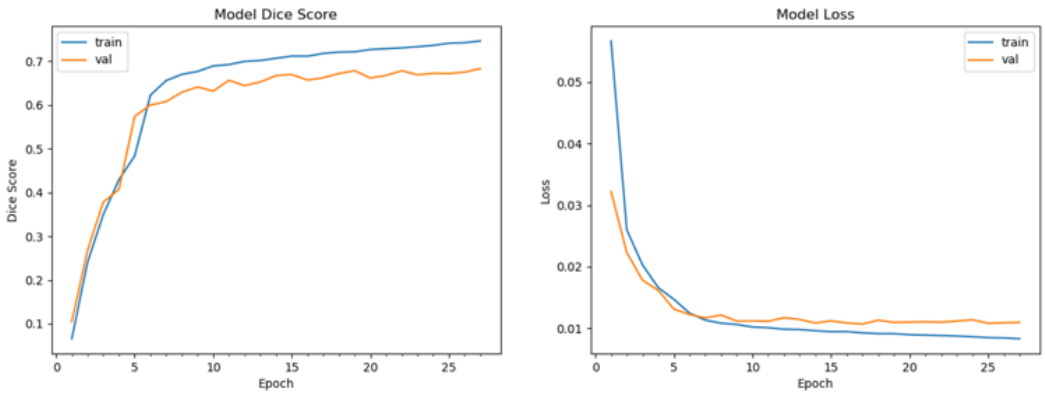

Figure 38 Model Learning for Loss Function and Dice Index for Augmented Dataset with Batch Size 4

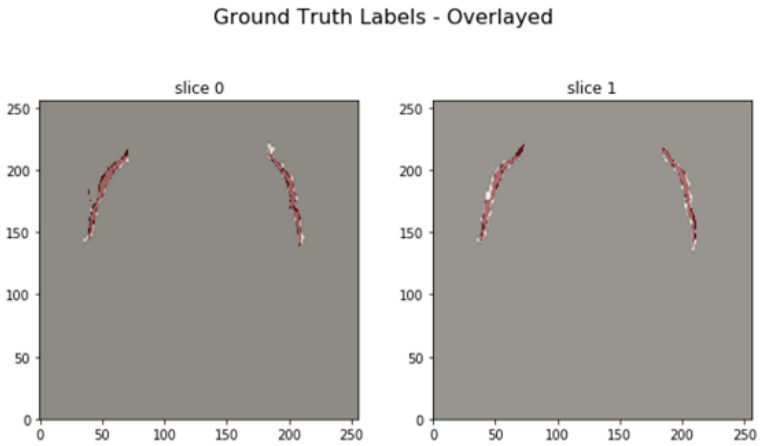

Figure 39 Model Segmentation Results for Augmented Dataset with Batch Size 4. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

Table below summarize the original model results:

Table 5. Results Summary, Original Model

Dataset	Batch Size	Dice Score
Not-Augmented Dataset	Batch size 1	0.68
	Batch size 4	0.48
Augmented Dataset	Batch size 1	0.66
	Batch size 4	0.67

2- Simpler Model:

This model is simpler than the original model because we removed two layers from the original one layer from the left side of the network's contracting path, and the second one from the right side of the expanding path, as shown in Figure 19. By simplifying the original design, we reduced the complexity of the model, and hence, reduced the overfitting problem. Next, we discuss some segmentation results from the same two datasets the not augmented and the augmented datasets.

1) Not-Augmented Dataset

Batch Size 1:

The simpler model training with the batch size of 1 took 308 seconds to train, with 1,925,573 numbers of parameters because the choice of the filter numbers was 32-64-128-256. The model trained with 80 epochs achieved a 0.67 Dice score at epoch

46, and the Early stopping method stopped the model training at epoch 46 to further stop overfitting. Figure 40 shows the model loss function and Dice score learning.

