

OPTIMIZED U-NET ARCHITECTURE FOR BRAIN TUMOR SEGMENTATION

*Minor project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

LOGESHWARAN K S	(20UECS0530)	(15366)
SATWIKA B	(20UECS0852)	(15368)
FAZEL KHAN P	(20UECS0729)	(17589)

*Under the guidance of
Mr. S. SARAN RAJ , M.E.,
Assistant Professor*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2023

OPTIMIZED U-NET ARCHITECTURE FOR BRAIN TUMOR SEGMENTATION

*Minor project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

LOGESHWARAN K S	(20UECS0530)	(15366)
SATWIKA B	(20UECS0852)	(15368)
FAZEL KHAN P	(20UECS0729)	(17589)

*Under the guidance of
Mr. S. SARAN RAJ, M.E.,
Assistant Professor*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2023

CERTIFICATE

It is certified that the work contained in the project report titled “OPTIMIZED U-NET ARCHITECTURE FOR BRAIN TUMOR SEGMENTATION” by “LOGESHWARAN K S (20UECS0530), SATWIKA B (20UECS0852), FAZEL KHAN P (20UECS0729)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Mr. S. Saran Raj

Assistant Professor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May 2023

Signature of Head of the Department

Dr. M. S. Murali Dhar

Associate Professor & HOD

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2023

Signature of the Dean

Dr. V. Srinivasa Rao

Professor & Dean

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2023

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any source in our submission. We understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

LOGESHWARAN K S

Date: / /

SATWIKA B

Date: / /

FAZEL KHAN P

Date: / /

APPROVAL SHEET

This project report entitled OPTIMIZED U-NET ARCHITECTURE FOR BRAIN TUMOR SEGMENTATION by LOGESHWARAN K S (20UECS0530), SATWIKHA B (20UECS0852), FAZEL KHAN P (20UECS0729) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Mr. S. SARAN RAJ, M.E.,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.**, for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr. M. S. MURALI DHAR, M.E., Ph.D.**, for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Internal Supervisor Mr. S. SARAN RAJ, M.E.**, for his cordial support, valuable information and guidance, he helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E.**, for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

LOGESHWARAN K S	(20UECS0530)
SATWIKA B	(20UECS0852)
FAZEL KHAN P	(20UECS0729)

ABSTRACT

One of the main causes of death in the world is brain tumors, and early detection and categorization are crucial for appropriate diagnosis and treatment planning. Recent years have seen the finest outcomes in medicine, particularly in the segmentation of brain tumors, thanks to deep learning-based learning. Due to its success in medical picture segmentation tasks, U-Net architecture, one of the deep learning approaches, has received a lot of attention. On the BraTS2020 dataset, which contains multimodal Magnetic Resonance Imaging (MRI) scans of the brain, we suggest a brain segmentation method using U-Net architecture. Preprocessing, data augmentation, U-Net model training, and testing are all part of the strategy. Several factors were considered when determining whether the proposed method was effective. The Dice Similarity Coefficient (DSC), sensitivity, and specificity are the various metrics used. The proposed method achieves an accuracy of 0.99 % in the BraTS2020 dataset, according to experimental findings, demonstrating the vast potential of U-Net for brain segmentation. This study advances the development of precise and efficient techniques for the diagnosis and planning of brain tumor treatments, which will benefit medical professionals.

Keywords: Brain Tumor, BraTS2020 dataset, U-Net architecture, Magnetic Resonance Imaging, Dice Similarity Coefficient.

LIST OF FIGURES

4.1	General Architecture of Proposed System	12
4.2	Data Flow Diagram	13
4.3	Data Extraction	14
4.4	Model Building	15
4.5	Model Training	16
4.6	Model Evaluation	17
5.1	Unit Testing Result	21
6.1	Output 1 - Model Training	25
6.2	Output 2 - Model Prediction	26
9.1	Poster Presentation	37

LIST OF TABLES

4.1	Description of the BRATS 2020 Dataset	18
6.1	Training and Validation Performance of Proposed System	22
6.2	Comparison of Existing CNN Model and Proposed System	23

LIST OF ACRONYMS AND ABBREVIATIONS

ANN	Artificial Neural Networks
BRATS	Brain Tumor Segmentation
CNN	Convolutional Neural Network
CT	Computed Tomography
DL	Deep Learning
DSC	Dice Similarity coefficient
ECI	International Electrotechnical Commission
FCN	Fully Convolutional Network
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
HGG	High-Grade Glioblastoma
ISO	International Organization for Standardization
KNN	K-Nearest Neighbors
LGG	Low-Grade Glioblastoma
ML	Machine Learning
RF	Radio Frequency
SVM	Support Vector Machine
PET	Positron Emission Tomography
VGG	Visual Geometry Group

TABLE OF CONTENTS

	Page.No
ABSTRACT	v
LIST OF FIGURES	vi
LIST OF TABLES	vii
LIST OF ACRONYMS AND ABBREVIATIONS	viii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Aim of the Project	2
1.3 Project Domain	2
1.4 Scope of the Project	2
2 LITERATURE REVIEW	3
3 PROJECT DESCRIPTION	8
3.1 Existing System	8
3.2 Proposed System	8
3.3 Feasibility Study	9
3.3.1 Economic Feasibility	9
3.3.2 Technical Feasibility	10
3.3.3 Social Feasibility	10
3.4 System Specification	11
3.4.1 Hardware Specification	11
3.4.2 Software Specification	11
3.4.3 Standards and Policies	11
4 METHODOLOGY	12
4.1 General Architecture	12
4.2 Design Phase	13
4.2.1 Data Flow Diagram	13

4.3	Module Description	14
4.3.1	Data Extraction	14
4.3.2	Model Building	15
4.3.3	Model Training	16
4.3.4	Model Evaluation	16
4.4	Steps to execute/run/implement the project	17
4.4.1	Accessing Dataset	17
4.4.2	Extracting Data	18
4.4.3	Creating a Model	18
4.4.4	Compiling and Model Fitting	18
4.4.5	Determining Efficiency	18
5	IMPLEMENTATION AND TESTING	19
5.1	Input and Output	19
5.1.1	Input Design	19
5.1.2	Output Design	19
5.2	Testing	20
5.3	Types of Testing	20
5.3.1	Unit Testing	20
5.3.2	Test Result	21
6	RESULTS AND DISCUSSIONS	22
6.1	Efficiency of the Proposed System	22
6.2	Comparison of Existing and Proposed System	22
6.3	Sample Code	23
7	CONCLUSION AND FUTURE ENHANCEMENTS	27
7.1	Conclusion	27
7.2	Future Enhancements	27
8	PLAGIARISM REPORT	29
9	SOURCE CODE & POSTER PRESENTATION	30
9.1	Source Code	30
9.2	Poster Presentation	37
	References	38

Chapter 1

INTRODUCTION

1.1 Introduction

With the development of machine learning and deep learning algorithms for the segmentation, identification, and prediction of patient survival from tumors or other diseases during the past several decades, the area of medical imaging has experienced a revolution. Additionally, it aids in the early detection of malignant brain tumors by medical professionals, improving prognosis. It is believed that glial cells give birth to gliomas, which most frequently affect adults and penetrate nearby tissues. Glioma is a major kind of brain tumor. There are two further glioma subtypes: High-Grade Glioblastoma (HGG) and Low-Grade Glioblastoma (LGG). While segmentation of 2D modalities is laborious, with variations, and prone to errors, radiologists manually review MRI modalities to produce quantitative information.

