ADDIS ABABA UNIVERSITY

Addis Ababa Institute of Technology

School OF Electrical and Computer Engineering

Volumetric Segmentation of Brain Tumors Using the 3D U-Net Architecture

A thesis submitted to School of Graduate Studies and Research in Partial Fulfillment of the Requirement for the Masters Degree

Prepared By: Mahlet Alemseged Bitew
Advisor: Dr. Fetene Mulugeta

September, 2022

Declaration

I, the undersigned, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Name: Mahlet Alemseged Signature: ______________________

Place: Addis Ababa, Ethiopia

Date of Submission: ___________________________

This thesis is submitted for examination with my approval as university advisor.

Advisor: Dr. Fetene Mulugeta Signature: ________________________

Addis Ababa University

Addis Ababa Institute of Technology

Department of Electrical and Computer Engineering

Volumetric Segmentation of Brain Tumors Using the 3D U-Net Architecture

By: Mahlet Alemseged Bitew

APPROVED BY BOARD OF EXAMINERS

____________________________________ _________________________

Chairman, Department of Graduate Committee Signature

Dr. Fetene M.__________________ _________________________

Advisor Signature

_______________________________ __________________________

Internal Examiner Signature

_______________________________ ___________________________

External Examiner Signature

Dedication

To my mother, for “all that I am and all that I ever hope to be I owe to my mother”.

Acknowledgements

First and foremost I would like to thank GOD, the almighty, for the wisdom, perseverance, strength and patience He gave me to get through this research work. Next, I would like to express my sincere gratitude to my advisor Dr. Fetene M. for his guidance and all the reviews and comments throughout the project. Also, I forward great thanks to my friends and my family who were there for me while doing this thesis work. And last but not least, I would like to thank my parents for the price they paid to see me complete my studies even though they are not here with me now.

Abstract

Brain tumor segmentation is one of the most challenging types of medical image segmentations. To overcome the problems of automated brain tumor classification, a deep learning approach is proposed herein based on a 3D U-Net model. A 3D model is chosen to get the 3D context of the tumors which are irregular in shape and could occur anywhere in the brain. In the process of building this model, first, the data is visualized using different formats of visualization in order to understand the underlying patterns of the data well. Then the 3D model is developed herein which consists of 22 layers of convolution of which about half do downsampling and the rest perform an up-sampling of the feature maps. The skip connections provide more context to the up-sampling layers which change the output back to its original size. The segmentation model is trained and evaluated on the BRATS 2020 dataset.

There are three versions of the model that are run and observed herein. Of the three versions the second one, Model2, appears to perform the best. This is the model having data augmentation with batch size of one. The presented model (Model2) attained a dice similarity coefficient (DSC) of 0.90. The result obtained shows that the presented method does a good job of segmentation and compared to other state-of-the-art methods our technique can be considered competitive in the area of automatic brain tumor segmentation. In addition, the proposed model is simple when compared to other brain tumor segmentation models which are complex; this inherently helps it require less segmentation time than earlier models as it only takes 3.44 seconds to segment one patient's MRI scan. This contributes to attaining functional real-time applicable models which can be of real help to physicians and radiologists during the classification of brain tumor precisely.

Key words: U-Nets, brain tumor segmentation, dice similarity coefficient (DSC), BRATS, MICCAI

About This Document

This document is a formal report of the thesis on Volumetric Segmentation of Brain Tumors Using the 3D-UNet Architecture. The theoretical backgrounds and the physical implementations are described. The results obtained are analyzed and conclusions and inferences are made as appropriate.

I. Definitions, Acronyms, and Abbreviations

The following table defines the acronyms and abbreviations used in this document

Term	Definition
AEs	Auto-Encoders
AI, ML, DL	Artificial Intelligence, Machine Learning, Deep Learning
Bi-RNNs	Bidirectional Recurrent Neural Networks
BraTS	Brain Tumor Segmentation
C-GRU	Convolutional-Gated Recurrent Units
CNN	Convolutional Neural Network
CT	Computed Tomography
DSC	Dice similarity coefficient
EDA	Exploratory data analysis
FCN	Fully Convolutional Network
FLAIR	Fluid Attenuated Inversion Recovery
FCN	Fully Convolutional Network
GANs	Generative adversarial networks
GRU	Gated Recurrent Units
HGG	High-Grade Glioma
IOU	Intersection over union
LGG	Low-Grade Glioma
LSTM Networks	Long-Short Term Memory Networks
MICCAI	Medical Image Computing and Computer Assisted Interventions
MD-LSTM	Multi-Dimensional Long-Short Term Memory Networks
MRI	Magnetic Resonance Images/ing
M-CT	Micro-Computed Tomography
ReLU	Rectified Linear Activation Unit
RNN	Recurrent Neural Networks
ROI	Region of interest
SISS-ISLES dataset	Sub-acute Ischemic Stroke Lesion Segmentation—Ischemic Stroke Lesion Segmentation dataset

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Glioma is a type of brain tumor that affects the Central Nervous System [1]. It is a serious illness with a survival rate of less than 2 years. Patients are mainly monitored by Magnetic Resonance Images (MRIs). It is the preferred technique of imaging because it is non-ionizing, non-invasive, and with good spatial and temporal resolution [2]. The idea behind this imaging employed herein is that one can visualize the tumor without the need to break the skin of the patient or physically enter into the internal parts of the body. Also, by looking at the MR images of the tumor and measuring its size the progress of the tumor and its response to medication can be analyzed properly. Here, semantic segmentation plays a huge role in identifying and labeling the tumor correctly; hence it is an important diagnostic task.

This task of delineation of the brain tumors within the MRI images of the brain is typically done manually by an expert radiologist [3]. However, it is a very time-consuming task; especially if there is a large database of images. There is also some variability amongst some radiologists with certain tasks which results in variability in segmented images. If data analysis is required for thousands of MR images of patients, manual delineation becomes virtually impossible to do. In order to augment the effort of the radiologists, deep learning programs that can effectively segment gliomas have become very valuable.

Nowadays, the utilization of deep learning is quickly escalating in almost every area of specialty, especially in the domains of medical image processing, medical image analysis, and bioinformatics [4]. On that account, deep learning has radically altered and improved the procedure of recognition, diagnosis, and prediction effectively in various areas of healthcare such as pathology, lung cancer, cardiac, abdomen, and retina. In this research, deep learning algorithms will be employed extensively to segment brain tumors (specifically high grade and low grade gliomas) and as a result, assist medical professionals in the timely detection and treatment of this deadly disease.

1.1 Background

There are several types of brain tumors and among them, gliomas are the most common. They also tend to spread too quickly leading to a quite short life expectancy when they occur in their highest grade. Recent treatments are chemotherapy, radiotherapy, surgery, or any combination of them [5]. In the attempt of improving the quality of life of oncological patients, treatment planning plays a vital role. MRI is especially applicable for the evaluation of gliomas in clinical practice because it is possible to acquire a sequence of MRI volumes providing complementary information [6].

As far as MR imaging is concerned, typically, multiple MR volumetric sequences are acquired. The acquired image volumes are essentially 3D arrays. Each image volume is composed of the in plane along with the in-depth image across the anatomy of the brain in the form of slices. Each image volume is commonly referred to as a sequence. And each sequence refers to an explicit method of exiting the magnetic spins inside the human body so that each of them results in a different kind of grayscale contrast in the images.

Normally, for the imaging of gliomas, there will be 4 types of volumetric sequences namely T1C, T2, FLAIR, and T2C [7]–[9]. Gliomas are made of four constituents, which is why there is a need for not just one but four sequences. This is because certain components of the gliomas are seen much more clearly in one type of sequence while other components may be visible in separate sequences.

Constituents of gliomas:

- **Edema:** A collection of fluid or water that appear to be fingerlike in shape. The best way to see them clearly is in the FLAIR and T2 weighted volumetric sequence.
- **Necrosis:** A collection of cells that are dead. Best visualized using T1 post-contrast sequence.
- **Enhancing Tumor:** Clearly display the breakdown of the blood-brain barrier which is best observed employing the T1C post-contrast volumetric sequence.
- **Non-Enhancing:** Areas in the tumor region that are not edema, necrosis, or enhancing tumor.

1.2 Dataset

The dataset intended to be used for this research is from Kaggle database [10]. This dataset is one that is used for the BraTS (Brain Tumor Segmentation) challenge held yearly by MICCAI, which is one of the major medical imaging conferences. To the best of our knowledge, this dataset that is used for the BraTS challenge is currently the largest and the most comprehensive brain-tumor dataset utilized for validating existent and upcoming algorithms for the purpose of detecting and segmenting brain tumors. It is public and multi-centric, meaning the MR images are gathered from different centers using different scanners. This is crucial as shadings of image sequences vary from hospital to hospital and our networks can successfully generalize to some new data from a different hospital if images from several centers are employed during training.

Moreover, the images in the dataset consist of both high grade gliomas, which are much more serious, and low grade gliomas, which can grow to become high grade if left untreated. Therefore, the dataset used is heterogeneous in this sense. The tumors in these two types of gliomas do look different so it is important to include both in the training data. This is important because if for instance only high grade gliomas are used during training, the resulting network will probably not work for low grade gliomas.

The volumetric sequences for each patient within the BraTS dataset that are used in the training of the model developed in this thesis project consist of:

- Fluid Attenuated Inversion Recovery (FLAIR)
- T1 weighted sequences
- T2 weighted sequence
- T1 post-contrast sequence and
- Pixel-level segmentation mask

Each technique used gives rise to different grayscale contrast that helps the radiologist identify the pathology in the tumor. The MR sequences are all skull striped, registered, and re-sampled to have an isometric resolution (1mm3). The dimension of each volume is of size 240x240x155

(Sagittal, Coronal, Axial) and there are four such volumes. In practice, the images can be sliced along any axis as they are 3D by nature. Also, the ground truths are provided for these datasets.

1.3 Statement of the Problem

Brain tumor detection aims to identify abnormal cells or lesion tissues and report the predicted class of each tissue. Some methods developed for this purpose return the bounding box as the detection result and the label as the classified result [11]–[13]. It is worth mentioning that other research works using the words "brain tumor detection" for their researches only give the bounding box of the tumor tissue as their detected output. While yet some other research methods return the center of the tumor core as the point of interest without performing any additional reasoning and segmentation.

Despite significant progress that has been made in the area of brain tumor segmentation, state-of-the-art deep learning based methods still experience unsatisfactory outcomes with several challenges that are yet to be solved. These challenges associated with brain tumor segmentation can be summarized as follows.

First, because Glioma is a result of mutation of the gluey cells [14] in the brain, which is a type of supportive cell surrounding nerve cells with a wide spatial distribution, they can occur anywhere inside the brain. This causes location uncertainty. Second, due to these tumors having irregularity in shape, there also exists a large morphological uncertainty that the model needs to track accurately. Third, the available volumes of images are of low quality and mostly in diffusion and low contrast [15]. For this reason, the boundary between biological tissues appears to be blurred and hard to detect in most cases. This limits the automated algorithms' ability to extract adequate information for further processing. Moreover, current manual annotation is highly dependent on personal experience, which can be the reason for some annotation bias introduced at the time of dataset labeling. Due to these annotation biases the detection algorithm may be confused during the learning and processing stages. Hence, considering these mentioned points, there is no doubt that there is a great necessity to address these problems worldwide.

1.4 Objectives of the Study

1.4.1 General Objective

The goal of brain tumor segmentation is to come up with an accurate delineation of brain tumor areas with correctly positioned masks. In light of this, this study hopes to achieve increased efficiency with a simple design requiring low processor power because the training is run online. Moreover, increased accuracy and reduced error rates are the main objectives of this project.

1.4.2 Specific Objective

The specific objectives of this project are the following set of major tasks:

1. Literature review of previous research undertaken in brain tumor segmentation area.
2. Selecting a dataset and understanding it thoroughly - EDA.
3. Data preprocessing – perform a series of operations in order to prepare data for training.
4. Design an appropriate model for the task at hand.
5. Develop a 3D data generator that feeds the model on the fly.
6. Develop methods to stop and resume running code and also methods to save the trained model at the end of each epoch.
7. Training the model.
8. Test the model with previously unseen (by the model) data.
9. Measuring the performance of the developed system and plotting several metrics that indicate system performance along with confusion matrix.

1.5 Scope and Limitation of the Study

Brain tumor segmentation is one of the most daunting types of segmentation. The main focus of this thesis is the design and development of a 3D segmentation model for a specific brain tumor named glioma. The dataset used for the training of the model is the BraTS 2020 dataset on Kaggle. For each patient within the dataset, one finds four sequences of brain MRI volumes along with the segmented mask of the brain scan. The developed model has the architecture type of 3D U-Net and is trained herein which achieves quite a high performance that is comparable to other SOTA techniques. Moreover, a 3D data generator class, a logger class, a model checkpoint class, and several other helper functions are constructed for the system to achieve this level of performance.

This thesis is conducted for the purpose of brain tumor segmentation, specifically gliomas; it does not include the segmentation of other types of brain tumors for example schwannoma. Gliomas can appear in any part of the brain and sometimes even the spinal cord, however, only brain MRI scans are employed herein, and therefore only brain tumors within the brain are dealt with. After voxel-level segmentation is performed, detecting the survival rate of patients, recommending medicine to patients, or even determining the severity of the disease are all beyond the scope of this thesis.

The major limitation in conducting this thesis was the unavailability of a Graphical Processing Unit (GPU) which is powerful to perform the model training locally. GPU is indeed the most important resource for anyone working on deep learning algorithms, especially for image processing. Also, the lack of continuous internet and power supply during training were hurdles in training the models.

1.6 Significance of the Study

Brain tumors are deadly diseases and require an expert to be detected successfully which is a costly matter. Moreover, these segmentations of MRI brain scans require a significant amount of time to segment pixel by pixel accurately. Also, the segmentation results appear to vary from expert to expert to some extent. Be that as it may, currently, this task is performed manually by hand with expert radiologists and physicians. However, if automatic segmentation is performed with software this task would be greatly simplified. It would cost much less time and effort. Also, accurate delineations can be performed, not to mention it would take a fraction of the money required by manual segmentation.

All in all, the benefits that can be had as a result of this thesis are explained as follows:

- This thesis will allow radiologists and physicians to appreciate the importance of artificial intelligence and computer vision in their respective fields.
- The thesis will help achieve high accuracy and the performed segmentation will be less variable across different centers and experts than the current ones.

- The cost of segmentation will reduce immensely as hiring experts to perform it is too costly.
- As a result of this thesis physicians will be assisted to identify the status and progress of the brain tumor faster which in turn helps patients get timely and high-quality treatment.
- This thesis helps experts to perform analysis of brain scans of hundreds or even thousands of patients and enables them to identify common traits of the disease. Also, other hidden matters can become visible once the segmentations of many brain scans are analyzed. This helps experts get one step closer to treating the disease.
- And finally, this thesis will serve as reference material for researchers interested in the area of brain tumor segmentation or even in computer vision in general.

1.7 Organization of the Thesis

This thesis paper consists of five chapters. Within the current chapter, Chapter 1, the background to the study, the specific and general objectives of the study, the problem statement, an overview of the dataset used, and the limitation and scope of the research work are presented.

Following this, Chapter 2 details available previous brain tumor segmentation techniques that are reviewed by subdividing these techniques based on the underlying structure of the models they employ.

In Chapter 3, a thorough discussion of machine learning, deep learning, and artificial intelligence is performed. Then the methods used to implement the brain detection model of this specific project and support functions that make it fully functional are discussed in detail. This includes the methods utilized for data preprocessing and data generation up until the plotting methods of the evaluation metrics.

Chapter 4 delivers results obtained after training the fully constructed model are discussed (Chapter 4). Moreover, results when training models with batch sizes of 1 and 2 are explained and compared and contrasted.

Finally, the paper is concluded in Chapter 5, where the pros and cons of the developed model are highlighted, and promising future works are indicated.

Chapter 2
Literature Review

Glioma is a common type of tumor that mainly arises in the brain; however, it can also occur in the spinal cord in some cases. Of all the brain tumor occurrences in humans, about 33% of them are found to be gliomas [16]. These tumors originate from the glial cells that encompass and support neurons. They could be high-grade or low-grade depending on how extensively the brain is attacked by the disease. It is a deadly disease and needs prompt and precise treatment. Segmentation of this tumor is a necessary step towards taking the required measures to help the patients. Automating this process of segmentation and performing high-quality segmentations is being highly researched nowadays in the computer vision area of study. And deep learning algorithms have become the go-to methods in the area of medical segmentations as a whole [15].

These deep learning researches of segmentation specific to the gliomas can be divided into four major groups depending on the underlying architectural structure they employ; these previous researches are discussed well in this chapter. Also, a quick review is provided on the disease itself and machine learning, Artificial Intelligence (AI), and deep learning.

2.1 The Human Brain

In the universe, there is no more complex structure than the human brain. It is a spongy mass of fat and protein that weighs just about three pounds on average; this spongy mass consists of many billions of two overarching types of cells—called glia and neurons [17]. Neurons have significant branch-like projections named axons and dendrites. These projections help the neuron cells collect and transmit electrochemical signals. On the other hand, there are several different types of glial cells that supply physical protection to neurons and help maintain their health and of the brain as a whole. The combination of this complex network of cells brings about every aspect of humanity as we know it. It simply would have been impossible to breathe, play, love, or remember without the brain.

The brain which performs all the vital work stated above and more is an extremely delicate organ that needs maximal protection. This protection is provided by the skull, a hard bone consisting of twenty-two bones, and three tough membranes known as meninges [18]. The space between the three membranes contains fluid that cushions the brain and avoids direct contact with the skull which can cause damage to this sensitive organ.

Certainly, problems arise when a machine as finely calibrated and complex as the brain gets harmed or malfunctions. And when something as dangerous as brain tumors arise, the brain faces immense difficulties especially if they are cancerous. There are two types of brain tumors based on their ability to multiply and spread [19]. The first ones are malignant or cancerous which grow and multiply and are far more vicious. The speed at which they multiply and grow varies from some being slow and to others fast. Only about a third of brain tumors are malignant. The second types are called benign (non-cancerous); they seem to be contained within a given space and do not multiply. However, both types of tumors can hinder the functions of the brain if they are big enough to press on neighboring nerves, blood vessels, or tissues.

2.2 Brain Tumor and Gliomas

Tumors of the brain, also known as intracranial tumors, are basically a mass of tissue growing uncontrollably [20]. The mechanisms that control healthy cell growth within the brain do not seem to put this turbulent mass growth in check. Currently, there are more than 150 distinct types of brain tumors [21]. However, there are two significant kinds of brain tumors namely primary and metastatic. Tumors originating from the brain tissues or in their proximity are called primary tumors. These types of tumors are further categorized as glial and non-glial. As their name implies glial tumors are made of glial cells whereas the non-glial types grow on or in the structures of the brain, comprising nerves, blood vessels, and glands. Metastatic brain tumors, on the other hand, are resulted from cancer cells spreading to the brain from other organs.

The causes of brain tumors are still uncertain to scientists [22]. Some of the known reasons include genetic mutations that cause uncontrollable mass growth in the brain. Moreover, exposures to large amounts of X-rays or earlier cancer treatments are recognized to be the only known environmental factors that bring about brain tumors. Furthermore, several brain tumors occur due to heredity conditions that exist among family members. These genetic factors are passed from generation to generation causing cancer of the brain in the family.