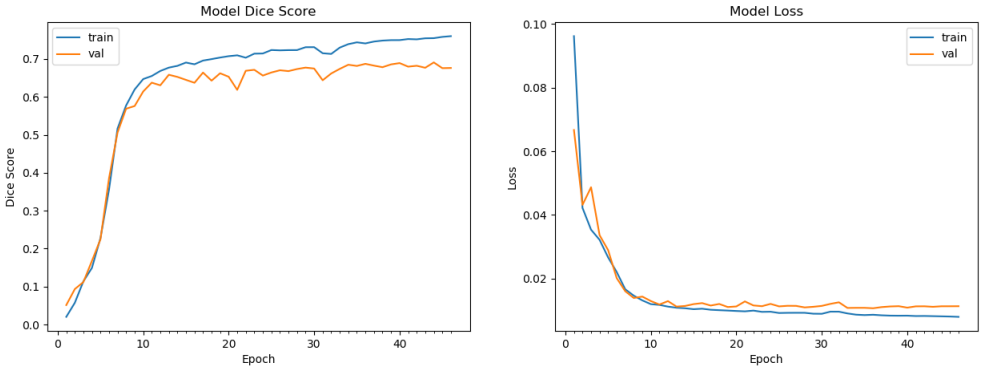

Figure 40 Simpler Model Learning for Loss Function and Dice Index on Not-Augmented Dataset with Batch Size 1

In comparison with all previous models with the batch size of 1, the simpler model is the best model that tackled the overfitting problem. Figure 40 shows the loss function for the simpler model, which we can compare it to Figure 32 and Figure 36. The segmentation results in Figure 41 show that the model is outputting comparable results to previous models.

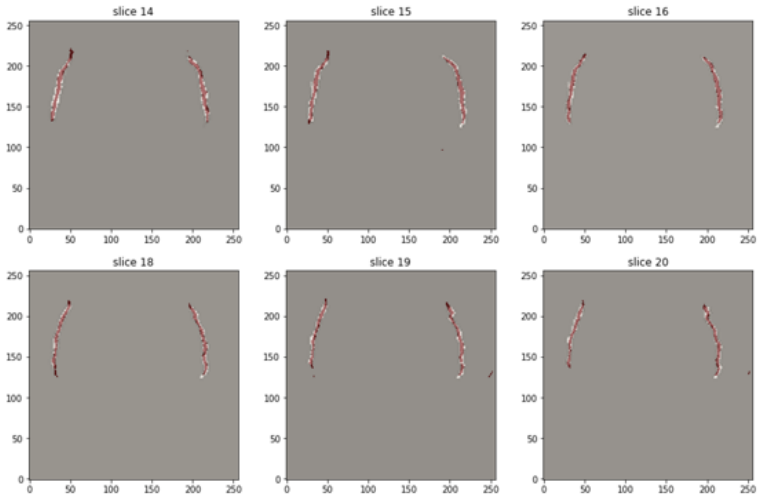

Figure 41 Simpler Model Segmentation Results for Augmented Dataset with Batch Size 1. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

Batch Size 4:

Similarly, with batch size 4, we tried to see if the model segmentation output improved by increasing the batch size during training. All hyperparameters and filter sizes are the same as in batch size 1 for the same model; the only change is the batch size. With batch size 4, The Early stopping method stopped the model training at epoch 48 and the training time was 222 seconds. The model loss function learning was similar to the loss function learning in Figure 40 of the simpler model with batch size 1. Clearly, this model reduced the overfitting problem in Figure 42's model loss. The model was able to achieve a 0.65 Dice score, but trying to classify other objects around the claustrum as claustrum, as in Figure 43, resulted in batch size 1 showing a better quality.

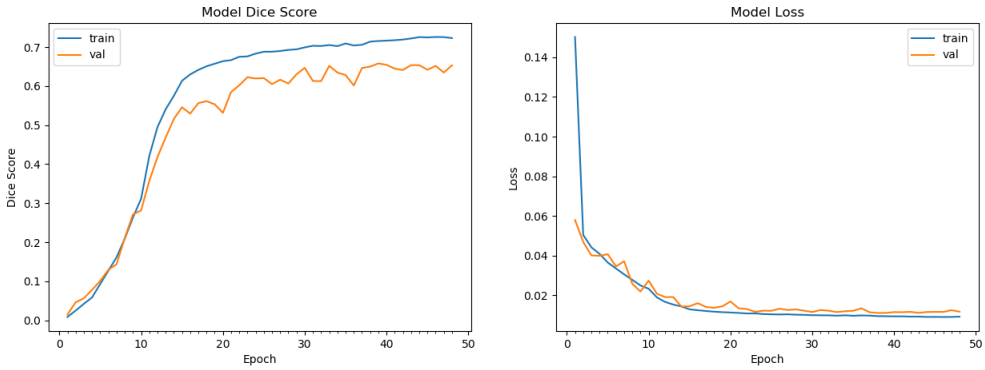

Figure 42 Simpler Model Learning for Loss Function and Dice Index on Not-Augmented Dataset with Batch Size 4