Medical image analysis requires the detection and delineation of various tumor locations from Magnetic Resonance (MR) images, which is where the work of brain tumor segmentation comes in. For the clinical diagnosis, treatment planning, and patient monitoring of those with brain tumors, accurate segmentation is essential. Due to its capacity to recognize spatial connections between voxels and learn complicated characteristics, deep learning approaches like the U-Net architecture have been frequently used for this job. An annual contest called the BraTS (Brain Tumour Segmentation) challenge offers a baseline dataset for testing brain tumor segmentation techniques. The BraTS 2020 dataset includes ground truth segmentation labels for four tumor areas together with multi-modal MR images comprising T1-weighted, T1-weighted contrast-enhanced, T2-weighted, and FLuid-Attenuated Inversion Recovery (FLAIR) images.

In this study, we build a deep learning model for brain tumor segmentation using the BraTS 2020 dataset using the U-Net architecture. The dataset is augmented to make it larger and the photos are preprocessed to normalize their intensities. To evaluate the effectiveness of our model, we employ a variety of assessment criteria, including accuracy, dice coefficient, precision, sensitivity, and specificity. In order

to confirm that our model is generalizable, we additionally cross-validate it. Our results show that the U-Net model performs well in terms of dice coefficient values and accuracy, indicating that it is efficient at segmenting brain tumors. Additionally, the model's precision, sensitivity, and specificity values demonstrate its capability to precisely identify the various tumor regions. In general, the segmentation of brain tumors is a valuable method presented by our study.

1.2 Aim of the Project

The aim of brain tumor segmentation using U-Net architecture is to accurately describe the brain tumor site by creating mask to MRI images and to find brain tumors with high accuracy using an optimized U-net architecture. It can even help people at an earlier stage by detecting the location and extension of the tumor regions.

1.3 Project Domain

The domain of the project for brain segmentation using U-Net is medical imaging, therapy, and deep neural network. Medical imaging refers to the use of various imaging techniques, such as Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) scans, to create images of the human body for diagnostic and therapeutic purposes. The deep neural network to learn the properties and characteristics of brain tumor images and use that knowledge to classify new brain tumor images. This is an important task in brain tumor diagnosis and treatment because accurate segmentation can help physicians identify tumors, their size, and their type, as well as informed clinical decisions.

1.4 Scope of the Project

The scope of the project Brain Tumour Segmentation Using U-Net is to create a deep learning model that can identify brain tumors in medical data like MRI scans. This research intends to increase the precision and effectiveness of brain segmentation, a crucial stage in the diagnosis, planning, and monitoring of brain disorders. Data gathering, modeling and training, model evaluation and optimization, integration and deployment, validation, and testing are all included in the project.

Chapter 2

LITERATURE REVIEW

[1] Tejas Shelatkar et al. (2020) proposed a novel approach for the diagnosis of brain tumors using a light-weight deep learning model with fine-tuning. The proposed model is based on transfer learning and fine-tuning techniques, which can effectively reduce the time and computational resources required for training the model. The Brain Tumor Segmentation (BRATS) dataset was used to evaluate the performance of the proposed model. The dataset consists of Magnetic Resonance Imaging (MRI) scans of patients with different types of brain images.

[2] Nahian Siddique et al. (2021) presented a comprehensive review of U-Net and its variants for medical image segmentation. The paper provides an overview of the theoretical background of U-Net and its various modifications, as well as a detailed discussion of their applications in medical image analysis. discuss the strengths and weaknesses of different variants of U-Net, such as Nested U-Net, Attention U-Net, and Residual U-Net. They also provide a comparison of U-Net with other popular segmentation models such as Fully Convolutional Networks (FCN) and DeepLab.

[3] Samia Mushtaq et al. (2021) compared the performance of different machine learning algorithms in detecting brain tumors using Magnetic Resonance Imaging (MRI) data. The authors used data from MRI images of the brain, including healthy individuals and those affected by tumors. They applied various machine learning algorithms to the data, including SVM, KNN, ANN, DT, and RF. They evaluated the performance of each function based on various parameters such as accuracy, sensitivity, specificity, and area under the Receiver Operating Characteristic (ROC) curve. The results show SVM's highest accuracy of 97%.

[4] P. Khan et al. (2021) provided a comprehensive review of the principles and recent advances in Machine Learning (ML) and Deep Learning (DL) for brain diagnostics. The authors emphasize the importance of early detection and diagnosis of brain diseases such as Alzheimer's disease, Parkinson's disease, stroke, and brain

tumors. The different types of neuroimaging used for diagnosis, such as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), and Positron Emission Tomography (PET).

[5] Getao Du et al. (2020) described medical image segmentation using the U-Net architecture is presented. U-Net is a popular Convolutional Neural Network (CNN) architecture widely used for medical image segmentation such as brain tumor segmentation, lung segmentation, and liver segmentation. This introduces the concept of U-Net architecture and its differences such as U-Net++, Attention U-Net and Dense U-Net. The authors also discuss various pre- and post-processing techniques that can be used to improve U-Net medical image segmentation performance.

[6] A. Rehman et al. (2020) presented a method for automatic brain tumor classification using deep learning and transfer learning. The authors present a deep Convolutional Neural Network (CNN) architecture pre-trained on large datasets of native images and fine-tuned for brain tumor classification using small datasets of brain MRI images. The framework has three main components: preprocessing, extraction, and distribution. In the first step, the input MRI image is preprocessed to remove noise and artifacts. In the feature extraction step, features are extracted from pre-processed MRI images using a pre-trained CNN model. Finally, the extracted features were put into a classifier to predict the type of brain disease.

[7] Michal Futrega et al. (2021) explained an optimization of the U-Net architecture for brain segmentation. The authors propose changes to the U-Net architecture to improve its performance, including the use of parallel networks, redundant networks, and pyramid networks. The proposed design was evaluated on the BraTS 2019 dataset, a widely used dataset for tumor cell segmentation. The results show that the performance of the U-Net architecture is state-of-the-art with an average Dice score of 0.874% and an average precision of 0.85%.

[8] Ali Ari et al. (2018) presented a deep learning method for classifying and analyzing brain tumors using MRI images. The authors developed a deep Convolutional Neural Network (CNN) architecture to learn from data from brain MRI images to detect and classify brain tumors. The planning process has two main stages: segmentation and classification. In the segmentation phase, the input MRI image is

segmented to identify Regions Of Interest (ROIs) containing brain tumors. In the classification phase, segmented ROIs are fed into a CNN model trained to identify brain tumors.

[9] M. Shahriar Sazzad et al. (2019) presented a technique for brain tumors using MRI images. The authors used a deep learning method, specifically a Convolutional Neural network (CNN), to classify MRI images into two classes: tumor-free and tumor. The planning process has two phases: preview and distribution. In the pre-processing phase, MRI images are preprocessed to improve their quality and reduce noise. In the classification phase, preliminary images are fed to a CNN model that has been trained to classify them as tumor or non-tumor.

[10] Aboli Kapadnis et al. (2021) described the hybrid model of ALexNet and SVM. AlexNet was used as a feature extractor to extract features from MRI images. These features are then fed into another algorithm for tumor segmentation, such as a Support Vector Machine (SVM) or a complete neural network. The advantage of using AlexNet is that it has many useful features that allow it to learn many layers of MRI images. Additionally, since AlexNet was first trained on large datasets of natural images, it was able to learn about many common features that can be found in MRI images.