Amongst brain tumors, gliomas are the most prevalent primary central nervous system neoplasm [23]. They are complex and heterogeneous and are therefore difficult to treat. Gliomas make up 78 percent of malignant brain tumors making them the most common malignant brain tumors among adults. They emanate from the supporting glial cells that assist the nerves of the brain. Gliomas are mostly but not on all occasions malignant. Occasionally, they do not have the tendency to spread or metastasize and are benign. In most cases however they are cancerous; hence they grow, multiply and spread.

These fast-multiplying cancerous masses are termed "low-grade" and "high-grade" by oncologists depending on the speed with which they spread [24]. Low-Grade Gliomas (LGG) have relatively very slow growth, and yet it is still important to note that they are malignant and could potentially grow to a high-grade level if left untreated. Nevertheless, treatments for LGG typically have a favorable prognosis and results observed as a result of treatment are highly fruitful. Surgeons can often remove the entire tumor without the need for follow-up treatment with mostly positive outcomes. Contrarily, High-Grade Gliomas (HGGs) have a very high speed of growth and need multiple forms of treatment to alleviate the pain and accompanying neurological symptoms. The recommended treatment options are chemotherapy, radiation therapy, and targeted therapy among others. These improve the patient's quality of life and survival rate [20]. The aim here in this project is to assist professionals in the diagnosis of both LGG and HGG by making use of deep learning techniques. In the following section, deep learning and AI are explained well before going deeper and explaining deep learning techniques of brain tumor segmentation as implemented by previous researchers.

2.3 AI, Machine Learning and Deep Learning

The main aim of artificial intelligence (AI) is to build machines that perform and act like humans [25]. Artificial intelligence was first presented in 1950 [26]. Originally, the aspiration of those AI pioneers was to assemble human-like machines powered by emerging computers that possess similar traits as the human intelligence. This concept is what is referred to as "General AI"—amazing machines with all our reasoning capability including our senses (or even more) along with astounding skills of thinking. Nevertheless, there is a reason these amazing general AI machines still remain in movies and novels of science fiction; they cannot be pulled off, at least not yet.

As of now, what can be practically done falls into the category of “Narrow AI” [26]. These are technologies that are capable of accomplishing specific enterprises as well as or even better than human experts. Typical cases of narrow AI are image classification on a service like Pinterest or face recognition on Facebook.

However, within the medical field, an extensive acceptance and application of AI was hindered due to several limitation factors present in earlier models. At the beginning of the 2000s, the dawn of deep learning helped overcome these restraints [27]. Currently, AI systems are self-learning and have the ability to make analysis of complex algorithms. Indeed, a new era of medicine has come to pass. Clinical practice in medicine can now be assisted immensely by AI. Significant contributions of AI in medicine include risk assessment models, improvement in diagnostic accuracy, and workflow efficiency. Nevertheless, note that all of the examples mentioned here are still examples of “narrow” AI in practice and not General AI.

Even though “narrow” these technologies exhibit some aspects of human intelligence. But how do they perform in some ways like humans? Where does their intelligence emanate from? This can be explained by going deeper into the circle of AI, which is machine learning, as shown in the figure given below (Figure 2.1).

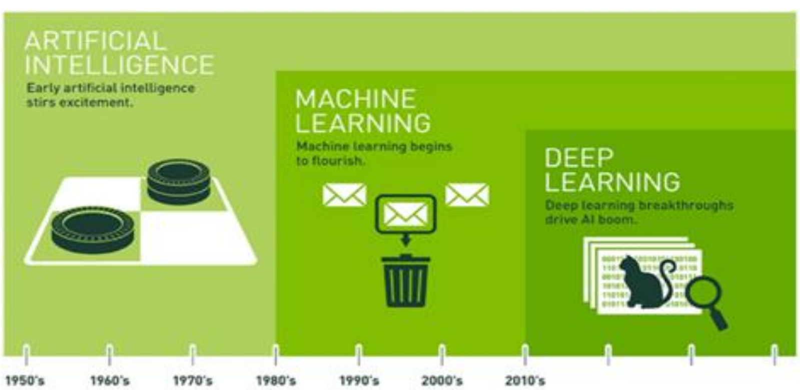


Figure 2.1: The advents of AI, ML, and DL in history: Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions [28]

2.3.1 Machine Learning

Machine learning (ML) is a procedure by which artificial intelligence can be achieved. "Machine Learning is the science (and art) of programming computers so they can learn from data" [28]. In rudimentary terms machine learning is a means of parsing data, learning from it, and being able to predict or determine something in the real world. Therefore, instead of coding software by hand and indicating to the machines how to accomplish each task line by line, large amounts of data and algorithms are used to train them. This enables machines to learn to perform the same task with no specific set of instructions.

Simply stated, machine learning enables machines to "learn". Traditionally, in order for computers to perform any task there is a need to provide a strict set of directions to follow. In the case of machine learning, however, instead of providing the computers a given set of instructions on how to perform a task they are given instructions on how to learn to do the same job.

ML is a direct result of the thoughts of early AI scientists. Ever since the idea of ML was conceived several algorithms came to life to make it practical. These algorithms include clustering, decision tree learning, reinforcement learning, inductive logic programming, and Bayesian networks among others [25]. As it is very well known, the ultimate goal of General AI is not yet achieved by these algorithms. For early machine learning methods, it was not even possible to achieve Narrow AI.

In the beginning, machine learning was too fragile and too susceptible to error [26]. This was the major reason why programmers in the computer vision and image detection area could not develop functional applications then. Nevertheless, algorithms like deep learning appeared as time went on and shook up everything in the area.

2.3.2 Deep Learning

With machine learning the basic change that was brought about was that computers did not need explicit programs. However, they still think and act like machines. For instance, when it comes to some complex tasks like gathering data from an image or streaming video they fail to reach the performance level of human beings.

Modeled after the human brain itself deep learning models are meant to overcome these kinds of hurdles [29]. They do so by introducing an extremely sophisticated technique to machine learning. Complex deep neural networks are made of several layers that are highly connected. Each layer in these multi-layered neural networks is made of nodes (like neurons in the human brain). Data passes from layer to layer through these nodes. As the number of nodes increases in a network, the net-like connection between layers increases and becomes increasingly complex. These computational models that are equipped with Deep Learning (DL) are made of multi-level processing layers representing data with multiple levels of abstraction. As a result, an increasingly abstract non-linear transformation of data is attained.

Owing to deep learning, AI has now seen the dawn of light and has a blazing future [27]. This is a consequence of neural networks performing quite well in recent years - partly due to the insane amount of data available lately and partly because of Moore's law. Although an enormous amount of data is needed to first build such models, they are able to generate results immediately and a relatively small amount of human monitoring and intervention is needed once the models are well-trained.

Recently, the significance of deep learning has been escalating in every field [30]. Especially in the medical fields of bioinformatics, image processing, and image analysis deep learning has been extensively used. Ergo, it has drastically improved and advanced the performance of effective diagnosis, recognition, and even prediction in several areas of medicine. For instance, lung cancer, cardiac, abdomen, and retina are a few areas in healthcare that have been hugely assisted by the occurrence of deep learning. In this research project, deep learning algorithms will be employed extensively to detect brain tumors (gliomas) and assist medical professionals in the timely detection and treatment of this deadly disease.

2.3.3 Deep Learning with TensorFlow

TensorFlow is a Deep Learning Framework that is quite popular amongst developers in the AI industry [31]. This framework was made by Google in order to ease up the creation of Deep Learning models. Deep learning is a subset of machine learning which employs neural networks that are multi-layered. Problems considered to be complex are now empowered with machine learning and are untangled with great fidelity. Several applications that have to deal with streaming videos, audio, volumetric images, or text can be implemented by making use of Deep Learning employing the TensorFlow framework. Thanks to TensorFlow, now many problems solved with machine learning are made much easier to create.

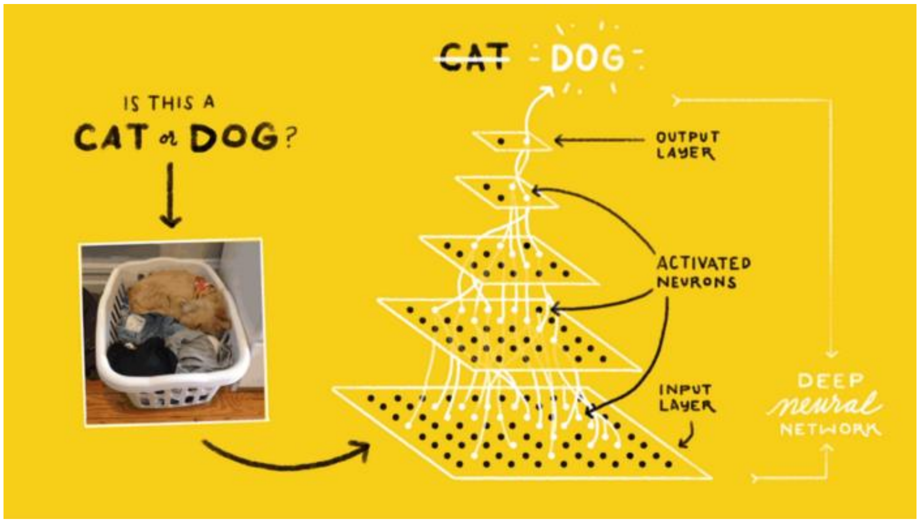


Figure 2.2: Deep Learning concept of TensorFlow [28]

The main cause of the high rate of acceptance of TensorFlow by developers is that it makes the building and deployment of applications convenient. Furthermore, devices with limited processing power are able to run applications made by it. For instance, computers with very limited processing power and even smartphones can run the library.

2.4 Brain Tumor Segmentation with Deep Learning

Deep learning is regarded as a leading subclass of machine learning by a myriad of researchers. State-of-the-art performance is attained in many areas like object detection [5] and natural language processing [6] due to the advent of deep learning. Lately, semantic segmentation also has performed favorably with the use of deep learning, and brain tumor segmentation is no exception to this. In Figure 2.3 the number of papers employing deep learning techniques in the BraTS competition held by MICCAI is presented. It can be observed that there is a shift towards deep learning segmentations as a result of the high system performance achieved by these methods starting from the year 2012.

Segmentation is a challenging task as every pixel within the image needs to be given a label or a class; when dealing with 3D MRIs things get even more complex [32]. For this reason, brain tumor segmentation is one of the most testing tasks among medical image segmentations. The aim here is to accurately delineate the brain tumor within the 3D MRIs making use of optimally placed masks. Several techniques based on deep learning have been implemented so far and have got good results [15]. Nowadays, many widely-known deep learning algorithms exist including convolutional neural networks and recurrent neural networks. In this section, deep learning based brain tumor segmentation techniques implemented by previous researchers are classified into four major categories and are discussed in detail. This classification is made based on the underlying architecture or structure of the algorithms the papers employed.

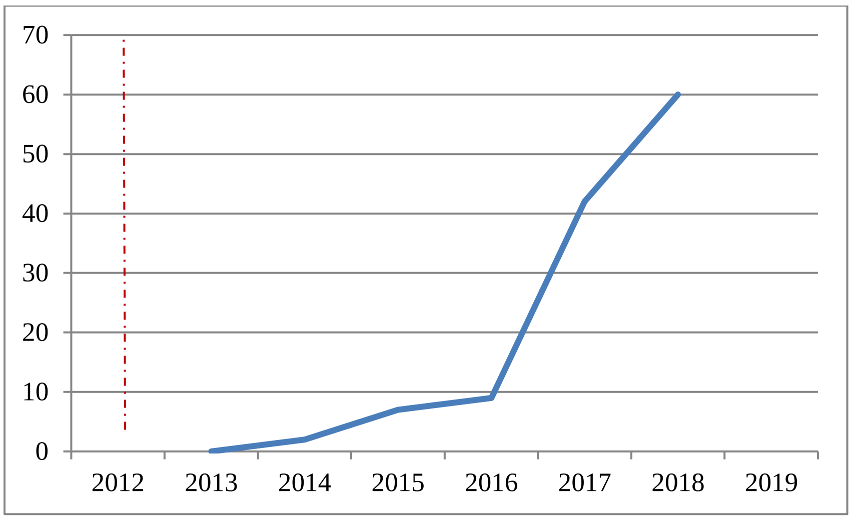


Figure 2.3: Number of papers employing deep learning technique in BraTS [15].

Every year a brain tumor segmentation challenge (BraTS) is held by MICCAI. The number of papers employing deep learning techniques is indicated above (Figure 2.3). It can be noticed that there is a shift to deep learning segmentations as a result of the high system performance achieved by these methods since 2012 (Red dashed line)[15].

2.4.1 Convolutional Neural Network Based

Convolutional Neural Networks mostly known as CNNs were first employed in hand-written digit recognition systems and image classification [33], [34] tasks. Since then, computer vision in general has benefited a lot from these types of networks. CNNs are excellent in feature learning making use of trainable filter weights and producing a number of feature maps. They conveniently learn features from images and even videos. Typically, a CNN is made of convolutional layers to learn features, activation layers to incorporate the real world's non-linearity, pooling to reduce dimensionality, and finally, a fully connected layer to perform the actual classification task. However, in many applications using deep neural networks CNNs have been extensively implemented as the backbone network excluding the final classification layer. Furthermore, for research and practical uses in the brain tumor segmentation area, these networks have been found to be highly effective without the need to use any non-convolutional layer at the end.

In this paper, the CNN based methods employed for brain tumor segmentation purpose are divided into two categories: (a) single-path CNN and multi-path CNN and (b) Fully Convolutional Networks (FCNs).

a. Single Path and Multiple Path CNN

CNNs that are of the type single-path comprise a single data processing path [35]. First, the network starts out by sampling the input image with convolutional layers. Following this are the pooling and non-linear rectifier layers. One factor that makes single-path networks preferable is their computational efficiency. For this reason, a great number of researchers such as [35]–[38] have put it to use for brain tumor segmentation.

Multi-path convolutional neural networks, on the other hand, have the ability to extract separate features from separate processing pathways having distinct scale values [15]. These different features are then merged or concatenated together and passed forward for additional processing. Typically, the path having a larger size kernel picks up features from a bigger reception field named global features. Conversely, local features coming from smaller reception fields are learned by kernels with smaller size. Global spatial information like the global locations of the target are produced by global features. Whereas, more detailed facts are supplied by local features; texture, size, and boundary are a few examples of the details indicated by local features. A rather early research work presented in [39] employed a multi-path CNN with 3 pathways for the purpose of pixel-wise delineation of brain MRIs. Pathways collecting local features comprise kernels of size 5×5 and 7×7. However, a 9×9 kernel is employed by the global pathway. Simultaneous feature extraction is performed by both the local and global pathways each having three layers of convolution.

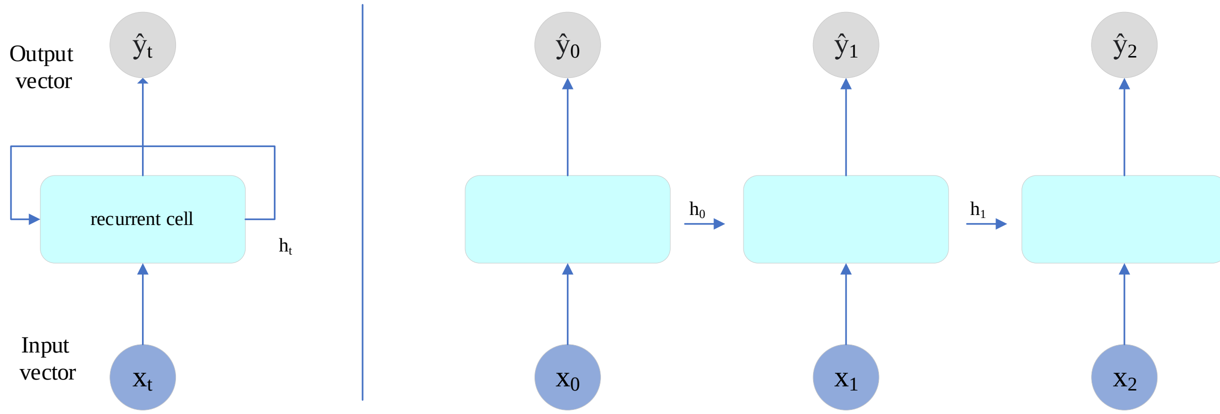
b. Fully Convolutional Based

CNNs have been employed in the image classification area for a while now. In the earlier years of using them, their final layer was a fully connected layer. This final layer produces a single value which can be interpreted as the predicted label with the highest probability. Long et al. [40] introduced fully convolutional networks which use deconvolutional layer/s to replace the final fully connected layer and predict segmentation masks directly. In the case of fully convolutional networks, the replacement of the final layer by the deconvolutional layer/s is for the purpose of transforming the down-sampled feature map back to the size of the original image. For this reason, FCNs can be trained conveniently in an image-to-segmentation map fashion resulting in an advantage in efficiency over CNN patch classifiers. A well-recognized variant of FCNs is known as U-Net [41]. It is a simple yet powerful architecture that is used for the model developed in this project and other previous models of earlier papers [42] of brain tumor segmentation.

2.4.2 Recurrent Neural Network Based

Recurrent Neural Networks (RNNs) were first assembled to handle sequential process problems where the current output of a system depends on the current input as well as representations of previous inputs of prior time [43]. This means that recurrent neural networks are able to learn the representation of the time sequence inputs. It is actually the first algorithm that remembers its input because it has an internal memory. And this makes it perfectly suited for problems of machine learning that involve sequential data.

As mentioned above, RNNs have a special memory function enabling them to remember and reuse previously acquired information. At each time step as a sequence is processed this memory function [or cell state] represented by h_t is updated. In Figure 2.4 it can be seen that the output of an RNN network is dependent on the current input and previous cell states. The depiction on the right of the image shows the individual time steps unrolled across time but this recurrence can also be shown using the cycle/loop shown on the left of the image. When it comes to the 3D data segmentation of brain tumors using RNNs, mostly one dimension of the 3D MRI is considered as the time dimension while the remaining two dimensions of the volumes are provided as the sequential inputs of the RNN network.



$$\underbrace{\hat{y}}_{\text{Output}} = f(\underbrace{x_t}_{\text{Input}}, \underbrace{h_{t-1}}_{\text{Past memory}})$$

Figure 2.4: Neurons with Recurrence [44]

Long Short-Term Memory (LSTM) networks are a mutant of RNNs and are currently the most common and the most broadly implemented class of RNNs [45]. Inside the LSTM cell, the historical information of LSTMs can be stored and processed over a long sequence. This is made possible by the gated cells that they employ. These gated cells of LSTM networks are more complex recurrent units to control what information is passed through and are quite helpful for tackling the vanishing gradients problem. Works of Graves et al. [46] and Byeon et al. [47] exhibited pioneer works for applying Multi-Dimensional LSTMs applied to image classification and segmentation tasks. Multi-Dimensional Long Short-Term Memory (MD-LSTM) network takes each pixel's entire spatio-temporal context for both images and videos into account in an elegant and recursive way which makes it different from CNNs. Consequently, this feature also makes the MD-LSTM networks quite useful for brain tumor segmentation as the dataset employed are 3D volumes of brain MRI scans.

One example of brain tumor segmentation based on the long short-term memory (LSTM) model using magnetic resonance images (MRI) was presented in [48]. First, N4ITK and Gaussian filters with a kernel size of 5 x 5 are employed to boost the quality of multi-sequence MRI. Following this, a deep LSTM model possessing four layers is used for classification. Within each layer, hidden units (HU) that are optimal to the system are chosen like 200 HU, 225 HU, 200 HU, and

225 HU, respectively. Before choosing these hidden layers, however, extensive experiments were performed to attain better results. Validation of the results obtained by the model is performed on the SISS-ISLES 2015 dataset and a number of BRATS datasets (BRATS 2021-15, 2018). The results reveal that the proposed method obtains good results in segmenting brain tumors.