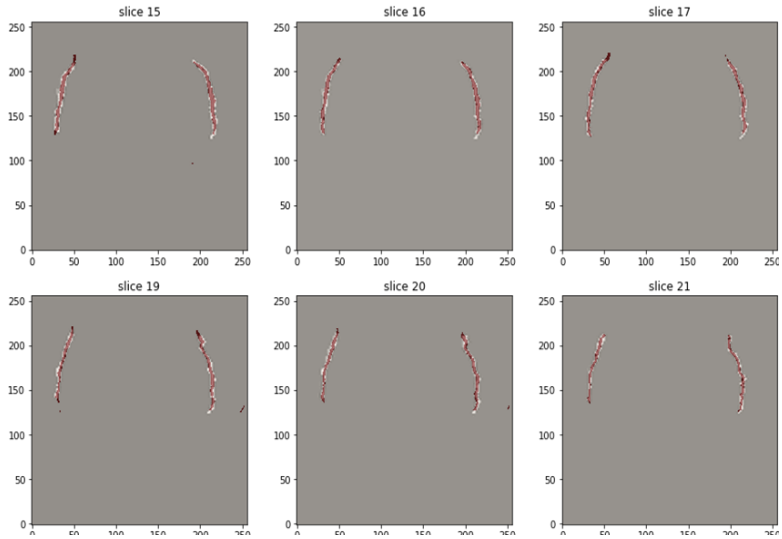

Figure 43 Simpler Model Segmentation Results for Augmented Dataset with Batch Size 4. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

2) Augmented Dataset:

The simpler model performed slightly better on the augmented dataset, and we now discuss a similar experiment to the not-augmented dataset. We start with a batch size of 1 and then a batch size of 4.

With a batch size of 1 as the model,140 epochs, and 482,021 numbers of parameters, the Early stopping method stopped the model training at epoch 47 with 1117 seconds of training time. This model took a longer time to train even though the number of parameters was by far the lowest when compared with all of the other models. The filter size changed to a 16-32-64-128 filter size for each layer. Similar to previous simpler models, the overfitting problem was reduced for the loss function and even for the Dice accuracy, as shown in Figure 44. The segmentation results were of a better quality as in the previous batch size 1 results. Figure 45 shows the segmentation output for this model.

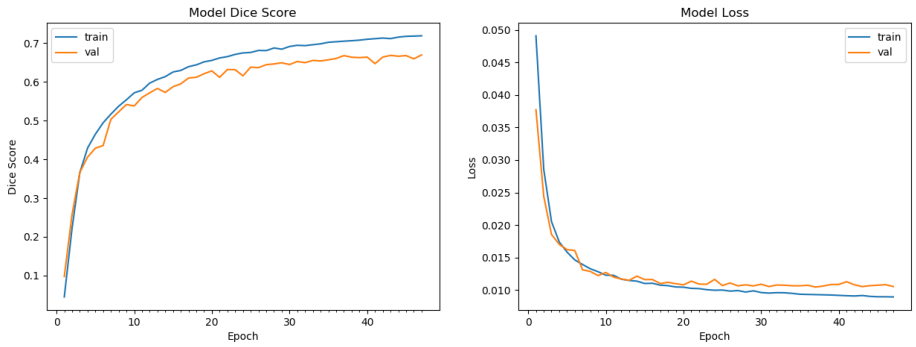

Figure 44 Simpler Model Learning for Loss Function and Dice Index for Augmented Dataset with Batch Size 1

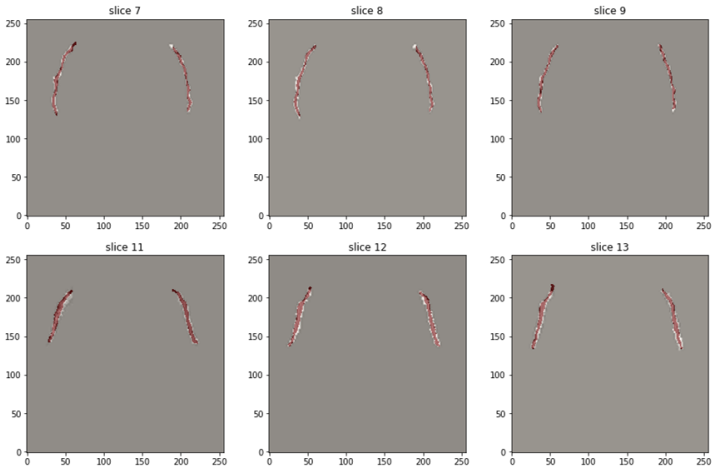

Figure 45 Simpler Model Segmentation Results for Augmented Dataset with Batch Size 1. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