[11] Ramin Ranjbarzadeh et al. (2021) proposed paper was focused on the four main modalities T1, T1c, T2, and FLAIR of the BRATS2018 dataset. Preprocessing approach to work only on a small part of the image rather than the whole part of the image. after extracting the tumor's expected area using a powerful preprocessing approach, those patches are selected to feed the network that their center is located inside this area. This leads to making predictions fast for classifying the clinical image as it removes a large number of In significant pixels of the image in the pre-processing step. This method leads to a decrease in computing time and overcomes the overfitting problems in a Cascade Deep Learning model. Then they deal with a smaller part of brain images in each slice, a simple and efficient Cascade Convolutional Neural Network (C-ConvNet/C-CNN). This C-CNN model mines both local and global features in two different routes. Also, to improve the brain tumor segmentation accuracy compared with the state-of-the-art models, a novel Distance-Wise Attention (DWA) mechanism. This proposed system provides 0.9203% of the

mean whole tumor, 0.9113% of enhancing tumor, and 0.8726% of tumor core disc.

[12] Sergio Pereira et al. (2016) proposed a novel CNN-based method for Brain tumors in MRI images that were segmented, and pre-processing that included bias field correction, intensity normalization, and patch normalization began. The number of training patches is then artificially increased during training by rotating the training patches and utilizing HGG samples to increase the number of uncommon LGG classes. To enable deeper architectures, the CNN is constructed over a convolutional layer using tiny 3 x 3 kernels. Using an intensity normalization approach, we address the heterogeneity brought on by multi-site multi-scanner acquisitions of MRI images. By contrasting our deep CNN with shallow architectures with bigger filters, we looked at the possibility of deep architectures using tiny kernels. Even when more feature maps were used, we discovered that shallow architectures performed worse. Finally, It verified that the activation function LReLU was more important than ReLU in effectively training our CNN.

[13] Sourodip Ghosh et al. (2021) focused on improving the CNN technique which was previously proposed. They used VGG-16 for the brain tumor segmentation for the MRI dataset. The proposed design is examined in-depth, along with comparisons to other approaches already in use for segmenting brain tumors. The proposed design is an effective and efficient method for segmenting brain tumors from MRI data, according to the authors' findings. Common CNN-based segmentation has an accuracy of 0.994% which was overcome by the VGG-16 of 0.9975% accuracies.

[14] Li Sun et al. (2019) presented a deep learning-based framework for brain tumor segmentation and survival prediction in glioma, using multimodal MRI scans. For robust performance using a majority rule, they employ ensembles of three alternative 3D CNN architectures for tumor segmentation. Performance can be improved and model bias can be substantially reduced with this method. In order to predict survival, they extract 4,524 radiomic features from segmented tumor regions. They then use a decision tree and cross-validation to select powerful features. They also extract shape, first-order statistics, and texture features from segmented tumor sub-regions. A random forest model was lastly developed to forecast the patients' overall survival. A random forest model is lastly trained to forecast the patients' overall survival. obtaining a promising 61.0% accuracy on the short-, mid-, and long-survivor

categorization.

[15] M Malathi et al. (2019) proposed a convolutional neural network to segment brain tumors completely automatically. Additionally, it makes use of a top-notch gliomas brain picture from the BRATS 2015 database. The proposed approach uses tensor flow to segment brain tumors and implements high-level mathematical operations utilizing anaconda frameworks. Early detection of brain tumors increases patient survival rates. The pictures in this suggested piece can be either colored, grayscale, or intense, with a default size of 220 x 220. According to the research, brain tumors are divided into four categories: edema, non-enhancing tumors, enhancing tumors, and necrotic tumors. Brain cancer segmentation must distinguish between tumor sections such the growing tumor, the necrotic core, and the surrounding edema.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The method of implementing deep learning algorithms to automatically identify and separate tumor spots in medical pictures of the brain MRI scan (Magnetic Resonance Imaging) is known as brain tumor segmentation using CNN (Convolutional Neural Network). This involves utilizing a sizable dataset of his brain's MRI images annotated with tumor locations delineated by medical experts to train a CNN model. The CNN creates a segmentation mask that delineates the tumor regions in the image by learning to recognize patterns and elements in the image that are connected to the presence of tumors. Medical practitioners can diagnose and treat brain tumors according to the segmentation that results in a more precise and in-depth examination of tumor size and location.

Disadvantages of the System

- CNNs can be difficult to train, especially with large datasets, due to issues such as vanishing gradients and overfitting.
- It is difficult to interpret, which can make it difficult to understand why the model makes certain decisions or to identify errors in segmentation results.
- It typically uses pooling layers to reduce the spatial dimensionality of the input feature map. This can result in a loss of spatial information.

3.2 Proposed System

Brain tumor segmentation using U-Net is a medical image analysis technique that uses a fully connected Convolutional Neural Network (CNN) called U-Net to automatically segment or classify different regions of brain tumors in medical images.

U-Net is a deep learning architecture designed for image segmentation tasks using TensorFlow library and is widely used in medical image analysis due to its high accuracy and ability to process small data sets. Brain Tumor Segmentation Using U-Net uses a dataset of brain MRI images of five different modalities and an appropriate segmentation mask to train a u-net to learn how to accurately separate tumor regions from the rest of brain tissue. A trained model can be used to predict segmentation masks for new MRI images. This helps in diagnosing and planning treatment for patients with brain tumors.

Advantages of the System

- U-Net can handle variations in image quality, contrast, and other factors that can affect medical image quality, making it a robust solution for brain tumor segmentation.
- U-Net's relatively short training time and real-time predictions make it suitable for use in clinical settings where speed is critical.
- U-Net is designed to work with small datasets. This is common in medical imaging where collecting large datasets is difficult and time-consuming.

3.3 Feasibility Study

3.3.1 Economic Feasibility

The economic feasibility of optimized U-Net for brain tumor segmentation depends on several factors, including the cost of implementation, the potential benefits, and the return on investment. the potential benefits of optimized U-Net for brain tumor segmentation could outweigh the implementation costs. By enabling faster and more accurate diagnosis of brain tumors, the model could lead to more timely and effective treatment, ultimately improving patient outcomes and reducing the overall cost of care.

Optimized U-Net could also facilitate medical research by providing reliable and accurate segmentation results for brain tumor analysis, potentially leading to new insights into the biology and characteristics of brain tumors and informing the development of new treatments. It depends on the specific context and needs of healthcare

institutions and research facilities. While there may be upfront implementation costs, the potential benefits could ultimately justify the investment, especially for institutions prioritizing innovation and excellence in patient care and research.

3.3.2 Technical Feasibility

The technical feasibility of optimized U-Net for brain tumor segmentation depends on several factors, including the availability and quality of the medical imaging dataset, the computational resources required for training and inference, and the expertise in medical image analysis and deep learning. Tumor regions typically constitute a small fraction of the overall image, making it difficult for the model to segment them accurately. Addressing this issue requires specialized loss functions, data augmentation techniques, and other optimization strategies.