Another type of RNN is known as Gated Recurrent Units (GRU) [49]. Instead of applying a forget gate and the input gate for combining the hidden and cell states like the LSTM networks, it employs an update gate. GRUs require less memory which makes them more capable of processing large volumetric data, and this, in turn, encourages deeper and larger networks for the same volume size. [50] performed brain tumor segmentation using Multi-Dimensional Gated Recurrent Units. The cell block employed here is convolutional GRU (C-GRU). Image prepossessing of tumor dataset like high-pass filtering and intensity normalization are also employed in this research. The setup which combines these techniques in [50] is found to outperform the state of the art in tumor segmentation.

Recurrent neural networks own significant power in handling time-sequence processing tasks. In order to perform segmentation Convolutional Neural Networks (CNNs) can be moved across 2D images or 3D videos to segment them. However, having fixed input size and typically being able to perceive only small local contexts of the pixels to be classified as foreground or background their performance with regards to understanding context is somewhat limited [15]. On the contrary, Multi-Dimensional Recurrent Neural Networks (MD-RNNs) can get the entire spatio-temporal context of every pixel in an image within a few sweeps through the pixels. This is the case, especially for Long Short-Term Memory (LSTM) networks. LSTM and GRU allow their networks to be equipped with the ability to filter historical information making use of the memory and forget gates. When dealing with 3D datasets current RNN models take the third dimension as the time sequence axis and this extends the networks' ability of using spatial information. In the past, there have been several RNN type state-of-the-art models developed which contributed to the betterment of brain tumor segmentation [47], [50], [51].

2.4.3 Deep Generative Based

Several challenges are still obstructing the performance of deep neural networks. First, deep neural networks, like many supervised methods, assume that the distribution among testing data and training data is the same which is also the case for traditional machine learning techniques [52]. For datasets in the real world, however, there exists a variation between testing and training data distributions. This, in turn, may cause prediction bias or over-fitting of models. Secondly, the labeling of datasets consumes a high amount of time and labor; this is the case, especially for medical applications requiring pixel-level segmentation of datasets. In addition, the majority of deep learning models put their focus on pixel-level segmentation while disregarding the connection or relation between adjoining pixels. This may result in a high level of accuracy in pixel-level classification although the overall segmentation result remains inconsistent because of the shape and size variations of the target images.

Deep generative models have shown significant improvements in data simulation and conditional density estimation [53]. In recent times, researchers have used generative adversarial networks (GANs) and auto-encoders (AEs) for brain tumor segmentation. Normally, generative adversarial networks are built from a generator and a discriminator. These two are trained to play a game of min-max together. For applications performing brain tumor segmentation, they are trained so that the generator generates the segmentation map of the tumor and the discriminator decides if the segmentation mask presented to it is generated or is selected from the ground truth in the dataset [54]. Meanwhile, auto-encoders work on the reconstruction of healthy brain images from the available training images. Then the contrast between the reconstructed healthy brain image and the original training data is the segmented tumor [55].

2.4.4 Model Ensembling

Different networks perform very diversely with different strengths and weaknesses. The models are influenced by the main architectures and the training settings. Recent research shows that model ensembling can average the variances within the sub-models and configuration-specific behaviors [15]. This can lead to an unbiased generic method with robust performance.

One main drawback of deep neural networks is the model's performance and behavior can be influenced by architectural choices and training data. Therefore, most of the proposed deep neural networks only perform well on specific datasets and pre-settled tasks. This refers to the limited generalization capability of deep neural networks. For example, feed-forward networks with a big kernel size can capture well spatial information while a small size kernel allows us to learn boundary features. Models with pixel-oriented post-processing can generate continuous tissue segmentation masks while other models may achieve better accuracy on pixel-level classification. In order to build more robust and more generalized segmentation methods, several models' outputs can be aggregated together with a high variance between each other, known as ensemble models.

There are several studies focusing on ensemble models for tumor segmentation [56]–[59]. [56] ensembled 26 neural networks for tumor segmentation and survival prediction. They use brain parcellation to produce location prior information for tumor segmentation. [57] reported a 3D CNN to perform pre-segmentation and refined the pre-segmentation mask using the ensembles of 4 different CNN architectures, where all the sub-nets share the same meta-parameters with different architectures and weights. The aim of this process is to capture the bias of the networks whilst keeping the same input information.

Instead of using multiple models as different information extraction pipelines, Chen et al. [58] used one set of DeconvNets [60] to generate the primary segmentation probability map, and another multi-scale Convolutional Label Evaluation Net is used to evaluate previously generated segmentation maps. False positives can be reduced using both the probability map and the original input image.

Ensemble models are effective to support brain tumor segmentation. In spite of their contribution, there are several drawbacks to using ensemble models. First, their computational costs are very high. In using ensemble models, to avoid over-fitting, high variances between sub-models should be introduced by configuring and training sub-models in different ways. This leads to heavy loaded computational costs in the training stage before the models can be adopted. Second, the voting scheme of concatenating sub-models lacks meaningful interpretations. Ensemble methods usually use an averaging scheme, or top-N scheme to calculate the confidence score of each voxel in each class. This may be further improved by optimizing the weight of each sub-model's output by minimizing sub-model-level loss functions.

2.5 Other Related Works

The convolutional neural network (CNN) includes several convolutional layers, which are concatenated with pooling layers followed by a fully connected (FC) layer to process the high-level input data. After its inception, the CNN technique was applied to several segmentation tasks of brain lesions and outperformed previous conventional approaches [61]. Indeed, challenging biomedical 2D images can be segmented with an accuracy close to human performance by using CNNs [41], [62], [63]. Because of this success, a number of attempts have been made to utilize 3D CNNs on volumetric biomedical datasets. Milletari et al. [64] present a CNN combined with a Hough voting approach for 3D segmentation. However, their implementation is not end-to-end and can only be applied to compact blob-like structures. The approach of Kleesiek et al. [65] is one of few end-to-end 3D CNN approaches for 3D segmentation. However, their architecture is not deep and has only a single max-pooling layer after the first convolutions; for this reason, it is unable to a make multi-scale analysis of structures.

Moreover, according to [66], prior to the advent of the U-Net structure, it was generally agreed upon that there was a need for thousands of annotated image data in order to successfully train deep networks. In [66] however, a network strategy that relies on the strong use of data augmentation was presented. This helped to use annotated samples more efficiently. At the time, the network outperformed a method that was previously best performing, a sliding-window convolutional network, on the ISBI challenge for the segmentation of neuronal structures in electron microscopic stacks. This network can be trained end-to-end without the need to have thousands of training images and with good performance. In addition, this original U-Net structure took less than a second to segment an image of 512 x 512.

For this research work, in an attempt to segment brain tumors, the 3D variant of this famous U-Net structure is utilized. In order to achieve high effectiveness in segmentation, extensive data augmentation is employed. The network is built without the use of external trained models.

2.6 Summary

As can be observed from the previous subsections, AI specifically deep learning has a lot to contribute to the field of medicine including the task of brain tumor segmentation. Brain tumor segmentation is quite difficult because the location of the tumor could be anywhere in the brain and the tumor region is also irregular in shape. Deep neural networks fit these problems well as they can model complex parameters with their deep layers.

There have been many attempts made to successfully segment brain tumors and still, several attempts continue to be made to meet the required very high precision in the medical field. In this chapter, all the methods are subdivided into four major types of structures, namely: CNN based, RNN based, DGM based and ensemble method, and discussed. All have their own advantages and disadvantages and they are described in the previous sub-sections; however, nowadays, the CNN methods are outperforming the others in the task of brain tumor segmentation. The U-Net structure especially is winning accolades in several medical segmentation tasks. This architectural structure is employed in this thesis in an attempt to achieve accurate segmentation of brain tumors.

Chapter 3
Methodology

Nowadays, with the advent of AI technology using ML automatic segmentation has become quite popular and highly effective [67]. These Segmentation solutions learn from examples of good segmentation how to produce similarly high-quality results in new cases. This strategy has already been applied to common anatomies of interest in the orthopedic field like the knee and hip [68], [69]. However, in the case of brain tumors, implementation of segmentation using DL AI techniques is still a work in progress [70].

In an attempt to have a functional brain tumor segmentation model in real-life hospitals, a fully automated method is employed herein with the U-Net architecture [41] as the underlying structure. In this chapter, this 3D U-Net model is explained thoroughly. Also, the main frameworks and libraries used in the making of this work are described well in the next subsection. Moreover, other necessary supplement classes and methods implemented in this project are discussed in this chapter.

3.1 Architecture of Proposed Model

In this thesis project, as was mentioned earlier, a 3D voxel-level image segmentation is implemented for the purpose of brain tumor segmentation. For this project, the images employed for training are obtained using the MRI and are from the Kaggle dataset. Since these MR images are 3D, a 3D image segmentation is implemented herein. This is because it allows for a more convenient visualization and quantification of the scanned image. Further virtual analysis of the generated 3D models, for instance via computer simulation, or even producing a physical depiction of the subject using 3D printing, all benefit from the 3D segmentation of images [71]. Also, doing segmentation with 3D architectures allows for rich segmentations [72]. Specifically, 3D U-Net architecture which has won several awards in different segmentation challenges [73], [74] is chosen, for the brain tumor segmentation task in this paper.

3.1.1 The U-Net Architecture

The U-shaped architecture is made of an encoder-decoder design which can be generally thought of as an encoder network followed by a decoder network. It was originally devised to enable end-to-end training for biomedical image segmentation using a limited amount of data [66]. U-Net is a type of fully convolutional network (FCN) (Figure 3.6). FCN [40] is one of the first architectures without having fully connected layers at the end of the model and can be trained end-to-end, for individual pixel prediction (e.g. semantic segmentation). FCN is a general architecture that makes effective use of transposed convolutions as an up-sampling method that can be trained.

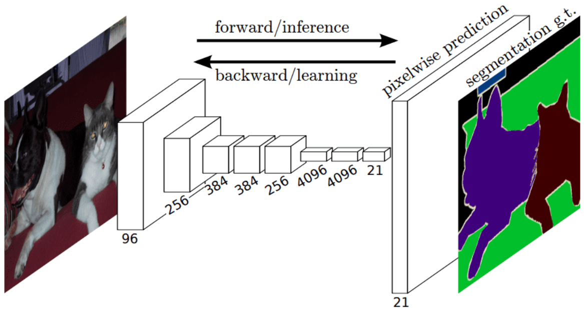


Figure 3.1: FCN Architecture [40]

Being one type of FCN, U-Net is a symmetric architecture [41] as depicted in Figure 3.7 that also implements up-sampling rigorously. In contrast to classification, where only the end result of the deep network is of importance, semantic segmentation not only necessitates pixel-level discriminations but also needs the means to project the discriminative features learned at different stages of the encoder during the time of downsampling onto the pixel space. The aim here is to semantically project the discriminative features (lower resolution) that the encoder learns during downsampling onto the pixel space (higher resolution) to ultimately attain a dense classification.

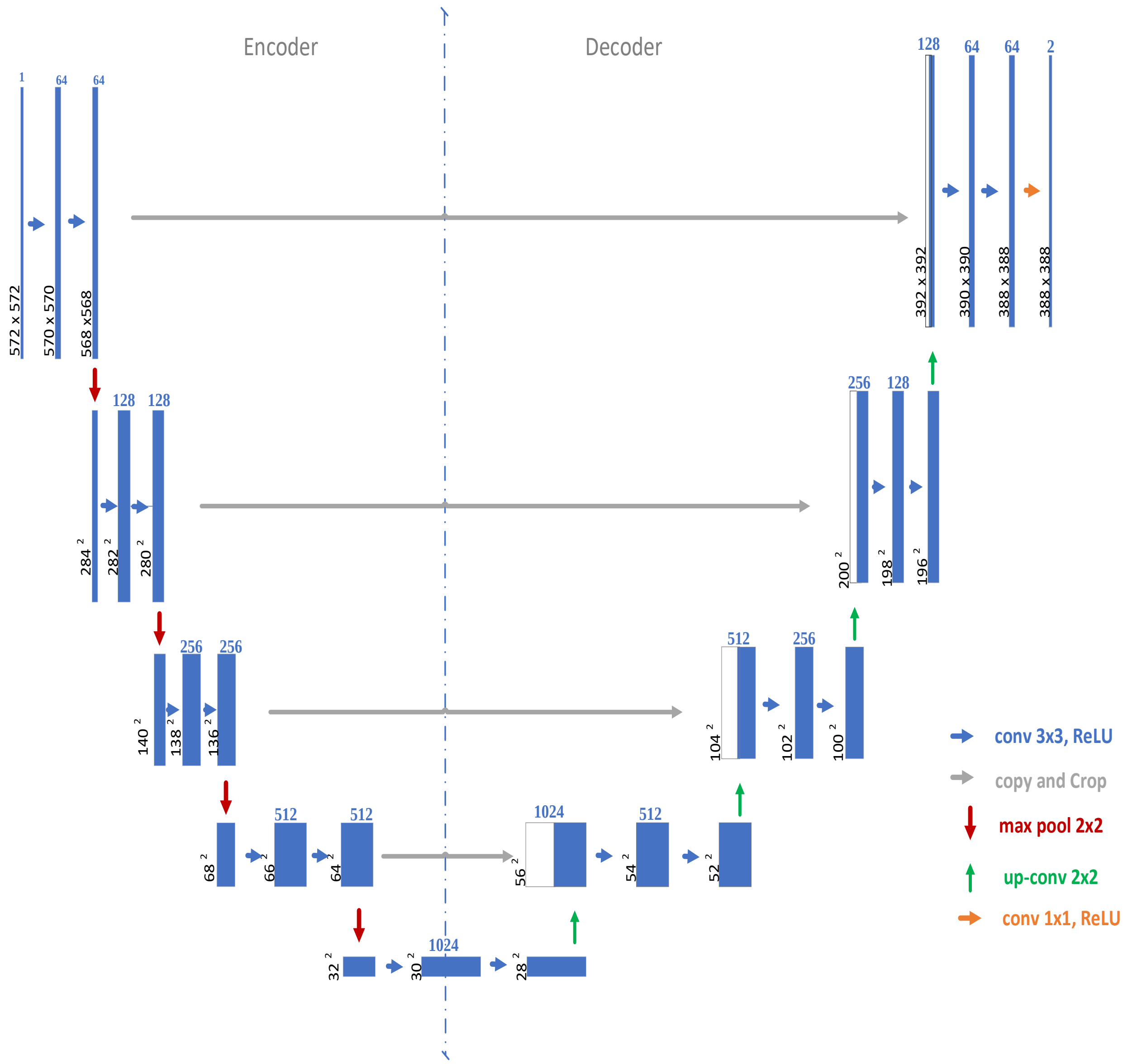


Figure 327: U-Net architecture. [41]

In summary, semantic segmentation varies from the typical classification owing to the fact that a pixel-level classification is needed, meaning each pixel is provided a class prediction by the model. This necessitates both discrimination at pixel-level and projection of the discriminative features learnt at multiple stages of the encoder onto the pixel space. This is done in the U-Net making use of concatenation operation of encoder side feature maps to those of the decoder side. The U-Net structure is a highly effective model typically employed for semantic segmentation of biomedical images.

3.1.2 The 3D U-Net Architecture

The 2D U-Net variant has great success in 2D biomedical image segmentation [41]. And it is still employed as a baseline method. But in order to find ways to fully make use of the 3D context of 3D images the 3D U-Net came to existence; indeed, 3D U-Net was introduced shortly after U-Net to process volumes [75]. Only 3 layers are shown in the diagram (Figure 3.8) but in practice, more layers could be used depending on the problem to be solved. For instance, for this paper, a 3D U-Net network of total five layers is employed.

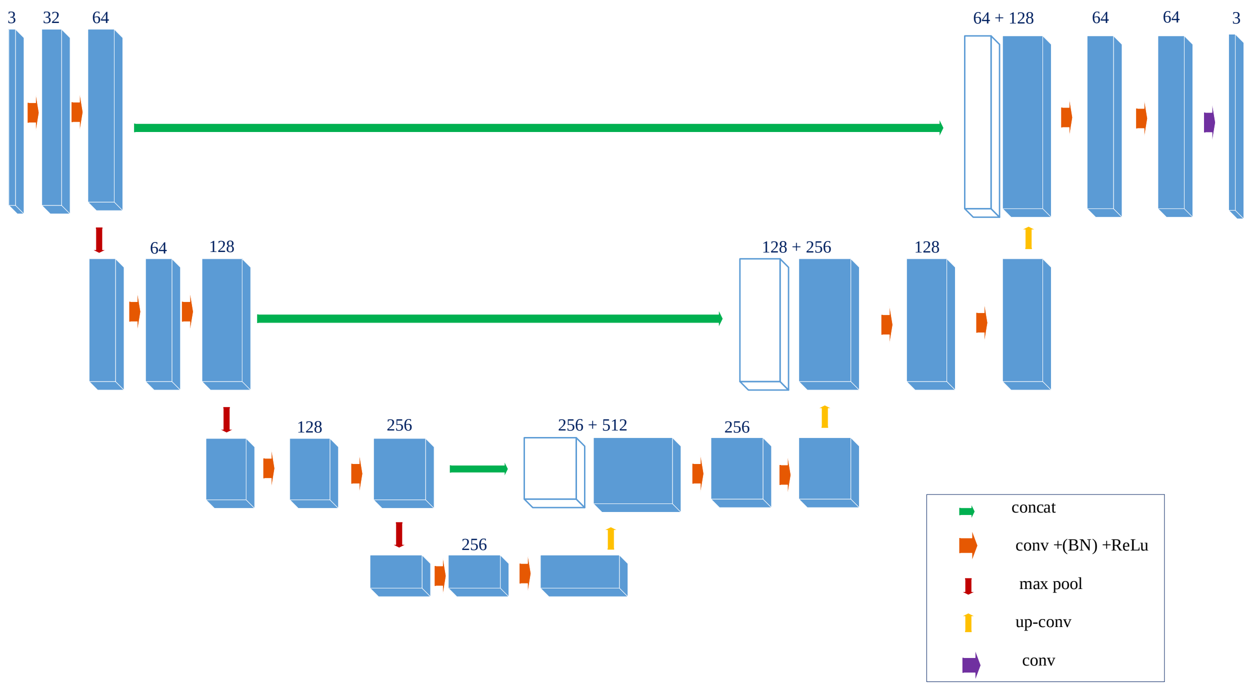


Figure 3.3: The 3D-UNet [75]

As it is shown in Figure 3.8, the 3D U-Net consists of an analysis path, on the left, and a synthesis path, on the right. The analysis path comprises within each layer two 3×3×3 convolutions where each convolution is followed by ReLU and then max-pooling of size 2×2×2 having strides of two in each dimension. In the synthesis pathway, however, each layer is made of an up-convolution of 2×2×2 with strides of two along each dimension. This is of course followed by two 3×3×3 convolutions each and ReLU. Shortcut connections from layers of equal resolution in the analysis path supply the necessary high-resolution features to the synthesis path. In the final layer, a 1×1×1 convolution decreases the number of output channels to the number of classes of the labels which is four for this project.

Now, for three-dimensional convolutional layers, things appear to be different – however, not quite divergent. Instead of three dimensions in the input image (the two image dimensions and the *channels* dimension), there will be four (the two image dimensions, the time/height dimension, and the channels dimension). Subsequently, the feature map will also be three-dimensional. This in turn results in the filter maps moving in three dimensions [76] as opposed to two. Meaning, the filters will move not only from the top to the bottom and left to right but also forward and backward. Thus, three-dimensional convolutional layers will be more expensive when it comes to the requirement of computational resources, but permit the retrieval of much richer insights.