Table below summarize the Simpler model results:

Table 6. Results Summary, Simpler Model

Dataset	Batch Size	Dice Score
Not-Augmented Dataset	Batch size 1	0.67
	Batch size 4	0.65
Augmented Dataset	Batch size 1	0.65
	Batch size 4	0.66

3- Deeper Model:

This model is deeper than the original model because we added two extra layers to the original model, one layer was added to the left side of the network (the contracting

path), and a second one was added to the right side (the expanding path) as seen in Figure 19's deeper model. By adding extra layers to the original design, we increased the complexity of the mode. We will next discuss some segmentation results from the augmented dataset only. Because the model could only segment the claustrum on this dataset, and that's the case for deeper models they require a higher amount of data to learn from.

- Augmented Dataset:

The deeper model was only trained on the augmented dataset, with a batch size of 1. The training time was 477 seconds with 7,775,589 parameters, because the choice of filters number was 16-32-64-128-256-512. We trained the model with 30 epochs, which was able to achieve a 0.67 Dice score at epoch 13. The Early stopping method stopped the model training at epoch 13 to stop further overfitting. Figure 46 shows the model loss function and Dice score learning. Figure 39 and Figure 47 shows the segmentation results for this model, and similarly, the model with batch size 1 output quality segmentation results.

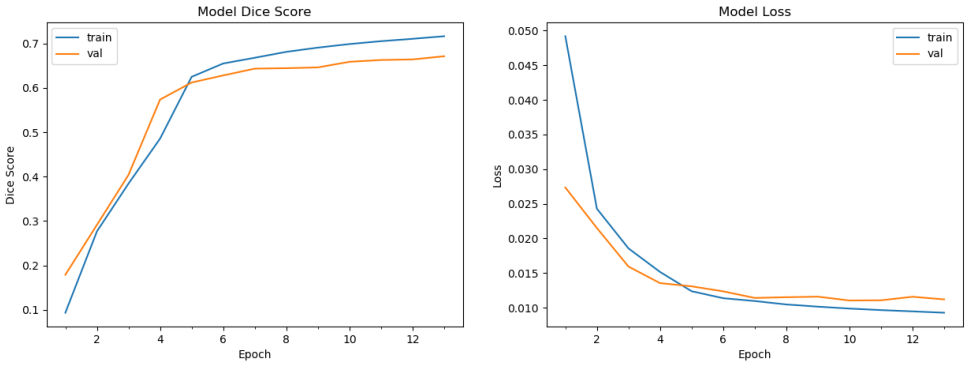

Figure 46 Deeper Model Learning for Loss Function and Dice Index for Augmented Dataset with Batch Size 1

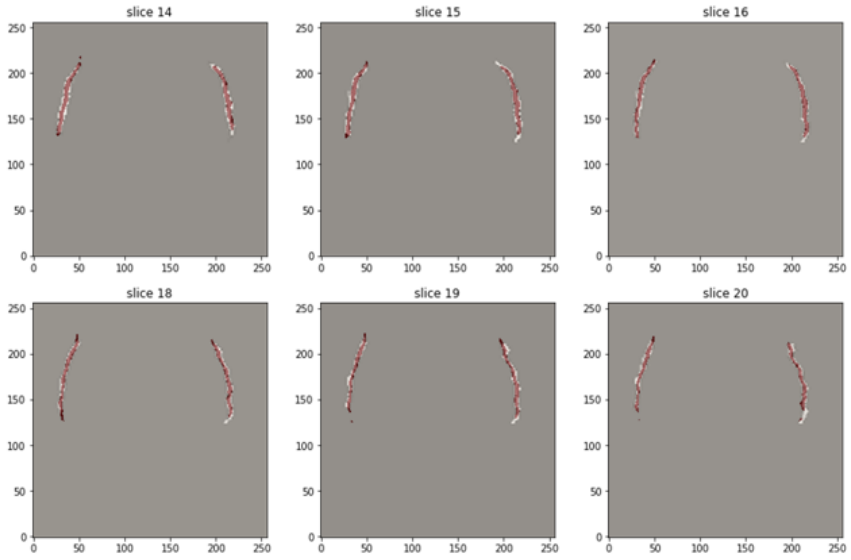

Figure 47 Deeper Model Segmentation Results for Augmented Dataset with Batch Size 1. Red Foreground is model segmentation, Beige Background is Ground Truth segmentation