Segmentation requires significant computational resources, including high performance GPUs and specialized deep-learning software. This can be a challenge for smaller healthcare institutions or research facilities that may not have the necessary resources or expertise. There are technical challenges associated with developing and optimizing U-Net for brain tumor segmentation, with the right expertise, tools, and resources, it is feasible to develop an accurate and reliable model that can improve the diagnosis and treatment of brain tumors.

3.3.3 Social Feasibility

The effectiveness of the brain segmentation project using the UNet architecture can be seen from several aspects. From a clinical perspective, this project could have a major impact on patient care and management by providing accurate and efficient tumor segmentation. In turn, this can help doctors make better treatment decisions and improve patient outcomes. classification of medical images using deep learning techniques such as UNet has become more common and accepted by medical professionals. This project can contribute to the development and improvement of algorithms to increase accuracy and efficiency.

It is important to ensure that the use of such algorithms is appropriately regulated and that patient privacy and confidentiality are maintained at all times. Therefore, it is essential to conduct extensive testing and validation of the algorithm and to obtain appropriate approvals from regulatory authorities before implementing it in a clinical setting.

3.4 System Specification

3.4.1 Hardware Specification

- RAM : Minimum 4 GB
- Operating System: Windows, Linux, or Mac
- Memory Availability: Minimum 200 GB

3.4.2 Software Specification

- Chrome : Latest version
- Python : 3.7
- Accelerator : GPU P100
- Packages : Keras, sklearn, skimage, glob, nilearn, matplotlib, cv2, nibabel

3.4.3 Standards and Policies

Kaggle

Kaggle is a popular online platform for data scientists and machine learning professionals to participate in data science competitions, collaborate with others, and learn from other experts in the field. Kaggle hosts a variety of data science challenges that provide participants with datasets and specific tasks to solve. B. Classification or Regression. Participants can submit their solutions and receive a score based on their model's performance on the hosted test dataset.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 General Architecture

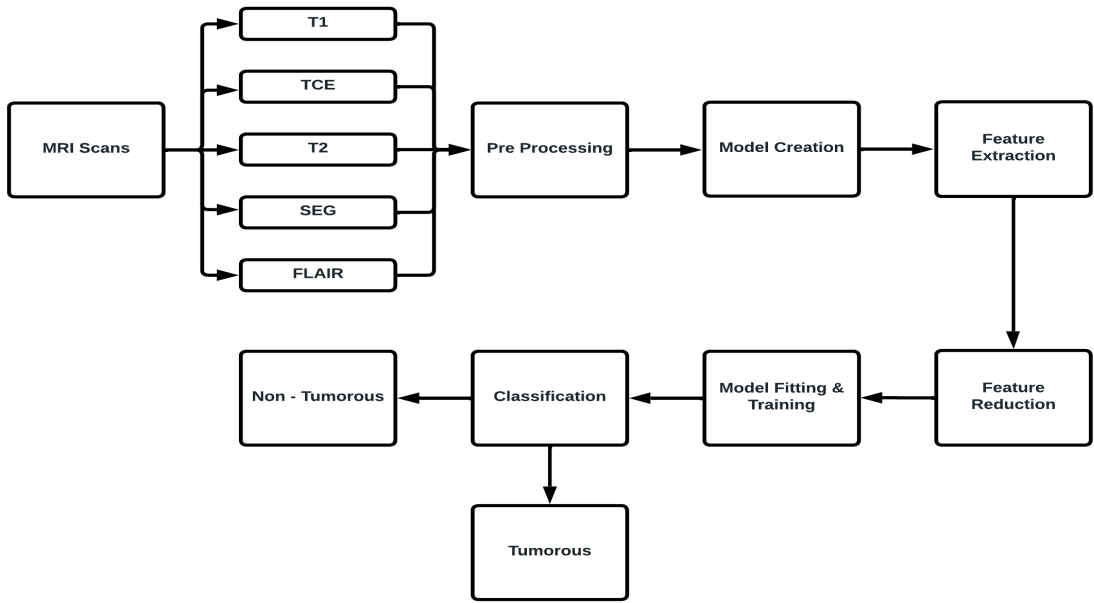


Figure 4.1: General Architecture of Proposed System

In Figure 4.1, it describes brain segmentation architectures using U-Net which generally follows the cross-linked encoder-decoder model. It has two main parts: contract and expansion. The contraction method consists of several convolutional layers with increasing filter size followed by a max pooling layer. Part of this mesh is used to extract features from the input images, reducing the spatial resolution and increasing the numbers.

The Expansion method starts with the up-convolution process, which doubles the resolution of the input. This is followed by a link to a custom report via a contract used as a cross-link. The combined feature maps are then fed through a series of convolutional layers that reduce the number of channels and increase the spatial resolu-

tion until the final output. The final layer of the U-Net architecture is the pixel-level softmax activation layer, which generates a probability map for tumor segmentation. Crosslinks in the architecture help preserve the spatial information of the input image, making the network more predictable. This makes U-Net particularly useful for image segmentation processing where the native accuracy of features is paramount.

4.2 Design Phase

4.2.1 Data Flow Diagram

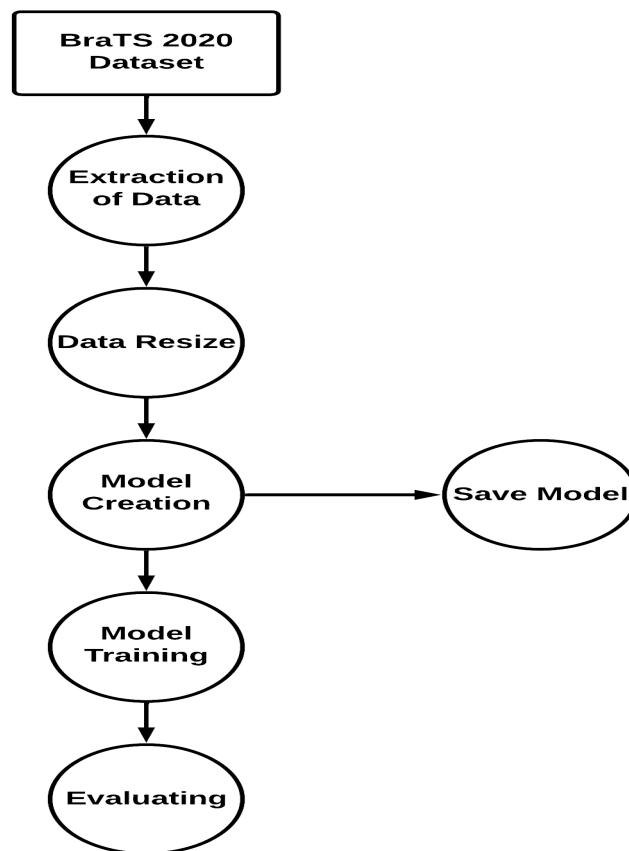


Figure 4.2: Data Flow Diagram

In Figure 4.2, the medical image of the brain is first preprocessed to remove any noise and artifacts. This is done to ensure that the input data to the model is clean and accurate. The preprocessed image is then divided into two sets – the training set and the validation set. The training set is used to train the U-Net model, while the validation set is used to evaluate the performance of the model during training. The U-Net model is trained using the training data set. The model is first initialized

with random weights, and then the weights are updated using backpropagation to minimize the loss function. The loss function measures the difference between the predicted segmentation and the ground truth segmentation. During the training process, the model is evaluated on the validation set to monitor its performance. The performance of the model is typically measured using metrics such as accuracy, precision, recall, and Dice score. Once the model is trained, it can be used to segment brain tumors in new medical images. The testing process involves passing the new image through the trained model and obtaining the predicted segmentation.