Had our volume been isotropic, meaning it looks the same no matter which direction we cut, then doing the segmentation in one 2D slice at a time would have made sense. But for images like brain MRIs, it is hard to get the full story unless there is information on its context as it can be any shape or form. That is to say that it helps to know how a tumor looks 10 slices above and 10 slices below to understand its structure fully. As a result, the 3D U-Net architecture which provides the additional context of the brain tumors is employed.

3.1.3 Specifics of the Model Implemented

Specifically, the model herein is configured as follows:

- First, a batch size of 1 is used to train the constructed model, and then the model is trained with data of batch size 2 to see if there is a significant difference that can be brought about with larger batch size. This means that one (and then two) samples are fed forward through the network each time, generating predictions, computing loss, and optimization. The higher the batch size, the higher the efficiency with which the improvement gradient can be computed, but more memory is required. It was not possible to train with an even higher batch size due to memory capacity limitations.
- The model is trained for 100 epochs for Model1 and Model2 and 50 epochs for Model3. One epoch, or full iteration, signifies all available samples have been fed forward once and starting the process over again is possible.

- The initial learning rate, or the acceleration with which the optimizer (in this thesis project, the Adam optimizer) will strive to improve once the gradient is estimated, is initially set to 0.001.
- There are 4 classes (whole, edema, enhancing, and non-tumor) of segmentation, therefore, the number of classes is set to 4.
- 20% of the total training data is used as validation data, the remaining 80% of data is further split into training and testing data where 15% of it is used for testing during training and the other 85% is used for the actual training of the model (Figure 3.9).

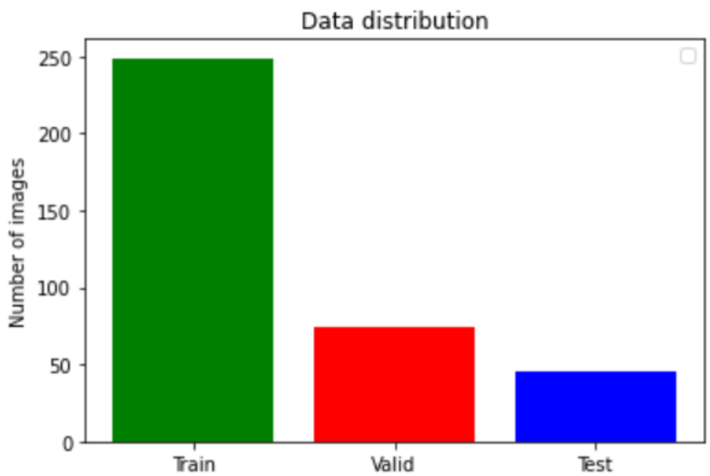


Figure 3.4: Split Ratio of Training, Validation and Testing Data

- Padding is set to 'same' meaning the model adds extra pixels on the edges during convolution operations.
- A dropout rate of 0.2 is employed as a regularization technique to counteract the model from overfitting. Dropout layers are added a couple of times before the final convolutional layers of a given level throughout the model.
- The activation function employed for all the 3D convolution operations is ReLU. The main reason why ReLU is employed here is for the reason that it is fast, simple, and empirically it seems to do the work well (i.e. empirically, early papers noted that training a deep neural network using the ReLU activation function appeared to reach convergence a lot more swiftly and reliably).
- In total the network has 22 convolutional layers. There are also four max-pooling and four concatenate layers in the downsampling and the up-sampling parts of the model respectively.
- Finally, verbosity is set to 1, which means that all possible output is sent to output screens. This helps in understanding what exactly happens during training but slightly slows down the process. Hence, when using those models in real-time, one may wish to turn the verbose mode off, but for training, it turns out to be valuable.

3.2 Tools Used

The major libraries that are employed in developing the AI model of this project are discussed in this section. The programming language used for this thesis project is python. This is so mainly because the TensorFlow framework uses it as a frontend programming language [77]. Another tool used by TensorFlow is Keras. It is indeed an amazing API for deep learning implementations that is embedded in TensorFlow. There are also other libraries imported and used while building the U-Net model of this project; the crucial tools applied in creating this thesis project are explained in the following sub-sections.

a. Python and TensorFlow

Python as a programming language is selected to be used for this project not only is it simple to understand and there are very many resources available to study it for free, but it is also the front end programming language for TensorFlow. TensorFlow is an open-source platform for implementing machine learning projects; moreover, it has end-to-end functionality. TensorFlow employs Python to deliver a convenient front-end API for the purpose of building applications with the framework. Nodes and tensors in TensorFlow are Python objects, while TensorFlow applications are themselves Python applications [77]. Nevertheless, the actual mathematical computations are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries [78]. Python only serves as traffic police directing the traffic between the pieces, and supplies high-level programming abstractions to hook them together.

TensorFlow itself is in fact one of the best frameworks available [79] that takes input data in the form of multi-dimensional arrays with higher dimensions namely tensors. Multi-dimensional arrays are quite practical when it comes to handling large amounts of data. TensorFlow is inherently based on data flow graphs having nodes and edges. TensorFlow is rather easy to execute in a distributed format for the reason that its execution mechanism is in the form of graphs. Hence, TensorFlow codes are typically executed across a cluster of computing devices with GPUs [80] (Figure 3.1). Moreover, there are very many models of deep neural networks that can be fully trained and run using TensorFlow. Most importantly, TensorFlow allows at-scale production prediction [78], with the same models used for training. Also, TensorFlow applications can be run on almost any target [79] that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs, or GPUs.

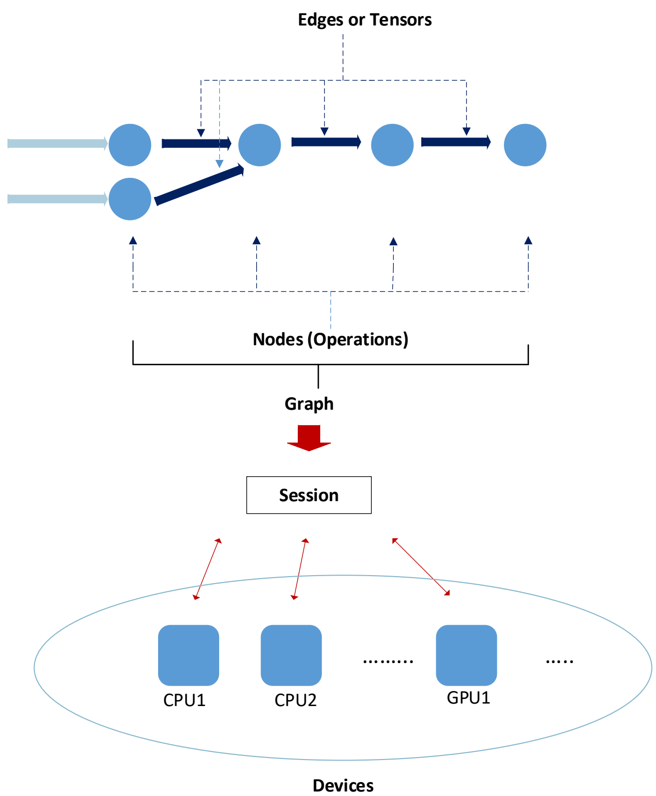


Figure 3.5: Dataflow graph of TensorFlow [81]

b. Keras

Keras is a high-level, deep learning API that was also built by Google for the implementation of neural networks [82]. Written in python, Keras is used to make the implementation of neural work applications much easier. It allows multiple backend neural network computations. Keras provides a python frontend with a high level of abstraction whilst giving the alternative of multiple back-ends to choose from for computation purposes (Figure 3.2).

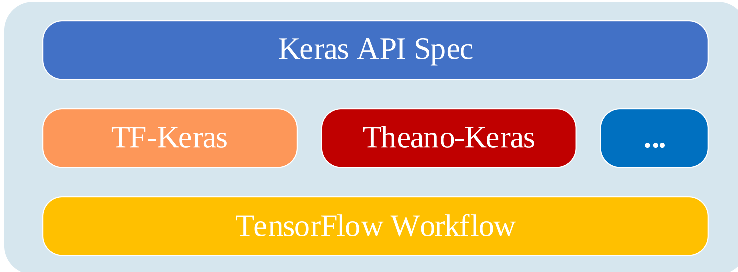


Figure 3.6: Keras backend [83]

Because Keras is deeply integrated with TensorFlow [82], developing customized workflows is achieved with ease and is relatively faster than it is for other frameworks [84]. Besides, it is modular in nature making it flexible, expressive, and a great suit for innovative research, hence using Keras on top of TensorFlow is a wise choice of API in developing this project.

Building Models in Keras

The diagram below shows the basic steps involved in building a model in Keras. These are the steps taken in building the 3D U-Net model of this project. Each is explained briefly as follows.

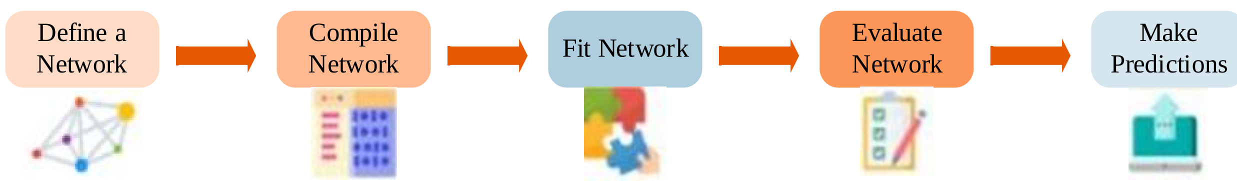


Figure 3.7: Building a model [28]

1. **Define a network:** Here, the different layers are defined in the model and the connections between them. In the case of this project, the 3D-Unet model is built, which comprises of several convolutional and max-pooling layers. Also, the model consists of an up-scaling path making use of transpose convolutions. The model is trained with data of batch sizes 1 and 2. The structure of the network developed herein was discussed in detail in section 3.1.
2. **Compile a network:** To compile a code means to transform it into a form that is appropriate for the machine to understand. In Keras, the model.compile() method performs this function. To compile the model, first the loss function is defined which calculates the losses in the model. The purpose of this loss function is to compute the quantity that the model should aim to decrease during training. For this project, "categorical_crossentropy" which calculates the crossentropy loss between the labels of the ground truth and predictions [85] made by the U-Net model is employed. This crossentropy loss function is normally employed when there are two or more label classes and in this case, there are four classes ergo this type of loss is

implemented. Output labels are expected to be supplied in one_hot encoded format and that is what is done in the code.

Moreover, another argument necessary to compile a model is the optimizer argument. Adam optimizer which is a stochastic gradient descent method [86] is what is chosen for this project with an initial learning rate of 0.001. Furthermore, there are several functions defined as ‘metrics’. The functions performing the metric operations are implemented locally. These metrics are there in order to observe the performance of the system in the prediction of each pixel as belonging to each label/class. For instance, system efficiency indicators such as precision and sensitivity are implemented; the only difference between the loss function and these metrics is that they are not used during the training of the model.

3. **Fit the network:** model.fit() function of Keras is used to fit our data following a successful compilation. The data generator supplies the data on the fly and the model is fit with batch sizes of 1 and 2.
4. **Evaluate the network:** After fitting our model, there is a need to evaluate the performance of our model. Evaluation is done using data that is set apart for this purpose, which is data the model will not see during training.
5. **Make Predictions:** model.predict() function is employed in order to make predictions using our model. This is to see how the model developed does in real time after all work in training and evaluating the system is done.

c. NumPy

NumPy happens to be a foundational package for scientific computing using Python. It is a Python library that provides a multidimensional array object and various derived objects (including masked arrays and matrices) [87]. Also, a variety of methods for fast operations on arrays, including mathematical, logical, shape manipulation, selecting, sorting, I/O, basic linear algebra, discrete Fourier transforms, basic statistical operations, random simulation, and a lot more are provided by this library.

There are two significant traits of NumPy arrays that made it the choice of library for this project. The first major useful trait is that NumPy arrays facilitate advanced mathematical and other types of operations on massive amounts of data [88]. Typically, such operations are executed more efficiently and with less code than is possible when using Python’s built-in sequences. The second useful thing about this library is that a growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; consequently, using NumPy arrays makes things easy to attain any help needed during code development. Hence, there are several points where NumPy arrays are employed in this project.

d. Nilearn and NiBabel

Nilearn is a Python package employed for neuro-imaging purposes and is one of the many Python “nipy” packages [89]. Nilearn supplies to developers tools of analysis techniques like functional connectivity and (machine-learning-based) multivariate “decoding”. Also, more “basic” tools like image manipulation and visualization are provided by it. The tools employed for data visualization of the brain in Nilearn are grouped within the plotting module. The plotting module is mainly considered to be one of Nilearn’s most useful attributes. Nilearn has a set of plotting methods to visualize brain volumes that are fine-tuned to specific applications. Specifically, the plot functions plot_anat(), plot_epi, plot_img(), and plot_roi() are implemented herein to visualize the brain MRI volumes in different formats allowing a very good understanding of the data. On the other hand, NiBabel is a Python tool that allows read (e.g. nib.image.load()) and writes to a few common medical and neuroimaging file formats.

In this project, both NiBabel and Nilearn are employed for reading and visualizing the data in the BRATS dataset. Visualizing the data is quite important before attempting any form of training and these libraries are quite useful for this task.

3.3 Visualizing the Data

By making use of the NiBabel and Nilearn libraries the images are visualized and investigated thoroughly in order to understand the BRATS 2020 dataset well. Understanding the brain images of the dataset provides insights into the shape and position of the tumors in our images. Moreover, the selection of algorithms and post and preprocessing techniques is made much easier with knowledge of the properties of the images used for training. In this section, the techniques employed for visualization herein are shown.

First, let’s see the load function. To load the images from the database, NiBabel [90] library is used and the function nibabel.load is utilized. This makes life much easier as there is no need to implement this function that fetches the images from the database from scratch. Figure 3.4 is a brain image of a single person. All the four volumes of images (i.e. T1, T2, flair, and T1ce) and of the segmented mask are loaded from the database by this load method.

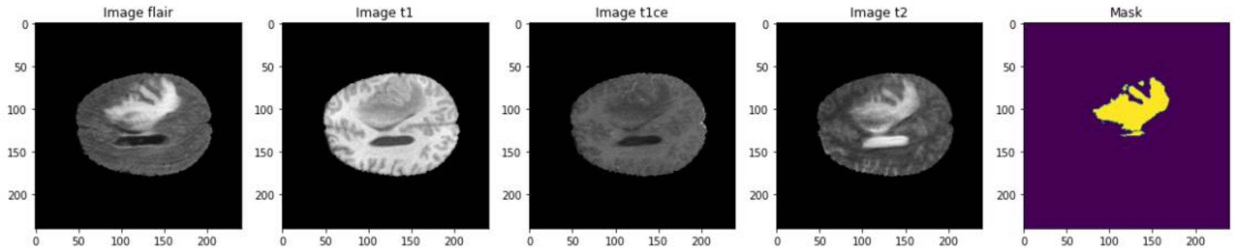


Figure 3.8: Four types of imagings with segmentation map fetched by nibabel.load

Next, different visualization methods employed within this project are shown as follows (Figure 3.5 a-d). These functions are specifically created for the purpose of visualizing brain images and all plotting methods used here plot three cuts (i.e. Frontal, Axial, and Lateral cuts) by default.

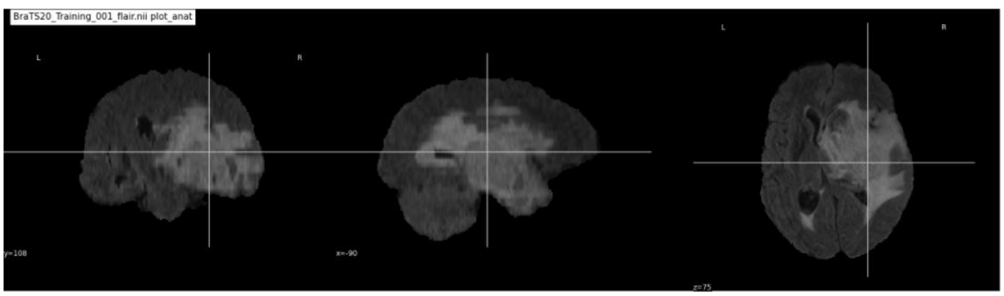
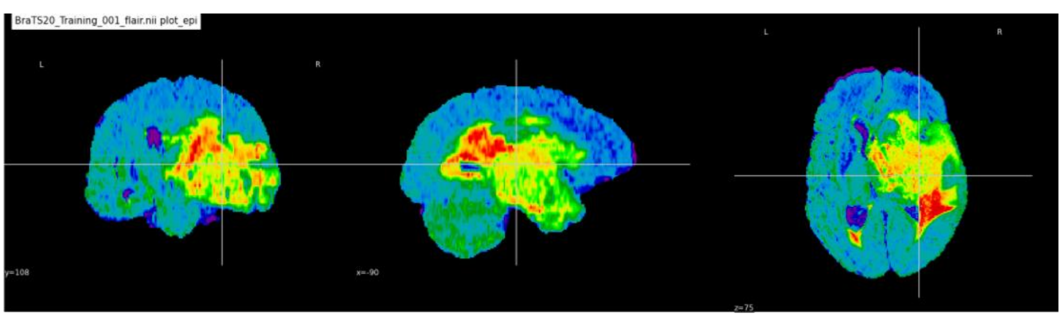
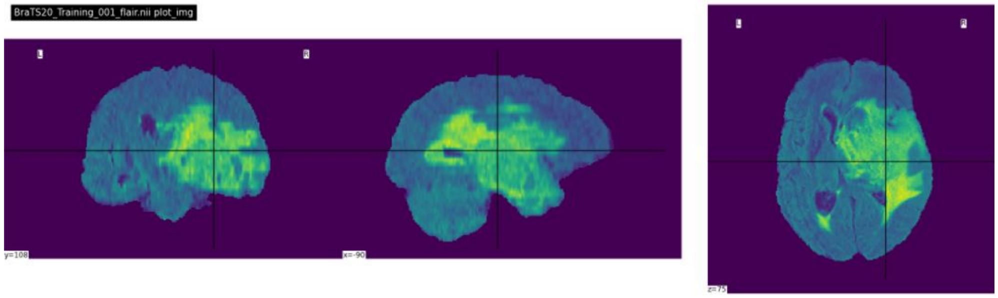


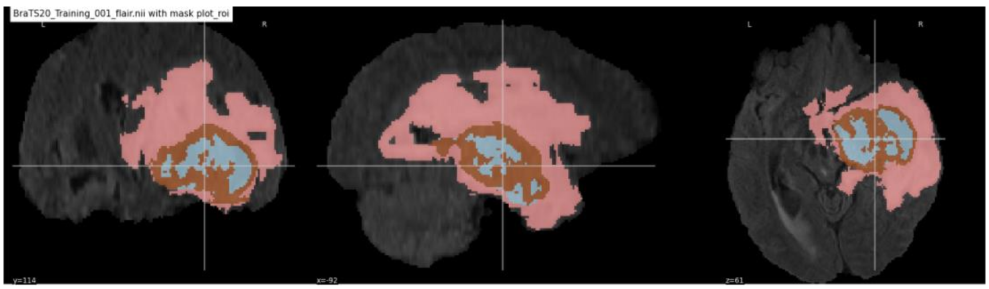
Figure 3.9 Different visualization methods of MRI brain scans (a): plot_anat - Anatomical plot of cuts of image 001 in database; function provided by Nilearn.