CHAPTER 6

PROPOSED MODEL APPLICATION

To make it easier to work with our proposed solution, we developed a web application that has an easy to use interface and provides the segmentation results in less than 3 seconds. The web application currently supports only the test data we have. We also added new and unlabeled test data provided by Dr. Kang in the Department of Psychiatry at the University of Missouri-Kansas City. The new test source is the same original source for all the datasets in the human connectome project. The application portion consists of three different parts:

1. Server Application
2. U-Net Model
3. Client or Web Application

The proposed application can provide a tremendous amount of help to doctors or other practitioners, the segmentation results can be a good start or a helpful guide while working on manual segmentation. Part of the future work is to provide the user the ability to upload an image from his own data to segment the claustrum.

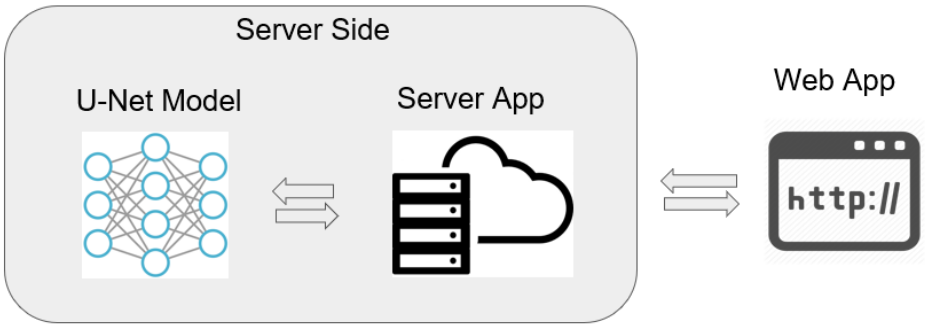

Figure 48 Proposed Application Architecture

1. Server Application:

The server application was built using Python language, and it is responsible for running the U-Net model online, but only if a request is received. The server mainly works with a POST requests and reads the incoming data (images) from the client application as base 64. The server decodes the received image and sends it to the U-Net model to segment the claustrum and save the segmentation output as an image. It also saves a new image, which is the original image overlaid with segmentation output.

2. U-Net Model:

The U-Net model works to predict the segmentation of an image. No training occurs, because we are loading the saved weights from one of the best models we had. The ability to load the previous weights and predict that the segmentation gives the application the ability to provide the segmentation output in seconds. Thanks to Keras library developers for making this library and providing a plenty of awesome features.

3. Client Application:

The client application is a webpage that has a JavaScript function to send the selected image to the server. If an image is sent successfully and the client receives the server confirmation, the results of the segmentation from the server will appear. Figure 49 shows the webpage interface, and Figure 50 illustrates the results received from the server.