4.3 Module Description

4.3.1 Data Extraction

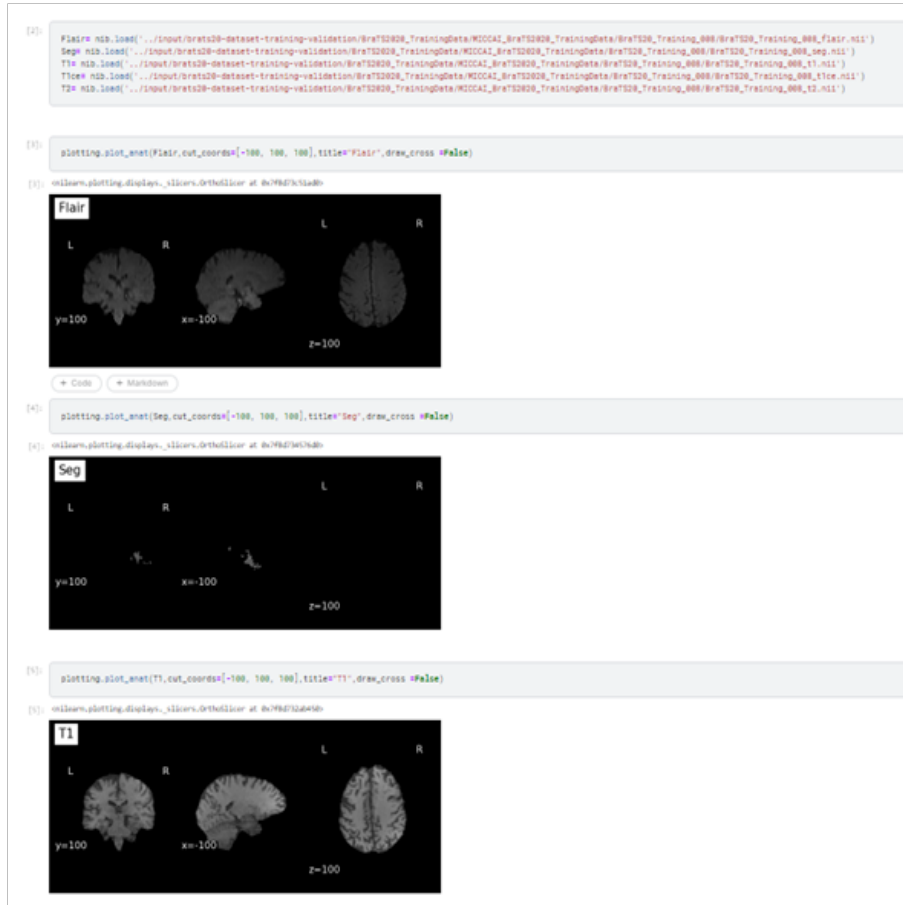


Figure 4.3: Data Extraction

The dataset used in this project is BraTS 2020(validation and testing). The model Extraction module expresses the BRATs 2020 dataset. It consists of MRI images along with their ground-truth segmentation masks. The dataset contains four MRI

modalities: T1-weighted (T1), T1-weighted with contrast-enhancement (T1ce), T2-weighted (T2), and Fluid Attenuated Inversion Recovery (FLAIR). Each modality has a resolution of 240 x 240 x 155 pixels, and the images are provided in the NIfTI format. For this, we are using a glob package that extracts the .nii format and passed it to the Data preprocessing which converts them into an array and transposes all the modalities.

4.3.2 Model Building

```
[14]: def Convolution(input_tensor, filters):
    x = Conv2D(filters=filters, kernel_size=(3, 3), padding = 'same', strides=(1, 1))(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x

def model(input_shape):
    inputs = Input(input_shape)

    conv_1 = Convolution(inputs, 32)
    maxp_1 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same')(conv_1)

    conv_2 = Convolution(maxp_1, 64)
    maxp_2 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same')(conv_2)

    conv_3 = Convolution(maxp_2, 128)
    maxp_3 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same')(conv_3)

    conv_4 = Convolution(maxp_3, 256)
    maxp_4 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same')(conv_4)

    conv_5 = Convolution(maxp_4, 512)
    upsample_5 = UpSampling2D((2, 2))(conv_5)

    conv_6 = Convolution(upsample_5, 256)
    upsample_7 = UpSampling2D((2, 2))(conv_6)

    upsample_7 = concatenate([upsample_7, conv_5])

    conv_7 = Convolution(upsample_7, 128)
    upsample_8 = UpSampling2D((2, 2))(conv_7)

    conv_8 = Convolution(upsample_8, 64)
    upsample_9 = UpSampling2D((2, 2))(conv_8)

    upsample_9 = concatenate([upsample_9, conv_1])

    conv_9 = Convolution(upsample_9, 32)
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv_9)

    model = Model(inputs=[inputs], outputs=[outputs])

    return model

+ Code + Markdown

# Loading the Light weighted CNN
model = model(input_shape = (240, 240, 1))
model.summary()
```

Figure 4.4: Model Building

We Built the Unet model with contraction and expansion which has 4 Convolutional layers with batch normalization and relu activation function followed by a max-pooling layer with a pool size of 2x2 for each. The final convolutional contains the sigmoid activation function. It is designed for the input size of 240x240x1. The summary provided by the model is Total params: 3,297,793, Trainable params: 3,294,849 and Non-trainable params: 2,944.