(b): plot_epi - Echo Plannar Imaging (EPI) plot of cuts of image 001 in database; function provided by Nilearn.



(c): plot_img - Plots of cuts of image 001 in database; function provided by Nilearn.



(d): plot_roi - Plots of cuts of an ROI/mask image of image 001 in database; function provided by Nilearn.

3.4 The Data Generator

As the ability to harness an ever-growing bulk of data expands, large datasets are increasingly fitting seamlessly into the lives of human beings. These datasets are highly memory consuming to the extent that they give rise to the need to create a different way of handling them. The same is true for this thesis project. The dataset used herein is that of BRATS and every patient's MRI contains four sequences with each sequence having the size of 240x240x155. There is a total of 369 patients' data and loading all of these on memory at a given time is highly memory consuming and unpractical. This is due to the fact that loading all of the training samples causes issues, such as system shutdown, as the dataset will not be able to fit runtime memory.

In a number of cases, even the most state-of-the-art devices and configurations will not have a sufficient amount of memory space to handle the data how it was normally done previously (i.e. loading the whole dataset at once employing just a single line of code) [91]. This is the reason why there was a need to find another way to perform the task efficiently. In this subsection, it is explained how data was generated on multiple cores in real time and was fed right away to the deep learning model.

Now, let's go through the details of how this Python class named `DataGenerator` is built, which will be used for real-time data feeding to the 3D-UNet model. The first function in this data generator class is the initialization function. It is also where the class inherits properties of keras.utils.Sequence so that nice functionalities such as multiprocessing can be leveraged. Relevant information about the data is passed as arguments to this function. The information includes dimension (e.g. a volume of length IMG_SIZE=128 will have `dimension=(128,128,128)`), number of channels, number of classes, batch size, or the decision whether to shuffle the data at generation. Also, important information such as the list of IDs that is to be generated at each pass is stored to an appropriate parameter within this method.

Next comes, the method `on_epoch_end` (Figure 3.10) which is triggered once at the beginning and once at the very end of every epoch. If the shuffle parameter's value is True, then a new order of arrangement in the dataset is generated at each pass. Otherwise, the same order of the dataset is maintained during the data generation stage. Shuffling the order in which images are fed to the model being trained is quite useful to have batches that do not look alike amongst different epochs. Eventually, this results in the robustness of the trained model.

```
def on_epoch_end(self):
    'Updates indexes after each epoch'
    self.indexes = np.arange(len(self.list_IDs))
    if self.shuffle == True:
        np.random.shuffle(self.indexes)
```

Figure 3.10: Method on_epoch_end always run at the end of every epoch

Another method that is located at the heart of the whole data generation process is the one that achieves the most significant job: actually producing/loading batches of data. The function which is private and performs this task is named `__data_generation__`. `This method` takes the list of IDs of the target batch as an input argument.

During data generation, this method reads the images whose corresponding IDs are provided via its input parameter named **batch_ids**. The data is fetched from the BraTS 2020 dataset on

Kaggle. For the purpose of loading the images a very useful library known as NiBabel is employed. After reading a certain image with a given id, it is loaded slice by slice into a usable numpy variable. Note that each slice of 240x240 images is resized to that of 128x128 before loading. This is to make the image size convenient for the (U-Net) network during upsampling and downsampling stages.

Moreover, only 128 slices are loaded for a given MRI volume which is not only a good size for our structure, but also there is no need to load all the 155 slices of volumes. That is because there are only usable brain images after about the first 21 slices. The slices before that are mostly empty. Therefore, slices starting from the 22^{nd} slice are loaded up until the 150^{th} slice; this makes the generated input volumes a size of (128x128x128). On the other hand, the segmented masks are one-hot-encoded making use of a TensorFlow library (tf.one_hot()). Finally, the function **__data_generation__** returns the loaded input along with the one-hot-encoded output mask which has the shape of (128, 128, 128, 4).

The following method is yet another function in the DataGenerator class namely **__len__**. This is where the total number of batches per epoch is set. Normally, this value is set to:

$$\frac{\#\ samples}{batch\ size}$$

This is done to enable the model to see each training sample at most once per epoch.

The final function that is used by the generator class is the **__getitem__** method which is used to load a certain batch of images when the batch corresponding to a given index is called. When provided the index of the batch to be generated, this function fetches the batch of data at that index.

It is observed that during the training phase, data is fetched from the dataset in parallel by the CPU and fed directly to the GPU supplied by Kaggle. The data generated in this case is volumes of images of size (128,128,128) and the output labels that are one-hot-encoded. This class is very useful as it loads the data on the fly instead of attempting to load 40 GB data to memory at a time.

3.5 Evaluation Metrics Used

So the model constructed herein performs semantic segmentation. It is indeed a crucial task. But how does one determine how good of a model it is after training the deep learning model? For the evaluation of our model, there are a number of choices available. In this subsection, the crucial metrics useful to measure and ensure that the model trained herein performs well are explained.

3.5.1 Dice loss coefficient(s)

The Sørensen–Dice coefficient is a statistic employed to quantify the similarity of two sets. Two botanists, by the name of Thorvald Sørensen [92] and Lee Raymond Dice [93], individually invented it and published in 1948 and 1945 respectively. The index is known by a handful of other names, especially Sørensen–Dice index, Dice's coefficient, [94] Sørensen index, and Dice similarity coefficient (DSC). When used on Boolean datasets, it can be formulated as:

$$DSC = \frac{2TP}{2TP+FP+FN}$$

where true positive (TP), false negative (FN), and false positive (FP).

When DSC serves as an evaluation metric in the task of segmentation, it can be visualized as the size of the overlap of the two segmentations (i.e. the segmented output of the model and the provided mask (ground truth)) divided by the total size of both objects. The Dice score is not only about measuring the number of positives detected correctly but also the number of predictions that are found to be false positive by the model. The DSC penalizes the system for the false positives found. Meaning, the Dice score also inflicts penalty on the system for the positives that the algorithm/method/model was unable to find. DSC spans between 0 and 1 and is the quotient of similarity [95]. The Dice score coefficient is a good indicator of performance for semantic segmentation hence is implemented herein.

3.5.2 Mean Intersection over Union (Mean IOU)

In general, the mean IOU metric which is also known as the Jaccard index [96] is a yardstick to measure the overlap between the predicted output of segmentation and the ground truth or mask provided in the dataset (Figure 3.11). For a given image, the IOU represents the similitude between the predicted region and the ground-truth region of an object existing within the image, for instance, the object in the case of this project can be the brain tumor to be detected. IOU is formally defined as the area of the intersection divided by the union of the mask and ground-truth regions as shown in Figure 3.11. Also, it can be represented as follows:

$$IOU = \frac{TP}{TP+FP+FN}$$

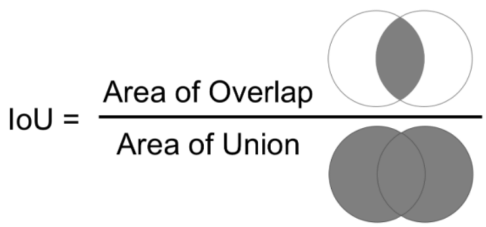


Figure 3.11: Pictorial representation of the Intersection over Union (IOU) metric [97]

Mean IOU is a common evaluation metric for semantic segmentation of images [98], which first computes the IOU for each semantic class and then computes the average overall classes. A Mean IOU value of a system greater than 0.5 is commonly considered a "good" prediction.

After choosing the metrics for the evaluation of the system, which are explained above, how does one actually include them in the code? Keras has a convenient way of listing the metrics to monitor during the training of the model [85]. This can be done by specifying the "*metrics*" argument and providing a list of function names (or function name aliases) to the *compile()* function of the model.

3.6 Callback Functions

TensorFlow callbacks are functions or blocks of code that are executed during a specific instance of time in the process of training a Deep Learning Model [82]. As deep learning models are getting more and more complex and resource-intensive the training times have also grown notably. Therefore, it is common for models to cost hours of training time. In the regular workflow before training the model, all the options and parameters like learning rate optimizers are set before the model training is started. Once the training process is begun there is no way to pause the training in case there's a need to change some parameters. In some cases, after the model has been trained for several hours at the later stages of the training a tweak of some parameters becomes necessary however it is impossible to do so with the ordinary workflow.

This is when the callback functions of TensorFlow save the day. First, the callback functions are defined then they are passed when calling the model.fit() method [31]. These callback functions perform a number of parameter tweaks or even terminate the training when no further improvement is observed in further training the model. Here the callbacks chosen for the specific task of brain tumor segmentation being performed by the 3D U-Net model are discussed as follows.

3.6.1 Stop and Resume: Logger and Model Checkpoint Classes

In an ideal world, Keras models would be trained in one pass, utilizing as much GPU and CPU power as needed. However, the world in which models are trained is far from ideal. Saving the infrastructure of a model eases the transition back into training for the neural network and will be more efficient than starting from scratch. In this subsection, methods that enable stopping and continuing of training and even ways of adjusting training at later times are discussed. This continuation of trainings is achieved with model checkpoints.

When it comes to the aim of attaining stop and resume functionality to the code, first of all, utility functions are constructed. The first utility function generates an output directory that has a unique name. This technique helps in the organization of outputs during multiple runs of the model. Following this, a Logger class is provided in order to route output to a log file located in the output directory. After that, some very basic training parameters and destination filename of outputs are defined.

Then a checkpoint class named MyModelCheckpoint is created. Normally, Keras provides a prebuilt checkpoint class called **ModelCheckpoint** that contains most of the functionality that is desired here [83]. Therefore, the local class MyModelCheckpoint inherits this class of Keras. These desired functionalities include its ability to run in parallel with training in order to save a model or weights of a model(in a distinct checkpoint file) at a given time interval, which later enables the loading and continuation of training from where it stopped. Also, this saving of a model within a provided interval can be performed repeatedly during the progression of training.

Moreover, it provides a couple of options to choose from; for instance, one can choose to keep only the model with “best performance” or all the model parameters by the end of each epoch regardless of performance. Therefore, the newly created class MyModelCheckpoint inherits this class and makes use of the important functionalities inherent to this class. In addition, other parameters like the state of the optimizer are saved by functionalities of MyModelCheckpoint class by pickling them at the end of each epoch. Furthermore, when there is a need to save other important parameters they can just be added to our pickled parameters making our system more suitable for future expansion.

Once the utility functions are defined and checkpoint class is written, the model is built as described in section 3.1.3. Training the model, however, requires a few extra considerations. The maximum number of epochs is specified, as usual; nevertheless, the user is allowed to also select the starting epoch number for training continuation. Now, the training can proceed using the Logger class to write the output to a log file in the output directory. By the end of each epoch there exists a new checkpoint saved which includes two files with their own set of functions:

- Pickle files contain the state of the optimizer.
- HDF5 files contain the saved model.

During the continuation of training, the paths to both the HDF5 and PKL files are necessary. Since Kaggle does not allow writing directly to the inputs directory at run time, saved models are originally saved to the working directory which is volatile. Therefore the saved files are downloaded before they are lost and are uploaded back to the permanent Kaggle database. After that, training can proceed starting from the epoch of our choice making use of the uploaded files’ path. The method that actually loads and extracts information from the HDF5 and PKL files is the utility function named load_model_data. This function loads the HDF5 and PKL files and then recompiles the model based on the PKL file. Based on the type of optimizer employed, it might not be mandatory to recompile. Finally, the model can be trained for additional epochs starting from where it stopped previously. In order to do that a new directory is generated and the Logger class is employed here too to be able to direct outputs to a log file.

The obstruction of neural network trainings could happen for reasons one cannot control. Sometimes this stoppage can be abrupt, such as a power failure or a network resource shutting down. Since Microsoft Windows is the operating system used herein, the training can also be interrupted by a high-priority system update. Additionally, the model is being trained on a Kaggle notebook; the training will be automatically stopped if it takes more than 9 hours [99] which is the case for the training of the U-Net model built in this project. Because of all of these uncertainties, it is best to save the model at regular intervals. Therefore, it was chosen to save the model along with other parameters at the end of every epoch as explained in this subsection. This allows a smooth restart of training starting from any one of the epochs saved.

3.6.2 Early Stopping

While training a model, weights of the neural networks are upgraded in order to improve the model's performance on training data. Initially, the improvements made using the training set correlate positively with changes observed while employing the test set. However, there comes a point where one begins to overfit on the training data and further "improvements" will result in lower generalization performance [28]. This is known as overfitting. Early stopping (Figure 3.12) is a method used to stop the training before overfitting happens.

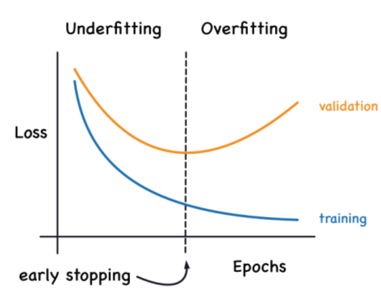


Figure 3.12: Overfitting, underfitting and early stopping [100]

Early stopping is employed herein making use of a TensorFlow callback function named `tf.keras.EarlyStopping` [100], hence it is a Keras function. This early stopping callback function is added to the list of the callback functions and is passed during the fitting of the model. Then the model is trained with early stopping. Early stopping is triggered by monitoring if a certain value (for example, loss) has improved over the latest period of time (controlled by the patience argument) [82]. Training is terminated when the monitored metric has stopped improving. Here, the loss variable is monitored and if this variable does not improve by a least 0.0001 for a consecutive 5 epochs the training is terminated.

3.6.3 Reduce on Plateau

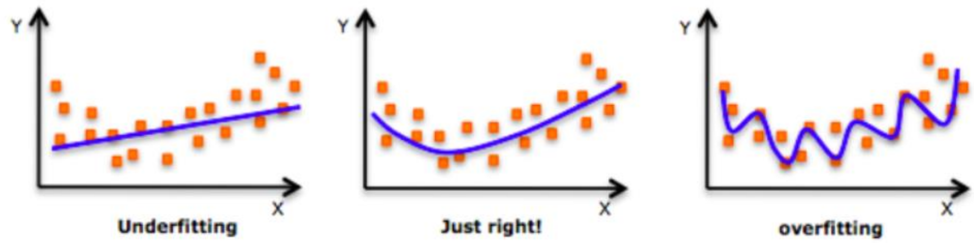
One of the callback methods employed to improve learning once it stagnates is decreasing the learning rate of the model. Mostly, it is reduced by a factor of 2 to 10. This callback method keeps track of a quantity and if no advancement in this quantity is observed for a specified number of epochs, the learning rate is lessened [31]. This entails that once the monitored metric has stopped improving and reached a plateau the training rate is further lowered. Here, validation loss is monitored, and when there are no improvements in it for 5 epochs, the learning rate is reduced.

3.6.4 CSV Logger

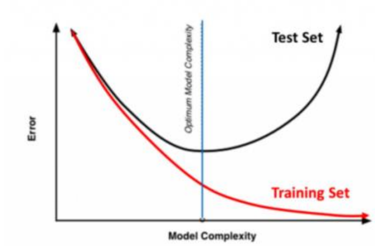
Callbacks are a crucial kind of object TensorFlow and Keras that are devised for the purpose of monitoring the performance in metrics during training [82]. Also, they allow some action to be taken that depends on the achievement of the model with regard to the tracked metric values. Specifically, CSVLogger is a callback function that permits the streaming of epoch results into a CSV file [85]. This helps in tracking the model performance so as to take necessary actions. In this project, the CSVLogger function saves the value of each metric defined earlier into a string format representation for every element in the dataset within a given epoch. This is quite useful in order to track and see what is happening to the metrics with each brain scan in the training dataset being fed to the model; this in turn helps to adjust the model and its parameters to attain good metric values.

3.7 Regularization Techniques Employed

Overfitting is a common obstacle in the process of training neural networks. A universal explanation of overfitting is that the model has reached a point where it is too complicated and its performance is much better in handling training datasets (resulting in very small train errors) than dealing with testing datasets (resulting in very high testing errors) [28] i.e. the model does not generalize well on previously unseen data. This can be observed well in Figure 3.13 (a). As we move from left to right in this image the model attempts to learn too well the details and ends up picking up noise from the training data, which ultimately gives rise to poor performance of the model on the unseen data. That is to say while going toward the right, the complexity of the model rises such that the training error reduces while the testing error increases. This is also exhibited in the figure below (Figure 3.13(b)).



(a).



(b).

Figure 3.13: (a) Illustration of over-fitting a model (b) Training vs. test set error [28]

Regularization is a method that makes slight alterations to the learning algorithm such that the model does not overfit and generalizes better. This means that the model's performance is boosted and it achieves higher performance on unseen data. Several techniques can be employed to prohibit overfitting such as early stopping and dropout [101]. Another solution for handling overfitting [102] is employing data augmentation for models that usually require an incredibly enormous amount of data. In cases like brain tumor segmentation where a large dataset is not available, data augmentation comes in handy to prevent the model from overfitting by providing the model with extra data. For this project, these three methods of regularization are implemented namely dropouts, early stopping and data augmentation. Early stopping has been discussed in section 3.6.2. The remaining two are discussed thoroughly in the coming sub-sections.

3.7.1 Dropouts

This technique is one of the most appealing types of regularization methods [91]. It helps attain very good outcomes and is therefore the most frequently employed regularization technique in the area of deep learning. The idea behind Dropout is to get rid of random input and hidden units at the time of the processing of each layer. Being aware that every node in a given layer is fully connected to the layer above, a few of these connections are taken off randomly during the training of the neural network (Figure 3.14).

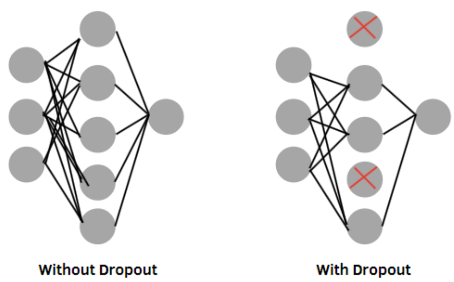


Figure 3.14: Illustration of Dropout [91]

It’s worth noting that using dropout alongside other techniques like batch normalization might interfere with it [83]. BN (batch normalization) normalizes each batch with its own set of mean and standard deviation. On the other hand, dropout randomly gets rid of a predefined ration of units in a neural network to prohibit overfitting. Therefore, employing the dropout layer and batch normalization layer — specifically putting them next to each other— causes disharmony among those two. Normally, the regularization effect of BN is little and is basically a side-effect of the normalization process. Conversely, Dropout is a rather simple yet strong regularizer to address the overfitting problem. On account of this, in this project, only dropout is employed with a rate of 0.2 quite a few times.

3.7.2 Data Augmentation

Overfitting represents a phenomenon where a neural network performs quite well on the training data but fails when it encounters new data coming from the same problem domain. Overfitting is a result of noise within the training data that is picked up by the neural network and identified as an underlying property of the data in the process of training. The simplest way to diminish overfitting is to enlarge the size of the training dataset [31]. Indeed, the achievement of most ML models, and deep learning models in particular, relies on the quality, quantity, and relevancy of the data used for model training. Nevertheless, an inadequate amount of data is one of the most common obstacles faced in the implementation of machine learning. The reason being the collection of data can be costly and tedious in many instances which is also the case for this project.

However, how can one train a “state-of-the-art” neural network for this thesis project with the meagre amount of data available from BRATS? Data augmentation can be employed to decrease reliance on training data collection and preparation in order to build more accurate machine learning models faster. Data augmentation is a set of methods employed to raise the amount of data by slightly modifying copies of already existing data or creating newly synthetic data from existing data. By executing data augmentation one can deter a neural network from learning irrelevant noisy patterns, essentially boosting overall achievement [81].