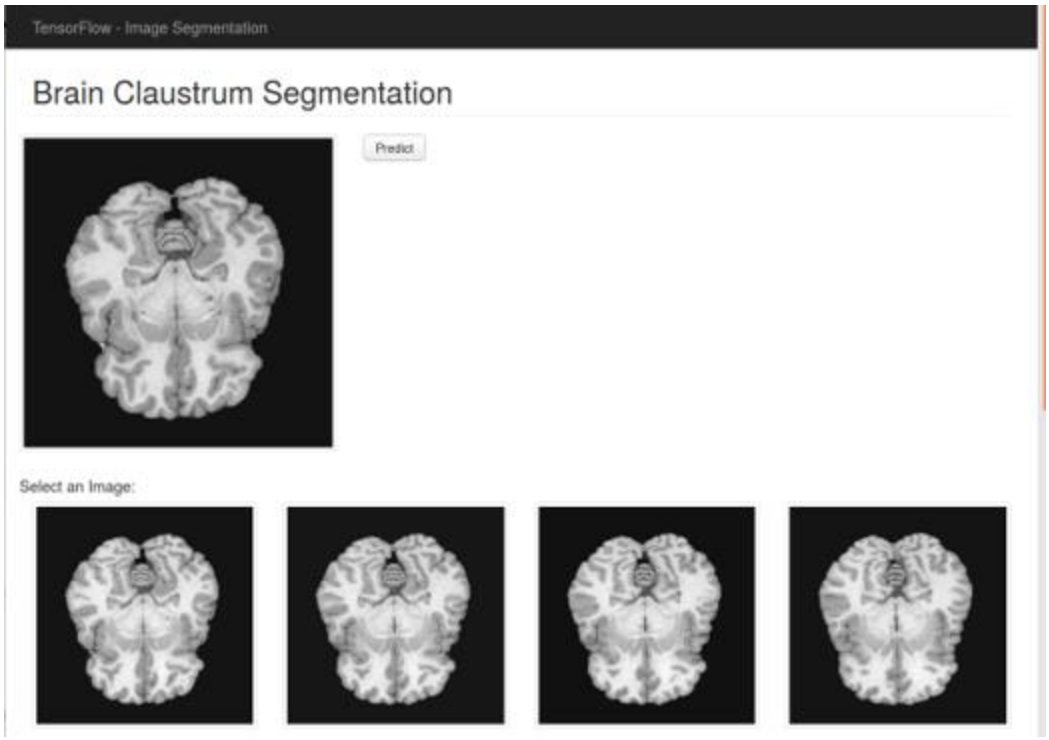

Figure 49 Client Side (Web Application)

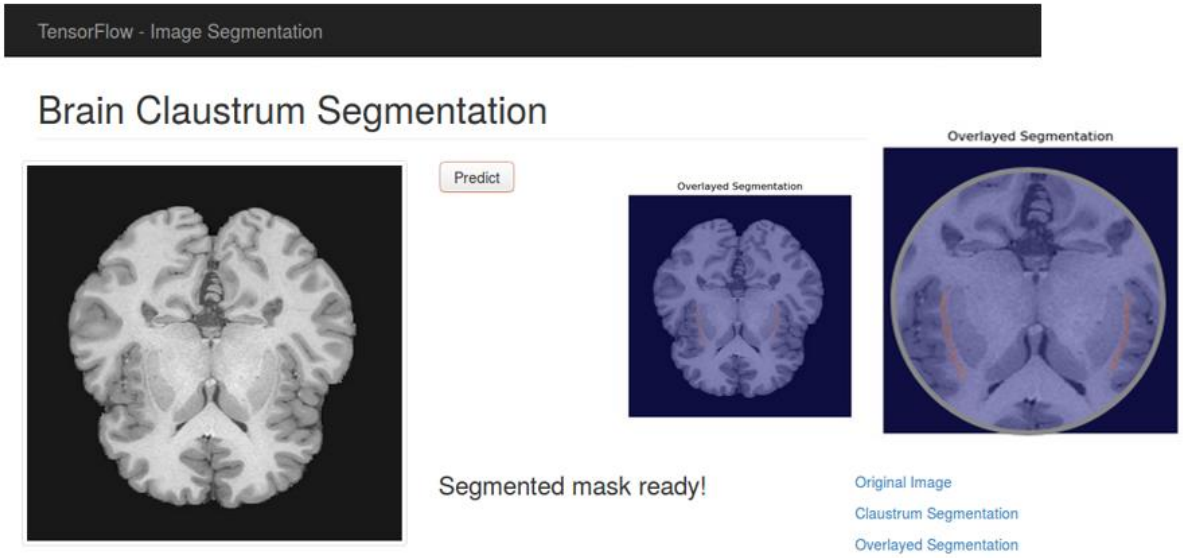

Figure 50 Web Application after Receiving the Results from the Server, with Zoom in Ability

CHAPTER 7

CONCLUSION and FUTURE WORK

7.1 Conclusion

The analysis of medical image data with deep learning convolutional neural networks has started to gain researchers' attention. The presented framework to semi-automate the segmentation of the brain-claustrum on MR images, can have a profound impact on the procedure of manual segmentation. The field of medical imaging has many challenges, and we successfully tackled most of the challenges faced in this thesis by exploring the current and previous related work.