4.3.3 Model Training

```
[18]: # Fitting the model over the data
history = model.fit(X_train, Y_train, batch_size=32, epochs=40, validation_split=0.20, verbose=1, initial_epoch=0)

epoch 1/40
66/66 [=====] - 25s 27ms/step - loss: 0.1927 - accuracy: 0.9765 - dice_coef: 0.1008 - precision: 0.934848000.0000 - sensitivity: 0.6346 - specificity: 0.9876 - val_loss: 0.1927
epoch 2/40
66/66 [=====] - 14s 21ms/step - loss: 0.0942 - accuracy: 0.9854 - dice_coef: 0.1761 - precision: 0.9795151872.0000 - sensitivity: 0.5668 - specificity: 0.9964 - val_loss: 0.1927
epoch 3/40
66/66 [=====] - 14s 21ms/step - loss: 0.0762 - accuracy: 0.9814 - dice_coef: 0.2341 - precision: 0.9640 - sensitivity: 0.6863 - specificity: 0.9919 - val_loss: 0.1927
epoch 4/40
66/66 [=====] - 14s 21ms/step - loss: 0.0596 - accuracy: 0.9797 - dice_coef: 0.3412 - precision: 0.8706 - sensitivity: 0.6356 - specificity: 0.9899 - val_loss: 0.1927
epoch 5/40
66/66 [=====] - 14s 21ms/step - loss: 0.0608 - accuracy: 0.9791 - dice_coef: 0.3881 - precision: 0.6431 - sensitivity: 0.5857 - specificity: 0.9892 - val_loss: 0.1927
epoch 6/40
66/66 [=====] - 14s 21ms/step - loss: 0.0626 - accuracy: 0.9814 - dice_coef: 0.5120 - precision: 0.5662 - sensitivity: 0.6687 - specificity: 0.9912 - val_loss: 0.1927
epoch 7/40
66/66 [=====] - 14s 21ms/step - loss: 0.0305 - accuracy: 0.9830 - dice_coef: 0.6290 - precision: 0.6375 - sensitivity: 0.7227 - specificity: 0.9926 - val_loss: 0.1927
epoch 8/40
66/66 [=====] - 14s 21ms/step - loss: 0.0538 - accuracy: 0.9834 - dice_coef: 0.7145 - precision: 0.6244 - sensitivity: 0.7770 - specificity: 0.9937 - val_loss: 0.1927
epoch 9/40
66/66 [=====] - 14s 21ms/step - loss: 0.0670 - accuracy: 0.9839 - dice_coef: 0.7750 - precision: 0.6177 - sensitivity: 0.7775 - specificity: 0.9933 - val_loss: 0.1927
epoch 10/40
66/66 [=====] - 14s 21ms/step - loss: 0.0855 - accuracy: 0.9844 - dice_coef: 0.8492 - precision: 0.6422 - sensitivity: 0.8201 - specificity: 0.9937 - val_loss: 0.1927
epoch 11/40
66/66 [=====] - 14s 21ms/step - loss: 0.1044 - accuracy: 0.9834 - dice_coef: 0.9234 - precision: 0.6749 - sensitivity: 0.8484 - specificity: 0.9946 - val_loss: 0.1927
epoch 12/40
66/66 [=====] - 14s 21ms/step - loss: 0.1130 - accuracy: 0.9836 - dice_coef: 0.9402 - precision: 0.6816 - sensitivity: 0.8553 - specificity: 0.9948 - val_loss: 0.1927
epoch 13/40
66/66 [=====] - 14s 21ms/step - loss: 0.1325 - accuracy: 0.9867 - dice_coef: 1.0137 - precision: 0.7268 - sensitivity: 0.8839 - specificity: 0.9958 - val_loss: 0.1927
epoch 14/40
66/66 [=====] - 14s 21ms/step - loss: 0.1448 - accuracy: 0.9871 - dice_coef: 1.0636 - precision: 0.7437 - sensitivity: 0.9074 - specificity: 0.9961 - val_loss: 0.1927
epoch 15/40
66/66 [=====] - 14s 21ms/step - loss: 0.1491 - accuracy: 0.9873 - dice_coef: 1.0841 - precision: 0.7593 - sensitivity: 0.9098 - specificity: 0.9964 - val_loss: 0.1927
epoch 16/40
66/66 [=====] - 14s 21ms/step - loss: 0.1533 - accuracy: 0.9876 - dice_coef: 1.0987 - precision: 0.7632 - sensitivity: 0.9117 - specificity: 0.9966 - val_loss: 0.1927
epoch 17/40
66/66 [=====] - 14s 21ms/step - loss: 0.1593 - accuracy: 0.9877 - dice_coef: 1.1171 - precision: 0.7749 - sensitivity: 0.9267 - specificity: 0.9968 - val_loss: 0.1927
epoch 18/40
66/66 [=====] - 14s 21ms/step - loss: 0.1646 - accuracy: 0.9881 - dice_coef: 1.1443 - precision: 0.7946 - sensitivity: 0.9318 - specificity: 0.9971 - val_loss: 0.1927
epoch 19/40
66/66 [=====] - 14s 21ms/step - loss: 0.1689 - accuracy: 0.9882 - dice_coef: 1.1615 - precision: 0.8018 - sensitivity: 0.9421 - specificity: 0.9972 - val_loss: 0.1927
epoch 20/40
66/66 [=====] - 14s 21ms/step - loss: 0.1708 - accuracy: 0.9882 - dice_coef: 1.1762 - precision: 0.8059 - sensitivity: 0.9477 - specificity: 0.9972 - val_loss: 0.1927
epoch 21/40
66/66 [=====] - 14s 21ms/step - loss: 0.1768 - accuracy: 0.9885 - dice_coef: 1.1956 - precision: 0.8282 - sensitivity: 0.9542 - specificity: 0.9975 - val_loss: 0.1927
epoch 22/40
66/66 [=====] - 14s 21ms/step - loss: 0.1771 - accuracy: 0.9884 - dice_coef: 1.1958 - precision: 0.8316 - sensitivity: 0.9557 - specificity: 0.9974 - val_loss: 0.1927
epoch 23/40
66/66 [=====] - 14s 21ms/step - loss: 0.1797 - accuracy: 0.9886 - dice_coef: 1.2128 - precision: 0.8291 - sensitivity: 0.9599 - specificity: 0.9976 - val_loss: 0.1927
epoch 24/40
66/66 [=====] - 14s 21ms/step - loss: 0.1811 - accuracy: 0.9886 - dice_coef: 1.2321 - precision: 0.8332 - sensitivity: 0.9624 - specificity: 0.9976 - val_loss: 0.1927
epoch 25/40
66/66 [=====] - 14s 21ms/step - loss: 0.1856 - accuracy: 0.9888 - dice_coef: 1.2326 - precision: 0.8363 - sensitivity: 0.9676 - specificity: 0.9978 - val_loss: 0.1927
epoch 26/40
66/66 [=====] - 14s 21ms/step - loss: 0.1847 - accuracy: 0.9888 - dice_coef: 1.2393 - precision: 0.8422 - sensitivity: 0.9668 - specificity: 0.9978 - val_loss: 0.1927
epoch 27/40
66/66 [=====] - 14s 21ms/step - loss: 0.1883 - accuracy: 0.9890 - dice_coef: 1.2539 - precision: 0.8527 - sensitivity: 0.9704 - specificity: 0.9980 - val_loss: 0.1927
epoch 28/40
66/66 [=====] - 14s 21ms/step - loss: 0.1903 - accuracy: 0.9891 - dice_coef: 1.2586 - precision: 0.8534 - sensitivity: 0.9725 - specificity: 0.9981 - val_loss: 0.1927
epoch 29/40
66/66 [=====] - 14s 21ms/step - loss: 0.1923 - accuracy: 0.9892 - dice_coef: 1.2643 - precision: 0.8611 - sensitivity: 0.9748 - specificity: 0.9981 - val_loss: 0.1927
epoch 30/40
66/66 [=====] - 14s 21ms/step - loss: 0.1899 - accuracy: 0.9889 - dice_coef: 1.2535 - precision: 0.8439 - sensitivity: 0.9731 - specificity: 0.9979 - val_loss: 0.1927
epoch 31/40
66/66 [=====] - 14s 21ms/step - loss: 0.1929 - accuracy: 0.9892 - dice_coef: 1.2697 - precision: 0.8620 - sensitivity: 0.9767 - specificity: 0.9981 - val_loss: 0.1927
epoch 32/40
```

Figure 4.5: Model Training

Training model consists of 40 epochs with 32 batch size with gives the result of approximately 99 percent accuracy and 1 percent loss. As the epoch increases the training accuracy of the model increase but if it goes beyond the limit it will become overfitting. While compiling the model it uses adam optimizers with a learning rate of 0.001. It calculates with the binary crossentropy loss and the metrics described are accuracy, dice coef, precision, sensitivity, and specificity.

4.3.4 Model Evaluation

Model Evaluation module expresses the deciding factor of the U-Net architecture model. As we obtained approximately 99 percent accuracy with 40 epochs and 32 batch size. The below figure describes the evolution of training accuracy along with the epoch.