In this project, online augmentation or augmentation on the fly is employed. This method is typically chosen for larger datasets [103], as the explosive raise in size can’t be afforded which is why it is also chosen herein. The data augmentation is performed within the DataGenerator class. This class inherits the Keras ImageDataGenerator class and extends it with some local functions accordingly to the needs of the current project as explained in section 3.4. The DataGenerator class generates data while also performing data augmentation on the fly.

Some basic but powerful data augmentation techniques are implemented herein, namely flip, rotation, scaling, and Gaussian noise [104]. Here, only a horizontal flip is implemented because there are no real-world scenarios where a brain scan is fed to the model vertically inverted. The horizontal flip helps the model generalize well even if the original images in the training dataset for instance mostly contain the brain tumor on one side of the brain scan. When it comes to rotation the MRI brain scans are rotated by 20 degrees. This is to account for brain scans that are slightly tilted during the scan. The empty space created during rotation is filled with the extension of the edge pixels.

Moreover, the input images used for training are scaled inwards with a scale of 0.2, meaning the original image size is reduced by a factor of 0.2. The edge pixels are then extended until the image reaches its original size for this particular project. Additionally, since overfitting often occurs when a neural network attempts to learn high frequency features (patterns that occur much) that may not be valuable a Gaussian noise, having zero mean, which fundamentally has data points in all frequencies is applied to the dataset. This effectively misshapes the high frequency features. But this also implies that the lower frequency components (often, the intended useful data) are also distorted, however, the neural network can learn to see past that.

In conclusion, data augmentation is a popular means to enlarge the dataset size by performing different changes to the original dataset. Currently, due to various reasons, including labeling costs and patients' privacy, available datasets on brain tumors are in relatively small scales. Therefore, a number of simple data augmentation techniques are employed on the fly to increase the dataset size.

3.8 Flow Diagram of the Overall System

As can be seen from the following image (Figure 3.15) all the functions and classes make up this system of software. It encompasses of functions starting from the data generator generating the appropriate size of data and also applying data augmentation, to the trained model and actual segmentation performed on the dataset, to evaluation metrics being calculated and plotted by the end of training.

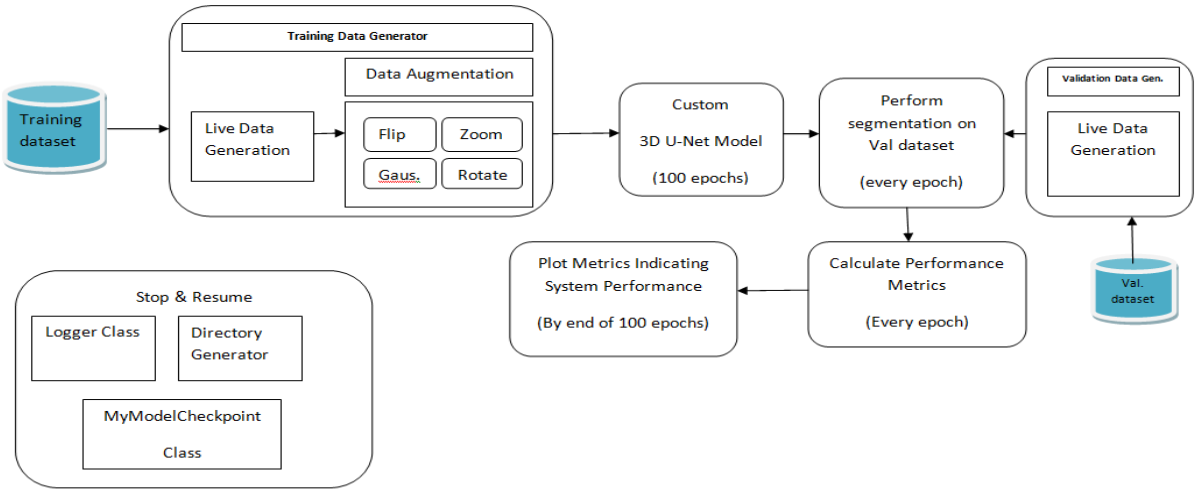


Figure 3.15: Overall flow diagram of system

Chapter 4
Results and Discussion

In this thesis project, an automatic brain tumor segmentation using Deep Learning AI was successfully implemented. The model developed is of the type 3D U-Net. In this section, the performance of the constructed model is discussed well and is presented alongside graphs of evaluation metrics. Also, the challenges faced during the development of this thesis project are presented. Furthermore, the effects of preprocessing and post-processing methods employed herein are explained thoroughly.

In the upcoming sub-sections, three models trained with different refinement techniques are explained. Also, a handful of metric graphs are drawn too. Moreover, the performance of the models developed herein is compared with results obtained by other researchers in the brain tumor segmentation area. In the end, a high achievement level model is trained as a result of the development of a good model, and each operation is executed to improve performance.

4.1 Dataset

The dataset employed herein is from BraTS 2020 dataset on the Kaggle database. The first thing done here is exploratory data analysis [EDA]. The main thing here is the first word exploratory. This is a step where we go through the dataset available to understand it so we can build a model the best way possible. In the attempt to understand the database at hand, it became clear that there are a total of 369 multimodal images which are of size 240x240x155. These images comprise both low grade and high grade glioma. The volumes were acquired from a number of institutions using multiple scanners (n=19) which helps the trained model to generalize to new images well. The format of these brain scans is NIfTI (.nii.gz). Moreover, for each patient, there are four MRI sequences and each compliment the other in terms of accurately showing the whole glioma within the brain.

These four sequences of volumes are accompanied by their respective segmentation mask. These masks were attained first by a segmentation performed manually by one to four raters conforming to the same annotation protocol. After that, their segmented outputs were confirmed by experienced neuro-radiologists. The annotations include the GD-enhancing tumor (Label 4 — ET), the peritumoral edema (Label 2 — ED), and the necrotic and non-enhancing tumor core

(Label 1 — NCR/NET). The provided brain scans are interpolated to the same resolution (1 mm^3) and are skull stripped.

These MRI brain scans from Kaggle were first visualized and studied well as mentioned above. This step is crucial for the selection of the right model architecture because understanding the underlying pattern of data is critical for identifying which type of model would successfully perform the segmentation task. As can be seen from the figure below (Figure 4.1), the brain lesions are of different size and shape. In order to understand the dataset thoroughly, the brain imaging libraries NiBabel and Nilearn were employed. Then the segmentation mask which originally contains labels 1, 2, and 4 is converted into 1, 2, and 3 to make it convenient to one-hot encode.

After this, data is fetched and delivered to the model during training by the data generator. During this process, the data is cropped and resized to a size of 128x128x128. This is a convenient size for the designed model. Furthermore, as the brain volumes are from different scanners they were found to have different ranges in value. For this reason, all voxel values were scaled into a range between 0 (minimum) and 1 (maximum) before supplying the images to the model. At first, the NIFTI (.nii.gz) format image dataset is used directly to train the model (Model1, Model2, and Model3).

4.2 Implementation

The TensorFlow version 2.8 is used to design and test the segmentation model of this project. Furthermore, an Intel Core i7 laptop computer with 8 GB RAM and a 2.5 GHz processor was employed in writing and running the code for this project. However, as the images employed are 3D as is the model designed, the processor of the laptop could not train the model. When training with even a batch size of 1 was attempted it caused the laptop to freeze and a manual system restart was necessary. Therefore, a processor of Kaggle, which is owned by Google and used by many researchers, was employed to design and train the model on the cloud. Kaggle provides NVidia P100 GPUs which proved to be quite useful for training the deep learning model of brain tumor segmentation. The designed model was trained online with one 9-hour session at a time. Then the

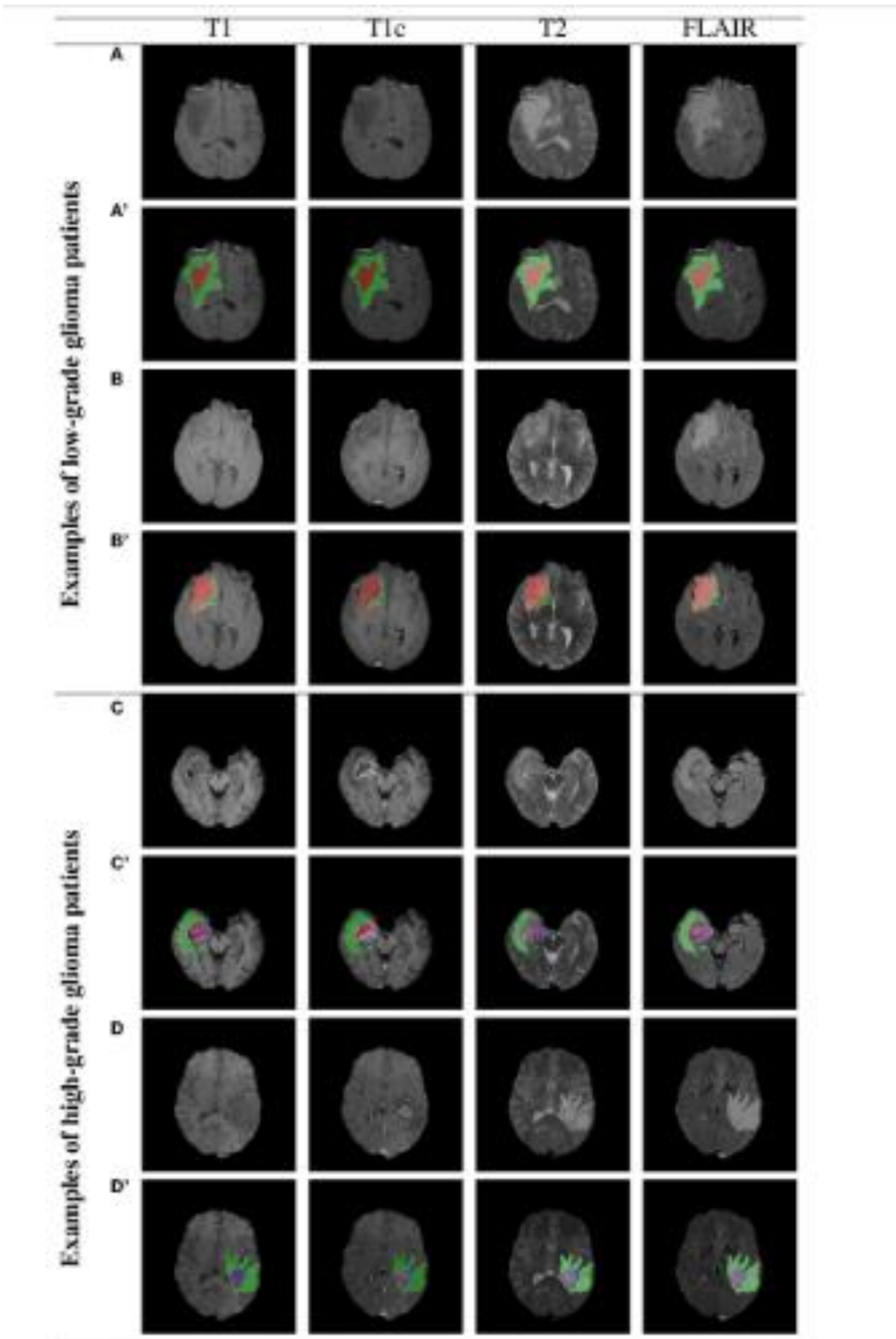


Figure 4.1: Two example low- and high-grade glioma patients from the BraTS 2018 dataset: red—GD-enhancing tumor (ET), green—peritumoral edema (ED), and blue—necrotic and non - enhancing tumor core (NCR/NET); (A–D) show original images, whereas (A'–D') present overlaid ground-truth masks [103].

model was saved and resumed from where it stopped as 9 hours of training were simply inadequate to achieve a high level of performance.

4.3 Performance Measures

The presented method is validated on various performance measures as given in Table 4.1. Dice similarity coefficient (DSC) defines the similarity quotient in 0 and 1 range using true positive (TP), false positive (FP) and false negative (FN). DSC has a vital role to measure the effectiveness of the proposed model. Also, Mean IOU is another important parameter that successfully depicts the performance of models in the task of segmentation.

Table 4.1: Evaluation metrics employed herein

Metric	Formula
Mean Intersection Over Union (IOU)	$Mean\ IOU = \frac{TP}{FP + 2TP + FN}$
Dice Similarity Coefficient (DSC)	$DSC = \frac{2TP}{FP + 2TP + FN}$
Sensitivity	$SE = \frac{TP}{TP + FN}$
Specificity	$SP = \frac{TN}{TN + FP}$
Accuracy	$ACC = \frac{TP + TN}{TP + TN + FP + FN}$
Precision	$PR = \frac{TP}{TP + FP}$

The evaluation metrics employed for the three model runs were loss, mean IOU, accuracy, precision, dice coefficient, specificity, and sensitivity/recall. These evaluation metrics are calculated using the formula shown in the table above.

4.4 Experiment Results

Several experiments were performed to obtain a model that can achieve good segmentation results. Coming up with a good model that generalizes well was indeed the most difficult hurdle faced in implementing this thesis project. The goal here was to attain a high level of performance in segmenting Gliomas making use of a 3D model developed within this project. After many trial and errors a 3D U-Net model consisting of eleven layers (five down sampling, five up sampling, and one middle or bottleneck layer) was constructed.

A major problem faced during the training was the issue of internet connection and power consistency. This is one of the major reasons that a relatively simple and powerful model was developed in this project. Simplicity was key here because as complexity increases the amount of GPU power required skyrockets and the training time increases alongside it. As a local GPU is not available and the model is trained on the cloud making use of Kaggle's GPU processor avoiding complex models was indeed necessary. Considering the limited internet connection and power issues in Ethiopia it took a while to train a high performance model.

At first, this model was run for about 100 epochs with batch size of one using the BraTS NIFTI format images directly. The result obtained was what can be considered "good" as a mean IOU higher than 70% was achieved. Then, in order to attain further improvement in performance, data augmentation was implemented as an additional feature and the model was run again. This resulted in significant improvement in the model's segmentation capacity. After this, the model was run with increased batch size of 2 to be able to observe if there are any improvements to be gained with increased batch size. Also, Gaussian noise with mean zero was applied to the dataset as an additional regularization technique. Further increase in batch size caused our training to be too slow hence it was not implemented herein.

On the other hand, in order to be able to train the model faster, the data format which was originally in the Neuroimaging Informatics Technology Initiative (NIfTI) format was converted to a simple NumPy array format. This change has resulted in a significant improvement in training time for the model. This is because NumPy arrays are simple to load for the data generator, which loads data at run time, resulting in significantly reduced training time. Also, images containing less than 1% of tumor mass were deleted as they do not contribute much to the betterment of the model. However, this model does not produce good results and needs further improvement which can be implemented in the future; hence, the results are not presented here.

Meanwhile, not all the evaluation metrics implemented herein were found to indicate the system’s performance accurately. In fact, only the mean IOU and DSC appear to follow the segmentation model’s performance nicely. This can be observed in Figure 4.2 where the model was run with batch size of 1 using NIFTI format images. For instance, around epoch 100 the model has an accuracy level above 99% while it can be seen in Figure 4.3 that the model does not do that well when actually performing segmentation on the test dataset. However, it is important to note that these metrics do indicate the model’s overall health. In the coming sub-sections, the specific results obtained when training all three models which are the result of employing different hyper-parameters are explained.

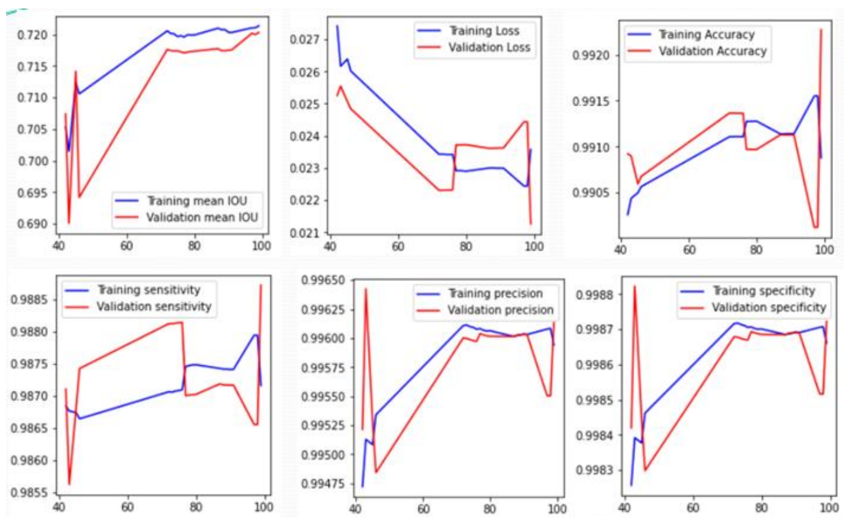


Figure 4.2: Evaluation metrics, epoch 40 to 100, for the 3D U-Net Model1 using NIFTI image format with batch size of 1.

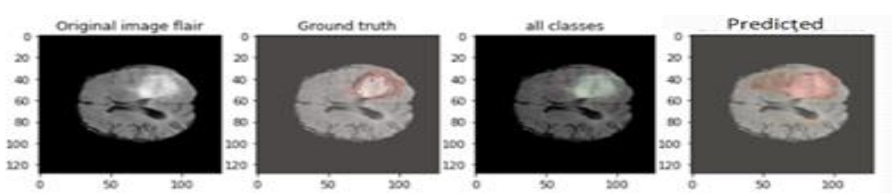


Figure 4.3: Segmentation output example that is not quite 100% successful

4.4.1 Experiment Results obtained with NIFTI image format

Employing NIFTI type MRI images three model trainings were performed in total and the following results were obtained.

a. **Model1:** 3D U-Net model with batch size of 1

This is the first functional model to be implemented. It was run for more than 100 epochs and achieves a mean IOU of 72% and a DSC of 0.52. It is considered a “good” performance level. The segmentation outputs were also good with some segmentation outputs having a spread of errors. Figure 4.2 depicts the values of the evaluation metrics between epochs 40 and 100 during training and validation in the graph. Some example segmentation outputs using the test dataset are also shown in Figure 4.4 below.

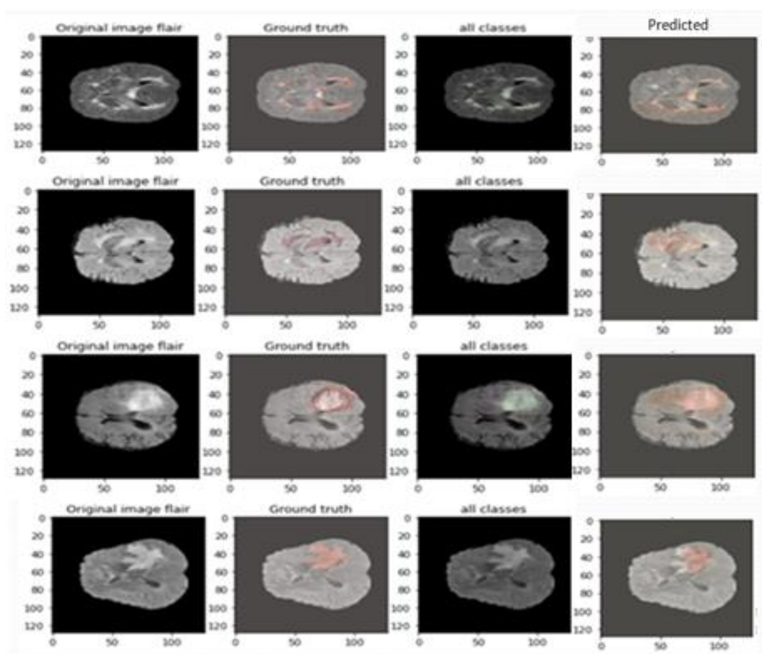


Figure 4.4: Segmentation output examples of Model1 (3D model with batch size of 1)

b. **Model2:** 3D U-Net model with batch size of 1 and data augmentation implementation

As it is mentioned earlier, in order to further improve the performance of the model, data augmentation is performed on the data from Kaggle at run time. This is achieved making use of the DataGenerator class. Specifically, the methods employed were: flipping, scaling, and rotating. This action actually increases the mean IOU value to 80.99% and DSC of 0.9093 on the test dataset which is comparable to models that are considered "state-of-the-art". This is indeed much better than the model developed in [105] which achieved 0.782 DSC. The model performs quite well during the data segmentation process as shown below (Figure 4.5). Also, the rest of the evaluation metrics implemented herein are presented in Figure 4.6.