CNNs proved to be the future and the current dominant method in computer vision. Furthermore, U-Net model capabilities in medical imaging are showing very good quality segmentation results as the choice and the modifications to the U-Net model can vary from dataset to dataset. The three U-Net models we experimented with produced comparable segmentation results. We also found that when training these models with batch size 1, the model suffered from an overfitting problem but achieved the highest performance. When training a U-Net model with binary cross entropy as a loss function and a Dice score to measure the accuracy, the model performed better if trained with smaller batch sizes.

The class imbalance problem is one of the major problems reported in the field of medical imaging. The dataset used in this thesis suffers from high-class imbalance. The applied solution to tackle the problem is the ROI selection only; this solution helped to reduce the imbalance ratio but not to the desired balance.

7.2 Future Work

The points stated below are the future work to this thesis. We plan to implement these points to improve the model segmentation results and the framework reliability. Some of the future work can result in profound results for the medical field and will also require the assistance of medical experts.

Improve the current segmentation results by performing the following:

- Applying the grid search to find the best possible network parameters.
- Further reducing the high-class imbalance.
- Employing 3D segmentation of Dicom or Nifti format.
- Finding the claustrum size or volume, and the claustrum pattern based on its structure.
- Finding the relation between gender, diseases, and claustrum size of shape by exploring the un-supervised clustering techniques.

REFERENCES

[1] H. Zhu. “Medical image processing overview”, *University of Calgary, Summer School Program-Introduction to Mathematical Medicine, held at the University of Waterloo,* 2003.

[2] J. Miano. *Compressed image file formats - JPEG PNG GIF XBM BMP*, USA:Addison Wesley Longman Inc, 1999.

[3] M. Larobina. and L. Murino, "Medical image file formats", *J Digit Imaging*, vol. 27, no. 2, pp. 200-206, Apr 2014.

[4] D. L. Pham. C. Y. Xu. and J. L. Prince, "A survey of current methods in medical image segmentation", *Annu. Rev. Biomed. Eng.,* vol. 2, pp. 315-337, 2000.

[5] J. Long, E. Shelhamer, and T. Darrell. “Fully convolutional networks for semantic Segmentation”. *In Computer Vision and Pattern Recognition,* pp. 3431-3440, 2015.

[6] Y. LeCun, Y. Bengio. and G. Hinton, "Deep learning", *Nature,* vol. 521, pp. 436, May 2015.

[7] J. Duncan. and N. Ayache, "Medical image analysis: Progress over two decades and the challenges ahead", *IEEE Trans. Pattern Anal. Machine Intell.,* vol. 22, pp. 85-106, Jan. 2000.

[8] F.C. Crick and C. Koch. “What is the function of the claustrum?.” *Philosophical Transactions of the Royal Society B: Biological Sciences vol.* 360, no. 1458 pp. 1271-1279, 2005

[9] J. Jones. Claustrum [online] Radiopaedia.org. Available at: https://radiopaedia.org/articles/claustrum

[10] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi. and F. A. G. Osorio. "A deep Learning architecture for image representation visual interpretability and automated basal cell carcinoma cancer detection" in *Medical Image Computing and Computer Assisted Intervention-MICCAI 2013*, New York:Springer, pp. 403 410, 2013.

[11] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P. Jodoin, and H.Larochelle. “Brain tumor segmentation with deep neural networks”, *Medical image analysis 2017*, pp. 18-31,2015.

[12] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro and E. Shelhamer. "cuDNN: Efficient primitives for deep learning", *Computing Research Repository (CoRR)*, Arxiv.org, 2018.

[13] Y. Bengio, A. Courville, and P. Vincent. "Representation learning: A review and new perspectives", *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 35, no. 8, pp. 1798 1828, Aug. 2013.

[14] P. J. Werbos. "Backpropagation through time: What it is and how to do it", *Proc. IEEE*, vol. 78, no. 10, pp. 1550-1560, 1990.

[15] Y. LeCun. "LeNet-5 convolutional neural networks", Internet: http://yann.lecun.com/exdb/lenet., 2015.

[16] D. Scherer, A. Müller, and S. Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition.”, *In International Conference on Artificial Neural Networks*, 2010.

[17] J. Shi, and J. Malik. "Normalized cuts and image segmentation", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905, Aug. 2000.

[18] A. Arnab, and P. H. S. Torr. "Pixelwise instance segmentation with a dynamically instantiated network", *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pp. 5, 2017.

[19] V. Badrinarayanan, A. Kendall and R. Cipolla. "SegNet: A deep convolutional encoder-decoder architecture for image segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 12, pp. 2481-2495, 2017.