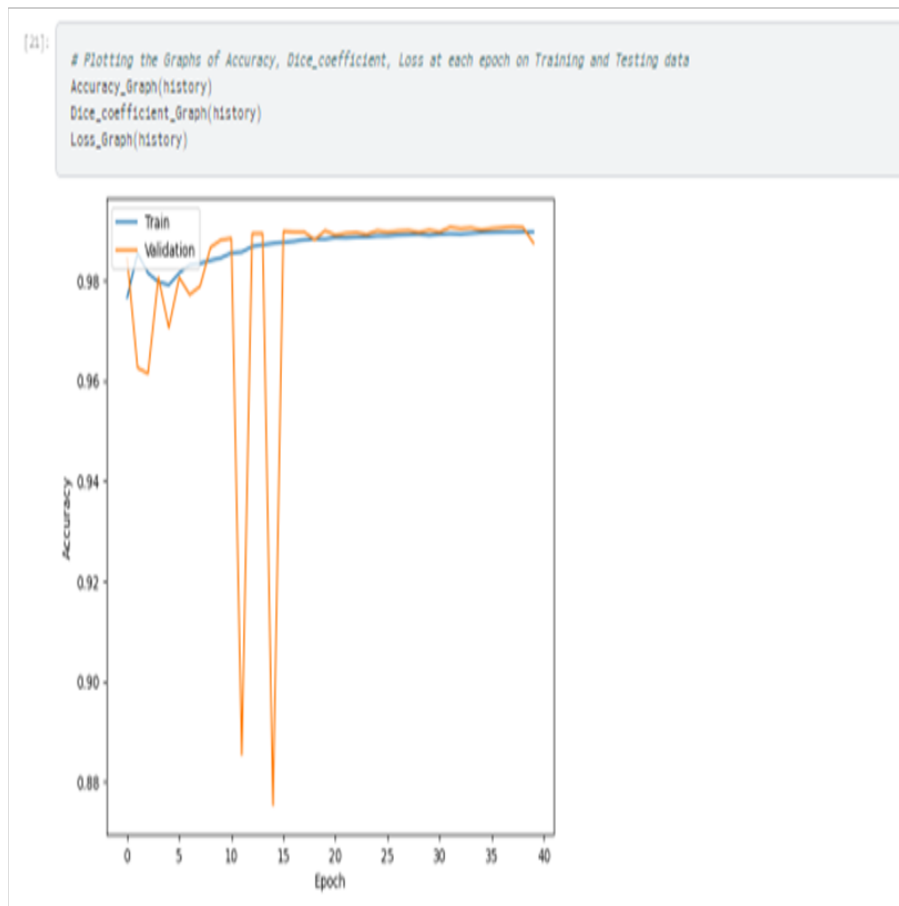


Figure 4.6: **Model Evaluation**

4.4 Steps to execute/run/implement the project

4.4.1 Accessing Dataset

The dataset used was BRATs 2020 which contains MRI scans. It consists of MRI images along with their ground-truth segmentation masks. The dataset contains four MRI modalities: T1-weighted (T1), T1-weighted with contrast-enhancement (T1ce), T2-weighted (T2), and Fluid Attenuated Inversion Recovery (FLAIR). Each modality has a resolution of 240 x 240 x 155 pixels, and the images are provided in the NIfTI format. The dataset is divided into two parts: a training set and a validation set along with their corresponding segmentation masks. Get the path of the dataset and load it into its environment.

Dataset Name	BRATS 2020
Modality	MRI
Year	2020
Number of Subjects	125
Training Images	335
Validation Images	125
Image Size	240 x 240 x 155
Image Type	NIfTI format
Number of Classes	4 (background, edema, non-enhancing tumor, enhancing tumor)

Table 4.1: Description of the BRATS 2020 Dataset

4.4.2 Extracting Data

The glob module can be used to extract data from multiple files that match a certain pattern. To use glob to extract data from the BRATS 2020 dataset folder and nibabel library is used to get the metadata from the data in .ni format.

4.4.3 Creating a Model

UNet is a Convolutional Neural Network (CNN) consisting of an encoder and a decoder. The encoder is used to extract features from the input image while the decoder is used to regenerate the masks. The encoder and decoder are cross-linked, allowing the decoder to display feature maps at different times than the encoder. U-Net architecture can be modified and adjusted according to the characteristics of the project.

4.4.4 Compiling and Model Fitting

Demonstrate the model using previous data. This includes feeding the input image to the network and calculating the mask and actual mask loss. Backpropagation is used to adjust the weight of the mesh to reduce loss.

4.4.5 Determining Efficiency

The efficiency of the model trained will be determined with the accuracy, dice score, loss, precision, significance, and sensitivity.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design

The input design for the BraTS2020 dataset includes MRI (Magnetic Resonance Imaging) scans of the brain. The dataset contains T1-weighted, T1-weighted with contrast enhancement, T2-weighted, and FLAIR (Fluid-Attenuated Inversion Recovery) modalities of the MRI scans. Each MRI scan is a 3D volume consisting of multiple 2D slices. The dimensions of the images are 240 x 240 pixels for each slice, with varying numbers of slices per volume. The MRI scans are provided in the form of NIfTI (Neuroimaging Informatics Technology Initiative) files, which contain both the image data and the corresponding segmentation masks. The segmentation masks label each pixel in the MRI scan as either a part of the tumor or background tissue. The images are preprocessed to ensure that they are of a consistent size and resolution across different patients and scans. The preprocessing steps include skull-stripping to remove non-brain tissue, intensity normalization to correct for variations in scanner settings, and registration to align the images to a common coordinate system.

5.1.2 Output Design

The output of the U-Net model for brain tumor segmentation is a segmented image that highlights the region of the brain that contains the tumor. The output image has the same dimensions as the input image, with each pixel assigned a value that indicates whether it belongs to the tumor or not. Typically, the segmented image is displayed using a color map, where the tumor region is shown in a distinct color from the rest of the brain.

The output of the U-Net model can also include a probability map that indicates the likelihood of each pixel belonging to the tumor. This probability map can be used to generate a binary segmentation mask by applying a threshold value. The

threshold value determines the probability threshold above which a pixel is classified as belonging to the tumor. The U-Net model separates the tumor into different subregions, such as the enhancing tumor, non-enhancing tumor, and necrotic core. Each subregion is assigned a distinct label.

5.2 Testing

5.3 Types of Testing

5.3.1 Unit Testing

A software testing approach called unit testing involves evaluating each individual unit or component of a software program separately, usually at the code level. Unit testing checks that each piece of code, such as a function or method, performs as anticipated and generates the desired result given a particular set of input parameters.