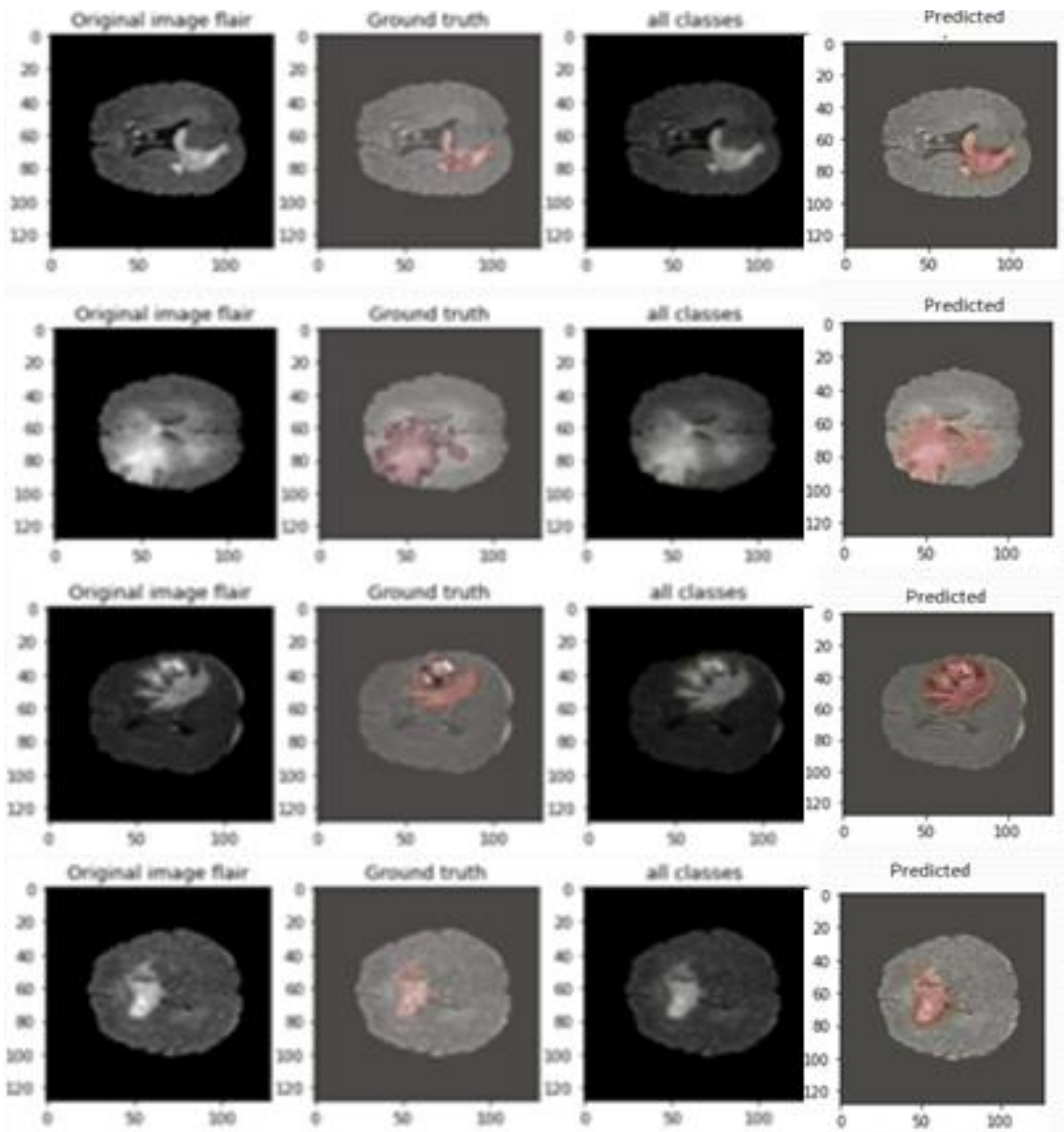


Figure 4.5: Segmentation outputs when using Model2: a U-Net model with data augmentation using batch size of 1.

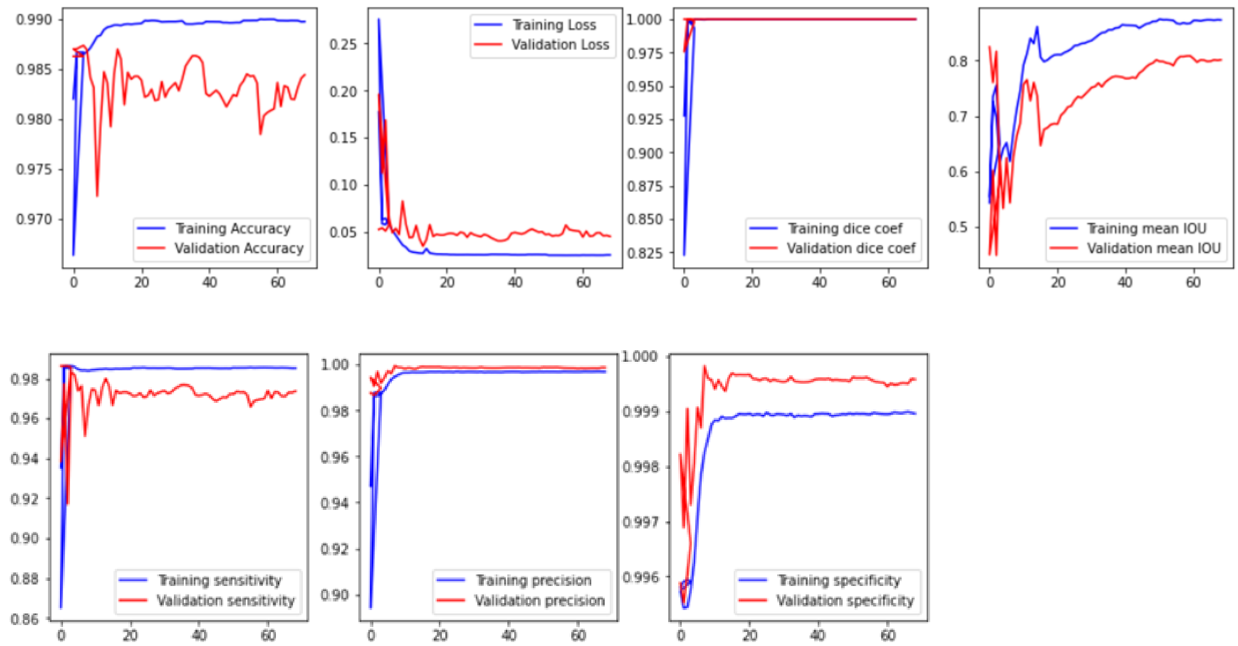


Figure 4.6: Evaluation metrics during training of Model2

c. **Model3:** 3D U-Net model with batch size of 2 and data augmentation implementation

This model is implemented to observe changes with increased batch size. Moreover, Gaussian noise is applied to the dataset which is considered as a regularization method. This model has achieved a mean IOU of 81.13% (Figure 4.7) and examples of segmentation plots can be seen in Figure 4.8 below. It can be seen that there is a little improvement in mean IOU as compared to Model2. This is the result of the Gaussian noise added to the data and the increased batch size. However, the speed of training was quite slow as two NIFTI format images, each consisting of four sequences, were loaded at a time during runtime. Implementing an even higher batch size seemed unpractical considering that the time taken for training with a batch size of 2 was too high.

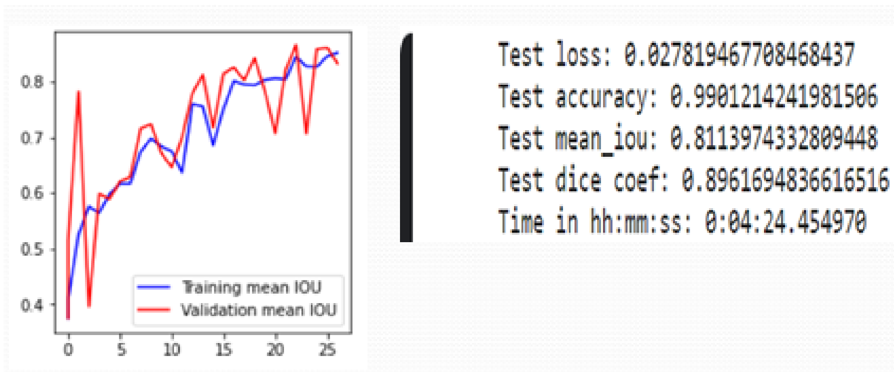


Figure 4.7: Model3 mean IOU values obtained by Model3 (a) Graphical representation of training and validation mean IOU (b) mean IOU found on test dataset

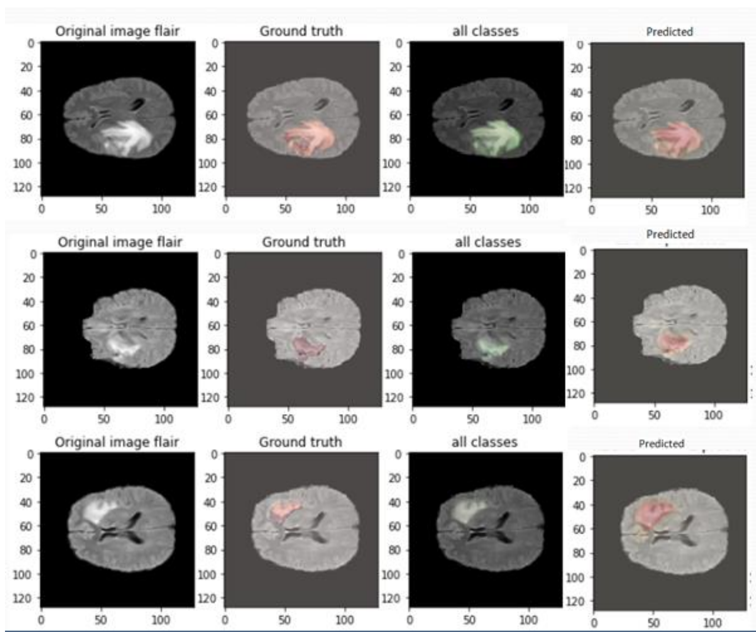


Figure 4.8: Segmentation outputs making use of Model3

4.5 Discussion

As mentioned earlier, the 3D U-Net model developed herein was run many times in order to attain a high-performance model for brain tumor segmentation. Finally, three functional models were developed with different performance levels which are presented herein. The mean IOU values along with the loss metric obtained from these three different models are presented in Table 4.2. As can be deduced from this table the mean IOU has improved from Model1 through

Model3. This shows the usefulness of the data augmentation techniques employed to improve the system's performance. The training time, however, is observed increasing from the first model through the third. This is because the amount of operation performed at run time is increased and for Model3 the increased batch size causes an increase in load time.

Table 4.2: Mean IOU values after training the model for a given number of epochs.

Model	Mean IOU	Epochs run	Hrs. per epoch
Model1	72.01%	100	1.5
Model2	80.99%	100	1.6
Model3	81.13%	50	2

Moreover, there were several other evaluation metrics implemented in this project. These are presented in Table 4.3 for all three models. It can be observed that a decent DSC value is attained by the models. DSC is employed by projects in the BraTS competition, therefore, attaining a good value of it is quite important to increase the significance of this project as compared to earlier models trained using the BraTS dataset. On the contrary, it is seen clearly that the rest of the evaluation metrics (accuracy, specificity, sensitivity, and precision) only indicate the overall health of the models but fail to indicate the model performance accurately.

Table 4.3: Brain tumor segmentation results making use of the three models (on test set)

Model 1	Mean IOU%	Acc%	SP%	SE%	DSC	PR%
Model1	72.01	99.04	96.71	98.70	0.5238	95.00
Model2	80.99	98.24	99.00	96.01	0.9093	99.00
Model3	81.13	99.00	98.34	98.50	0.8960	98.02

4.6 Comparison with Past Related Works

A DSC value of 0.9093 is attained by Model2 this is quite an improvement when compared to 0.7823 in [105]. Also, it performs better when compared to the BRATS 2018 winner model [24] which achieved an average of 0.821 in DSC. Moreover, [48], only performs a classification between LGG and HGG and does not perform any further classification as to where the tumor is located or how large it is. The models developed herein, however, employ a simple yet powerful architecture and do a good job of voxel-level segmentation. Moreover, even Model1, which is a crude version of the model, achieves a higher accuracy of 99.04% than in [48]. In [42], an out-of-the-box model is developed for biomedical image segmentations and quite impressive results are obtained. However, developing it took a very high level of expertise and it still achieves lower DSC than this project because it is more general purpose that can also be used for segmenting other biomedical images. The following table (Table 4.4) shows the DSC values of the papers discussed here along with that of Model2 and Model3.

Table 4.4: Brain tumor segmentation results of models developed herein and previous researches

Model/Method	**Dataset**	**Year**	**DSC**
[24]	BraTS	2018	0.821
[48]	BraTS	2018	0.990
[105]	BraTS	2020	0.782
[42]	BraTS	2020	0.853
Proposed (Model2)	BraTS	2020	0.909
Proposed (Model3)	BraTS	2020	0.896

All in all, the 3D U-Net model developed in this project achieves high segmentation performance while using a relatively simpler model that makes good use of convolution operations. Other models like [42] and [58] employ complex methods requiring much higher processing power. The model constructed herein has performed well in the segmentation of brain MRIs while maintaining simplicity and contributed to the realization of automated brain tumor segmentation within real-life hospitals since it achieves high performance. Also, the time taken for segmenting one subject's MRI employing Model2 is 3.44 seconds which is much less than earlier models [106], [107], [108] which took 84, 193, and 208 seconds per patient. Moreover, the model can be refined further in the future as the model is very convenient to upgrade.

Chapter 5
Conclusion and Future Works

5.1 Conclusion

In this work, a fully functional semantic segmentation model for brain tumor segmentation using multimodal 3D MRIs which achieved very high performance is built. The underlying architecture of the network implemented herein for the purpose of segmentation of gliomas is 3D-UNet architecture. A 3D data generator was also built to feed data to this model at runtime. This is because not all the data can be loaded to memory at a time. Furthermore, data augmentation is performed on the fly by this data generator class. The logger class, on the other hand, is made for saving the model at the end of every epoch in a very convenient to manage way while the model checkpoint class is crucial for the purpose of stopping the execution of the code and then resuming from where it stopped. This is necessary as it needs many an hour to complete the model training and it cannot be completed in one go. Therefore, picking up from where the execution stopped is necessary instead of restarting from zero every time the training is restarted.

While experimenting with the network's architectures, several alternative approaches were tried out. For instance, first, a batch size of 1 was run on the 7iCore laptop; this was however still beyond the capacity of the computer. Then this same model with a batch size of 1 was trained (Model1) on Kaggle GPU and was found to be functional with a "good" result. Kaggle's GPU during the time of implementation of this project is the Nvidia P100 with 16GB memory space. Intending to increase the model's performance, it was experimented with several data augmentation techniques which provided significant improvement in the model performance (Model2). On the other hand, increasing the network depth further did not improve the performance, but increasing the network width (the number of features/filters) consistently improved the results. Also, there was an attempt to train the developed model with a batch size of 2 (Model3). The result was pretty good even though a larger amount of time was required to train it. Therefore, training with an even higher batch size seemed unpractical. The mean IOU results obtained were 72.01%, 80.99%, and 81.13% for Model1, Model2, and Model3 respectively while their respective DSC values attained were 0.52, 0.90, and 0.89. As visible from the segmentation results, the network performs well and achieves quality results; it is able to generalize well between the training set and test set with results comparable to state-of-the-art methods.

5.2 Future Works

Going forward there are several ways that can possibly improve the effectiveness of the current model. One example of this is advanced data augmentation that can be used to regularize the model. For instance, generative adversarial networks (GANs) can be employed to come up with new synthetic data. Furthermore, design improvements can be tried for the 3D model itself in order to achieve higher performance level. Training separate models for the HGG and LGG is one variation that can be tested. Moreover, preprocessing techniques can be considered in order to attain less training time which can be the foundation to try out many improvement techniques for the model at a much faster rate.

References

[1] F. B. Mesfin and M. A. Al-Dhahir, “Gliomas,” *StatPearls*, Jun. 2022.

[2] J. O. S. H. Cleary and A. R. Guimarães, “Magnetic Resonance Imaging,” *Pathobiol. Hum. Dis. A Dyn. Encycl. Dis. Mech.*, pp. 3987–4004, Jan. 2014, doi: 10.1016/B978-0-12-386456-7.07609-7.

[3] H. Peni Agustin Tjahyaningtijas, “Brain Tumor Image Segmentation in MRI Image,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 336, no. 1, 2018, doi: 10.1088/1757-899X/336/1/012012.

[4] D. Shen, G. Wu, and H.-I. Suk, “Deep Learning in Medical Image Analysis,” 2017, doi: 10.1146/annurev-bioeng-071516.

[5] L. Liu *et al.*, “Deep Learning for Generic Object Detection: A Survey,” *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 261–318, 2020, doi: 10.1007/s11263-019-01247-4.

[6] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, no. Mlm, pp. 4171–4186, 2019.

[7] B. H. Menze *et al.*, “The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS),” *IEEE Trans. Med. Imaging*, vol. 34, no. 10, 2015, doi: 10.1109/TMI.2014.2377694.

[8] S. Bakas *et al.*, “Data Descriptor: Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features Background & Summary,” 2017, doi: 10.1038/sdata.2017.117.

[9] S. Bakas *et al.*, “Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge,” 2019.

[10] “BraTS2020 Dataset (Training + Validation) | Kaggle.” [Online]. Available: https://www.kaggle.com/datasets/awsaf49/brats20-dataset-training-validation. [Accessed: 07-Oct-2022].

[11] M. Ghafoorian *et al.*, “Deep multi-scale location-aware 3D convolutional neural networks for automated detection of lacunes of presumed vascular origin,” *NeuroImage Clin.*, vol. 14, pp. 391–399, 2017, doi: 10.1016/J.NICL.2017.01.033.

[12] Q. Dou *et al.*, “Automatic Detection of Cerebral Microbleeds from MR Images via 3D Convolutional Neural Networks,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1182–1195, 2016, doi: 10.1109/TMI.2016.2528129.

[13] Q. Dou *et al.*, “Automatic cerebral microbleeds detection from MR images via Independent Subspace Analysis based hierarchical features,” *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, vol. 2015-Novem, pp. 7933–7936, 2015, doi:

10.1109/EMBC.2015.7320232.

[14] "Glioma - Symptoms and causes - Mayo Clinic." [Online]. Available: https://www.mayoclinic.org/diseases-conditions/glioma/symptoms-causes/syc-20350251. [Accessed: 07-Oct-2022].