[20] O. Ronneberger, P. Fischer, and T. Brox. "U-net: Convolutional networks for biomedical image segmentation", *Proc. Int. Conf. Medical Image Comput. Comput.-Assisted Intervention*, pp. 234-241, 2015.

[21] J. Wang, and L. Perez. “The effectiveness of data augmentation in image classification using deep learning” No. 300. *Technical report Stanford university, 2017*.

[22] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet classification with deep convolutional neural networks", *Proc. Neural Information and Processing Systems,* pp. 1097-1105, 2012.

[23] A. Asperti, and M. Claudio. “The Effectiveness of data augmentation for detection of gastrointestinal diseases from endoscopical images.”, *Bioimaging.,* 2017.

[24] S. C. Wong. A. Gatt. V. Stamatescu. And M. D. McDonnell, “Understanding data augmentation for classification: when to warp?”, *Digital Image Computing:*

Techniques and Applications (DICTA), 2016 International Conference on , pp. 1-6. IEEE, 2016.

[25] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. *Hypermedia image processing reference*. England: John Wiley & Sons Ltd, 1996.

[26] B. Kayalibay, G. Jensen, and P. V. D. Smagt. "CNN-based segmentation of medical imaging data", *Computing Research Repository (CoRR),* vol. abs/1701.03056, 2017.

[27] K. Kushibar, S. Valverde, S. Gonzalez-Villa, J. Bernal, M. Cabezas, A. Oliver and X. Llado. "Automated sub-cortical brain structure segmentation combining spatial and deep convolutional features", *Computing Research Repository (CoRR),* abs/1709.09075.

[28] P. Christ, F. Ettlinger, F. Grun, M. Elshaera, J. Lipkova, S. Schlecht, F. hmaddy, S. Tatavarty, M. Bickel, P. Bilic, M. Rempfler, F. Hofmann, M. Anastasi, S. Ahmadi, G. Kaissis, J. Holch, W. Sommer, R. Braren, V. Heinemann and B. Menze. "Automatic Liver and Tumor Segmentation of CT and MRI Volumes using Cascaded Fully Convolutional Neural Networks", *Computing Research Repository (CoRR),* abs/1702.05970.

[29] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition", In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. "Going Deeper with Convolutions", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015, 1-9.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large scale image recognition", *Computing Research Repository (CoRR),* abs/1409.1556.

[32] R. Girshick, J. Donahue, T. Darrell and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation", *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580-587, 2014.

[33] R. Girshick. "Fast R-CNN", *IEEE International Conference on Computer Vision (ICCV),* pp. 1440-1448, 2015.

[34] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks", *Proc. 28th Int. Conf. Neural Inf. Process. Syst.,* 2015.

[35] K. He, G. Gkioxari, P. Dollar and R. Girshick. “Mask R-CNN”, *IEEE International Conference on Computer Vision (ICCV),* pp. 2980-2988, 2017.

[36] R. Roy, M. Pal, and T. Gulati. “Zooming digital images using interpolation Techniques”, *International Journal of Application or Innovation in Engineering & Management (IJAIEM),* Volume 2, Issue 4, 34-45, 2013, ISSN 2319 – 4847.

[37] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization", *Computing Research Repository (CoRR)*, Arxiv.org, 2014.

[38] I. Mohamad and D. Usman, "Standardization and its effects on k-means clustering algorithm", *Research Journal of Applied Sciences, Engineering and Technology,* vol. 6, no. 17, pp. 3299-3303, 2013.

[39] W. Sarle. "comp.ai.neural-nets FAQ, Part 2 of 7: Learning", [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part2/.

[40] L. N. Smith. "Cyclical learning rates for training neural networks", *In Applications of Computer Vision (WACV),* 2017 IEEE Winter Conference on (pp. 464-472).

[41] L. R. Dice. "Measures of the amount of ecologic association between species", *Ecology*, vol. 26, no. 3, pp. 297-302, July 1945.

VITA

Ahmed Albishri completed his bachelor's degree in Information Technology from the University of King Abdul-Aziz, Saudi Arabia. He then worked as a teaching assistant in Saudi Electronic University, Saudi Arabia, for ten months. Then he received a scholarship to study for his master's degree in computer science. He started his master's in computer science at the University of Missouri-Kansas City (UMKC) in August 2016, with an emphasis on Data Science.