[15] Z. Liu *et al.*, "Deep Learning Based Brain Tumor Segmentation: A Survey," vol. 14, no. 8, pp. 1–21, 2020.

[16] "Gliomas | Johns Hopkins Medicine." [Online]. Available: https://www.hopkinsmedicine.org/health/conditions-and-diseases/gliomas. [Accessed: 07-Oct-2022].

[17] "Human Brain: facts and information." [Online]. Available: https://www.nationalgeographic.com/science/article/brain-2. [Accessed: 07-Oct-2022].

[18] N. Patel and O. Kirmi, "Anatomy and imaging of the normal meninges," *Semin. Ultrasound. CT. MR*, vol. 30, no. 6, pp. 559–564, Dec. 2009, doi: 10.1053/J.SULT.2009.08.006.

[19] C. I. Hill, C. S. Nixon, J. L. Ruehmeier, and L. M. Wolf, "Brain Tumors," *Phys. Ther.*, vol. 82, no. 5, pp. 496–502, May 2002, doi: 10.1093/ptj/82.5.496.

[20] M. G. Castro *et al.*, "Current and future strategies for the treatment of malignant brain tumors," *Pharmacol. Ther.*, vol. 98, no. 1, pp. 71–108, Apr. 2003, doi: 10.1016/S0163-7258(03)00014-7.

[21] American assiociation of neurological surgeons, "Brain Tumors - Classifications, Symptoms, Diagnosis and Treatments." [Online]. Available: https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Brain-Tumors. [Accessed: 08-Oct-2022].

[22] M. L. Bondy *et al.*, "Brain Tumor Epidemiology: Consensus From the Brain Tumor Epidemiology Consortium On behalf of the Brain Tumor Epidemiology Consortium," vol. 113, no. S7, 2008, doi: 10.1002/cncr.23741.

[23] C.-H. Lee, K.-W. Jung, H. Yoo, S. Park, and S. H. Lee, "Epidemiology of Primary Brain and Central Nervous System Tumors in Korea Clinical Article," *J. Korean Neurosurg. Soc.*, vol. 48, no. 2, 2010, doi: 10.3340/jkns.2010.48.2.145.

[24] A. Myronenko, "3D MRI brain tumor segmentation using autoencoder regularization," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11384 LNCS, pp. 311–320, 2019, doi: 10.1007/978-3-030-11726-9_28.

[25] C. Kerrigan, "Introduction To Ai," *Artif. Intell.*, pp. 30–36, 2022, doi: 10.4337/9781800371729.00013.

[26] V. Kaul, S. Enslin, and S. A. Gross, "History of artificial intelligence in medicine," *Gastrointest. Endosc.*, vol. 92, no. 4, pp. 807–812, 2020, doi: 10.1016/j.gie.2020.06.040.

[27] W. Horn, "AI in medicine on its way from knowledge-intensive to data-intensive systems," *Artif. Intell. Med.*, vol. 23, no. 1, pp. 5–12, 2001.

[28] A. Géron, *Hands-On Machine Learning with Scikit-Learn*. O'Reilly Media, Inc., 2017.

[29] L. A. Review, M. A. Khan, K. M. Khan, and S. H. Almotiri, "brain sciences Brain Tumor Analysis Empowered with Deep Future Challenges," *Brain Sci.*, pp. 1–33, 2020.

[30] H. Wang and B. Raj, "On the Origin of Deep Learning," pp. 1–72, 2017.

[31] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep Learning With TensorFlow: A Review," *J. Educ. Behav. Stat.*, vol. 45, no. 2, pp. 227–248, 2020, doi: 10.3102/1076998619872761.

[32] A. A. Taha and A. Hanbury, "Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool," *BMC Med. Imaging*, vol. 15, no. 1, Aug. 2015, doi: 10.1186/S12880-015-0068-X.

[33] Y. LeCun *et al.*, "Backpropagation applied to digit recognition," *Neural computation*, vol. 1, no. 4. AT & T Laboratories, pp. 541–551, 1989.

[34] T. F. Gonzalez, "Handbook of approximation algorithms and metaheuristics," *Handb. Approx. Algorithms Metaheuristics*, pp. 1–1432, 2007, doi: 10.1201/9781420010749.

[35] S. Maqsood, R. Damaševičius, and R. Maskeliūnas, "Multi-Modal Brain Tumor Detection Using Deep Neural Network and Multiclass SVM," *Medicina (Kaunas).*, vol. 58, no. 8, 2022, doi: 10.3390/medicina58081090.

[36] S. Pereira, A. Pinto, V. Alves, and C. A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1240–1251, 2016, doi: 10.1109/TMI.2016.2538465.

[37] D. Zikic, Y. Ioannou, M. Brown, and A. Criminisi, "Segmentation of Brain Tumor Tissues with Convolutional Neural Networks," *Proc. MICCAI-BRATS*, no. September 2014, pp. 36–39, 2014.

[38] T. Brosch, L. Y. W. Tang, Y. Yoo, D. K. B. Li, A. Traboulsee, and R. Tam, "Deep 3D Convolutional Encoder Networks With Shortcuts for Multiscale Feature Integration Applied to Multiple Sclerosis Lesion Segmentation," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1229–1239, 2016, doi: 10.1109/TMI.2016.2528821.

[39] P. Moeskops, M. A. Viergever, A. M. Mendrik, L. S. De Vries, M. J. N. L. Benders, and I. Isgum, "Automatic Segmentation of MR Brain Images with a Convolutional Neural Network," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1252–1261, 2016, doi: 10.1109/TMI.2016.2548501.

[40] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017, doi: 10.1109/TPAMI.2016.2572683.

[41] O. Ronneberger, P. Fischer, and T. Brox, "UNet: Convolutional Networks for Biomedical

Image Segmentation," *IEEE Access*, vol. 9, pp. 16591–16603, 2015, doi: 10.1109/ACCESS.2021.3053408.

[42] F. Isensee, P. F. Jäger, P. M. Full, P. Vollmuth, and K. H. Maier-Hein, "nnU-Net for Brain Tumor Segmentation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12659 LNCS, pp. 118–132, 2021, doi: 10.1007/978-3-030-72087-2_11.

[43] Q. Yang, Z. He, F. Ge, and Y. Zhang, "Sequence-to-sequence prediction of personal computer software by recurrent neural network," *Proc. Int. Jt. Conf. Neural Networks*, vol. 2017-May, pp. 934–940, 2017, doi: 10.1109/IJCNN.2017.7965952.

[44] E. Carvalho *et al.*, "Exploiting the use of recurrent neural networks for driver behavior profiling," *Proc. Int. Jt. Conf. Neural Networks*, vol. 2017-May, pp. 3016–3021, 2017, doi: 10.1109/IJCNN.2017.7966230.

[45] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Phys. D Nonlinear Phenom.*, vol. 404, no. March, pp. 1–43, 2020, doi: 10.1016/j.physd.2019.132306.

[46] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," *Adv. Neural Inf. Process. Syst. 21 - Proc. 2008 Conf.*, pp. 545–552, 2009.

[47] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with LSTM recurrent neural networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 3547–3555, 2015, doi: 10.1109/CVPR.2015.7298977.

[48] J. Amin, M. Sharif, M. Raza, T. Saba, R. Sial, and S. A. Shad, "Brain tumor detection: a long short-term memory (LSTM)-based learning model," *Neural Comput. Appl.*, vol. 32, no. 20, pp. 15965–15973, 2020, doi: 10.1007/s00521-019-04650-7.

[49] R. Dey and F. M. Salemt, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," *Midwest Symp. Circuits Syst.*, vol. 2017-Augus, no. 2, pp. 1597–1600, 2017, doi: 10.1109/MWSCAS.2017.8053243.

[50] S. Andermatt, S. Pezold, and P. C. Cattin, "Automated segmentation of multiple sclerosis lesions using multi-dimensional gated recurrent units," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10670 LNCS, pp. 31–42, 2018, doi: 10.1007/978-3-319-75238-9_3.

[51] M. Rezaei *et al.*, "A conditional adversarial network for semantic segmentation of brain tumor," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10670 LNCS, pp. 241–252, 2018, doi: 10.1007/978-3-319-75238-9_21.

[52] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," *SN Comput. Sci.*, vol. 2, no. 6, pp. 1–20, 2021, doi: 10.1007/s42979-021-00815-1.

[53] Q. Liu, J. Xu, R. Jiang, and W. H. Wong, “Density estimation using deep generative neural networks,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 118, no. 15, 2021, doi: 10.1073/pnas.2101344118.

[54] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, “Semantic Segmentation using Adversarial Networks,” 2016.

[55] K. Vaidhya, S. Thirunavukkarasu, V. Alex, and G. Krishnamurthi, “Multi-modal brain tumor segmentation using stacked denoising autoencoders,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9556, pp. 181–194, 2016, doi: 10.1007/978-3-319-30858-6_16.

[56] P. Y. Kao, T. Ngo, A. Zhang, J. W. Chen, and B. S. Manjunath, “Brain tumor segmentation and tractographic feature extraction from structural MR images for overall survival prediction,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11384 LNCS, pp. 128–141, 2019, doi: 10.1007/978-3-030-11726-9_12.

[57] M. Cabezas *et al.*, “Survival prediction using ensemble tumor segmentation and transfer learning,” pp. 1–9, 2018.

[58] L. Chen, P. Bentley, and D. Rueckert, “Fully automatic acute ischemic lesion segmentation in DWI using convolutional neural networks,” *NeuroImage Clin.*, vol. 15, no. May, pp. 633–643, 2017, doi: 10.1016/j.nicl.2017.06.016.

[59] J. Amin, M. A. Anjum, M. Sharif, S. Jabeen, S. Kadry, and P. Moreno Ger, “A New Model for Brain Tumor Detection Using Ensemble Transfer Learning and Quantum Variational Classifier,” *Comput. Intell. Neurosci.*, vol. 2022, 2022, doi: 10.1155/2022/3236305.

[60] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1520–1528, 2015, doi: 10.1109/ICCV.2015.178.

[61] K. Kamnitsas *et al.*, “Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation,” *Med. Image Anal.*, vol. 36, pp. 61–78, 2017, doi: 10.1016/j.media.2016.10.004.

[62] M. Seyedhosseini, M. Sajjadi, and T. Tasdizen, “Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 2168–2175, 2013, doi: 10.1109/ICCV.2013.269.

[63] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, “Hypercolumns for object segmentation and fine-grained localization,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 447–456, 2015, doi: 10.1109/CVPR.2015.7298642.

[64] F. Milletari *et al.*, “Hough-CNN: Deep learning for segmentation of deep brain regions in MRI and ultrasound,” *Comput. Vis. Image Underst.*, vol. 164, pp. 92–102, 2017, doi:

10.1016/j.cviu.2017.04.002.

[65] J. Kleesiek *et al.*, "Deep MRI brain extraction: A 3D convolutional neural network for skull stripping," *Neuroimage*, vol. 129, pp. 460–469, 2016, doi: 10.1016/j.neuroimage.2016.01.024.

[66] W. Weng and X. Zhu, "INet: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, pp. 16591–16603, 2021, doi: 10.1109/ACCESS.2021.3053408.

[67] C. E. Cardenas, J. Yang, B. M. Anderson, L. E. Court, and K. B. Brock, "Advances in Auto-Segmentation," *Semin. Radiat. Oncol.*, vol. 29, no. 3, pp. 185–197, Jul. 2019, doi: 10.1016/J.SEMRADONC.2019.02.001.

[68] A. E. Morabito *et al.*, "Comparison of Three 3D Segmentation Software Tools for Hip Surgical Planning," 2022, doi: 10.3390/s22145242.

[69] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8150 LNCS, no. PART 2, pp. 246–253, 2013, doi: 10.1007/978-3-642-40763-5_31.

[70] E. S. Biratu, F. Schwenker, Y. M. Ayano, and T. G. Debelee, "A survey of brain tumor segmentation and classification algorithms," *J. Imaging*, vol. 7, no. 9, Sep. 2021, doi: 10.3390/JIMAGING7090179.

[71] M. Fogarasi, J. C. Coburn, and B. Ripley, "Algorithms used in medical image segmentation for 3D printing and how to understand and quantify their performance," *3D Print. Med.*, vol. 8, no. 1, pp. 1–15, 2022, doi: 10.1186/s41205-022-00145-9.

[72] Y. He *et al.*, "Deep Learning based 3D Segmentation: A Survey," *Proc.*, vol. 1, no. 1, pp. 1–36, 2021.

[73] C. Wang, "3D Deeply-Supervised U-Net Based Whole Heart Segmentation," vol. 3, pp. 242–249, 2018, doi: 10.1007/978-3-319-75541-0.

[74] H. J. Bae *et al.*, "Fully automated 3D segmentation and separation of multiple cervical vertebrae in CT images using a 2D convolutional neural network," *Comput. Methods Programs Biomed.*, vol. 184, p. 105119, 2020, doi: 10.1016/j.cmpb.2019.105119.

[75] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-net: Learning dense volumetric segmentation from sparse annotation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9901 LNCS, no. June 2016, pp. 424–432, 2016, doi: 10.1007/978-3-319-46723-8_49.

[76] P. Sandur, C. Naveena, V. N. M. Aradhya, and Nagasundara K. B., "Segmentation of Brain Tumor Tissues in HGG and LGG MR Images Using 3D U-net Convolutional Neural Network," *Int. J. Nat. Comput. Res.*, vol. 7, no. 2, pp. 18–30, 2018, doi: 10.4018/ijncr.2018040102.

[77] A. Nagpal and G. Gabrani, “Python for Data Analytics, Scientific and Technical Applications,” *Proc. - 2019 Amity Int. Conf. Artif. Intell. AICAI 2019*, pp. 140–145, 2019, doi: 10.1109/AICAI.2019.8701341.

[78] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2016.

[79] P. Goldsborough, “A Tour of TensorFlow,” 2016.

[80] X. Yi *et al.*, “Fast training of deep learning models over multiple GPUs,” *Middlew. 2020 - Proc. 2020 21st Int. Middlew. Conf.*, pp. 105–118, 2020, doi: 10.1145/3423211.3425675.

[81] G. Zaccone, “Getting started with tensorflow : get up and running with the latest numerical computing library by Google and dive deeper into your data!,” 2016.

[82] K. Ramasubramanian and A. Singh, “Deep Learning Using Keras and TensorFlow,” *Mach. Learn. Using R*, pp. 667–688, 2019, doi: 10.1007/978-1-4842-4215-5_11.

[83] N. K. Manaswi, “Understanding and Working with Keras,” *Deep Learn. with Appl. Using Python*, pp. 31–43, 2018, doi: 10.1007/978-1-4842-3516-4_2.

[84] Z. Wang, K. Liu, J. Li, Y. Zhu, and Y. Zhang, “Various Frameworks and Libraries of Machine Learning and Deep Learning: A Survey,” *Arch. Comput. Methods Eng. 2019*, pp. 1–24, Feb. 2019, doi: 10.1007/S11831-018-09312-W.

[85] N. Ketkar, “Introduction to Keras,” *Deep Learn. with Python*, pp. 97–111, 2017, doi: 10.1007/978-1-4842-2766-4_7.

[86] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, 2015.

[87] O. Travis, *Guide to NumPy*. Trelgol Publishing USA, 2007.

[88] S. Van der Walt and M. Aivazis, “The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering,” *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011.

[89] A. Abraham *et al.*, “Machine learning for neuroimaging with scikit-learn,” *Front. Neuroinform.*, vol. 8, no. FEB, pp. 1–10, 2014, doi: 10.3389/fninf.2014.00014.

[90] M. Brett *et al.*, “nipy/nibabel: 2.4.1,” May 2019, doi: 10.5281/ZENODO.3233118.

[91] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, and X. Gao, “Deep learning in bioinformatics: Introduction, application, and perspective in the big data era,” *Methods*, vol. 166, no. 2019, pp. 4–21, 2019, doi: 10.1016/j.ymeth.2019.04.008.

[92] A. Češka, “A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons,” *Folia Geobot. Phytotaxon.*, vol. 1, no. 3, pp. 93–100, 1966, doi: 10.1007/BF02852438.

[93] L. R. Dice, “Measures of the Amount of Ecologic Association Between Species,” *Ecology*, vol. 26, no. 3, pp. 297–302, Jul. 1945, doi: 10.2307/1932409.

[94] A. Carass *et al.*, “Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis,” *Sci. Rep.*, vol. 10, no. 1, pp. 1–19, 2020, doi: 10.1038/s41598-020-64803-w.

[95] J. Looman and J. B. Campbell, “Adaptation of Sorensen’s K (1948) for Estimating Unit Affinities in Prairie Vegetation,” *Ecology*, vol. 41, no. 3, pp. 409–416, Jul. 1960, doi: 10.2307/1933315.

[96] O. A. Adegbola, P. O. Idowu, T. L. Adebisi, J. Adeleke, D. O. Babajide, and J. A. Ojo, “Brain Tumour Segmentation Using JeisloNet - a Unet Architecture,” vol. 5, no. 9, pp. 45–50, 2021.

[97] H. J. Drescher, “Advances and opportunities in image analysis of bacterial cells and communities,” *AJR. Am. J. Roentgenol.*, vol. 186, no. 2, pp. 227–236, 2017.

[98] Z. Wang, E. Wang, and Y. Zhu, *Image segmentation evaluation: a survey of methods*, vol. 53, no. 8. Springer Netherlands, 2020.

[99] A. Y. Wang *et al.*, “What Makes aWell-Documented Notebook? A Case Study of Data Scientists’ Documentation Practices in Kaggle,” *Conf. Hum. Factors Comput. Syst. - Proc.*, May 2021, doi: 10.1145/3411763.3451617.

[100] L. Prechelt, “Early Stopping - But When?,” pp. 55–69, 1998, doi: 10.1007/3-540-49430-8_3.

[101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[102] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” *Microsoft Res.*, no. Icdar, pp. 1–6, 2003.

[103] J. Nalepa, M. Marcinkiewicz, and M. Kawulok, “Data Augmentation for Brain-Tumor Segmentation: A Review,” *Front. Comput. Neurosci.*, vol. 13, no. December, pp. 1–18, 2019, doi: 10.3389/fncom.2019.00083.

[104] J. Nalepa *et al.*, “Data Augmentation via Image Registration,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 2019-Septe, pp. 4250–4254, 2019, doi: 10.1109/ICIP.2019.8803423.

[105] N. Sohail, S. M. Anwar, F. Majeed, C. Sanin, and E. Szczerbicki, “Smart Approach for Glioma Segmentation in Magnetic Resonance Imaging using Modified Convolutional Network Architecture (U-NET),” *Cybern. Syst.*, vol. 52, no. 5, pp. 445–460, 2021, doi: 10.1080/01969722.2020.1871231.

[106] R. Ranjbarzadeh, A. B. Kasgari, S. J. Ghoushchi, S. Anari, M. Naseri, and M. Bendechache, “Brain tumor segmentation based on deep learning and an attention mechanism using MRI multi-modalities brain images,” *Sci. Reports* |, vol. 11, no. 10930,

p. 17, 2021, doi: 10.1038/s41598-021-90428-8.

[107] D. Zhang *et al.*, “Exploring Task Structure for Brain Tumor Segmentation from Multi-Modality MR Images,” *IEEE Trans. Image Process.*, vol. 29, 2020, doi: 10.1109/TIP.2020.3023609.

[108] D. Zhang, G. Huang, Q. Zhang, J. Han, J. Han, and Y. Yu, “Cross-modality deep feature learning for brain tumor segmentation,” *Pattern Recognit.*, vol. 110, Feb. 2021, doi: 10.1016/J.PATCOG.2020.107562.