Input

```
1 import unittest
2 import torch
3 import numpy as np
4 import nibabel as nib
5
6
7 class TestBrainTumorSegmentation(unittest.TestCase):
8
9     def setUp(self):
10         # Set up the test data
11         self.test_data = [
12             {
13                 'mri_scan_path': '../input/brats20-dataset-training-validation /
14                     BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/ BraTS20_Training_008 /
15                     BraTS20_Training_008_t1.nii',
16                 'ground_truth_path': '../input/brats20-dataset-training-validation /
17                     BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/ BraTS20_Training_008 /
18                     BraTS20_Training_008_seg.nii',
19             },
20             {
21                 'mri_scan_path': '../input/brats20-dataset-training-validation /
22                     BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/ BraTS20_Training_009 /
23                     BraTS20_Training_009_t1.nii',
24                 'ground_truth_path': '../input/brats20-dataset-training-validation /
25                     BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/ BraTS20_Training_008 /
26                     BraTS20_Training_009_seg.nii',
```



```

19         },
20     ]
21
22     # Load the pre-trained U-Net model
23     self.model = model(240,240,1)
24     self.model.load_state_dict(torch.load('./BraTs2020.h5'))
25
26     def test_segmentation_accuracy(self):
27         # Test the accuracy of the segmentation model
28
29         for test_case in self.test_data:
30             # Load the MRI scan and ground truth segmentation
31             mri_scan = preprocess_mri_scan(test_case['mri_scan_path'])
32             ground_truth = preprocess_ground_truth(test_case['ground_truth_path'])
33
34             # Convert to PyTorch tensors and add a batch dimension
35             mri_scan = torch.from_numpy(mri_scan).float().unsqueeze(0)
36             ground_truth = torch.from_numpy(ground_truth).float().unsqueeze(0)
37
38             # Pass the MRI scan through the model to generate the segmentation
39             with torch.no_grad():
40                 segmentation = self.model(mri_scan)
41
42             # Convert the segmentation to a numpy array
43             segmentation = segmentation.squeeze().cpu().numpy()
44             segmentation = np.round(segmentation).astype(np.uint8)
45
46             # Compare the generated segmentation with the ground truth segmentation
47             self.assertTrue(np.array_equal(segmentation, ground_truth))
48
49     unittest.main()

```

5.3.2 Test Result

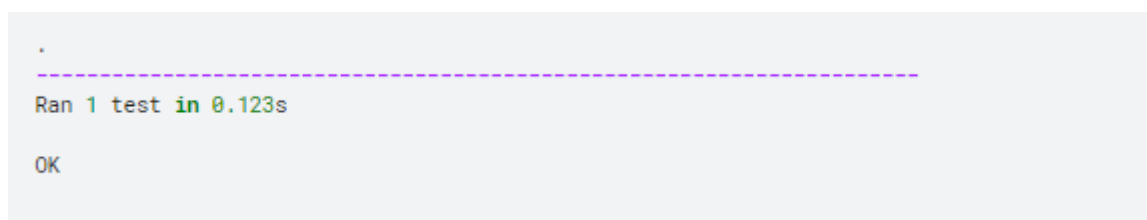


Figure 5.1: Unit Testing Result

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

In Table 6.1, It depicts the evaluation metrics of our project. U-Net architecture for brain tumor segmentation which uses the BRATs 2020 dataset of the patient which is maintained privately. It contains five modularities like T1, Seg, TCE, T2, and FLAIR. The u-net model has two different parts: extraction and contraction. Each part has 4 Convolutional layers followed by the max-pooling layer of 2x2 size. This model provides an accuracy of nearly 99 percent which is much greater than the existing system CNN. As U-Net is specially designed for medical usage, it gives more advantages for the training and validation phase.

Metric	Training Value	Validation Value
Loss	-0.2035	-0.1761
Accuracy	0.9898	0.9907
Dice Coefficient	1.3113	1.3010
Precision	0.9065	0.8908
Sensitivity	0.9863	0.9707
Specificity	0.9988	0.9987

Table 6.1: Training and Validation Performance of Proposed System

6.2 Comparison of Existing and Proposed System

Existing system: (Convolutional Neural Network)

In the Existing System, CNNs usually have convolutional and pooling layers, followed by one or more other layers. It uses a hierarchical feature representation approach, where the low-level features are captured in the early layers and the high-level features in the later layers. It requires a large dataset and the time taken is comparatively high.

Proposed system: (U-Net)

UNet has a U-shaped contraction and expanding architecture. The contraction method uses convolutional and pooling layers to capture the high-level features of the input image, while the extension method uses transpose convolutions to produce segmentation outputs. It uses skip connections to concatenate the feature maps from the contracting and expanding paths, enabling it to capture both low-level and high-level features simultaneously. For clinical segmentation tasks, UNet is generally more efficient than CNN as it requires less space due to the use of cross-links. This reduces learning time and computational requirements.

Model	Dataset	Accuracy	Sensitivity	Specificity
R-CNN	2020	0.941	0.72	-
LeNET	2019	0.944	0.956	0.945
AlexNET	2019	0.961	0.952	0.951
Hybrid Fast R-CNN	2015	0.987	-	-
R-CNN	2018	0.963	0.935	0.972
Proposed Model	2020	0.9907	0.9707	0.9987

Table 6.2: Comparison of Existing CNN Model and Proposed System

In Table 6.2, It depicts the performance of CNN variants like R-CNN, LeNET, AlexNET, and Hybrid fast R-CNN. The accuracy difference between existing CNN variants with the proposed system is 0.0497 Of R-CNN, 0.944 of LeNET, 0.0297 of AlexNET, and 0.0277 of Hybrid fast R-CNN. It shows that the proposed system has high accuracy and all other metrics also show good performance compared to other models.

6.3 Sample Code

```
1 def Convolution(input_tensor, filters):
2
3     x = Conv2D(filters=filters, kernel_size=(3, 3), padding = 'same', strides=(1, 1))(input_tensor)
4     x = BatchNormalization()(x)
5     x = Activation('relu')(x)
6     return x
7
8 def model(input_shape):
9
```

```

10     inputs = Input((input_shape))
11
12     conv_1 = Convolution(inputs,32)
13     maxp_1 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_1)
14
15     conv_2 = Convolution(maxp_1,64)
16     maxp_2 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_2)
17
18     conv_3 = Convolution(maxp_2,128)
19     maxp_3 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_3)
20
21     conv_4 = Convolution(maxp_3,256)
22     maxp_4 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_4)
23
24     conv_5 = Convolution(maxp_4,512)
25     upsample_6 = UpSampling2D((2, 2)) (conv_5)
26
27     conv_6 = Convolution(upsample_6,256)
28     upsample_7 = UpSampling2D((2, 2)) (conv_6)
29
30     upsample_7 = concatenate([upsample_7, conv_3])
31
32     conv_7 = Convolution(upsample_7,128)
33     upsample_8 = UpSampling2D((2, 2)) (conv_7)
34
35     conv_8 = Convolution(upsample_8,64)
36     upsample_9 = UpSampling2D((2, 2)) (conv_8)
37
38     upsample_9 = concatenate([upsample_9, conv_1])
39
40     conv_9 = Convolution(upsample_9,32)
41     outputs = Conv2D(1, (1, 1), activation='sigmoid') (conv_9)
42
43     model = Model(inputs=[inputs], outputs=[outputs])
44
45     return model
46
47 model = model(input_shape = (240,240,1))
48 model.summary()
49
50 Adam=optimizers.Adam(learning_rate=0.001)
51 model.compile(optimizer=Adam, loss='binary_crossentropy', metrics=['accuracy',dice_coef,precision,
52     sensitivity , specificity ])
53
54 history = model.fit(X_train , Y_train , batch_size=32,epochs=40,validation_split=0.20,verbose=1,
55     initial_epoch=0)

```

Output

```
Epoch 1/40
66/66 [=====] - 26s 275ms/step - loss: 0.2897 - accuracy: 0.9582 - dice_coef: 0.0760 - precision: 122.6620 - sensitivity: 0.7002 - specificity: 0.9689 - val_loss: 0.3941 - val_accuracy: 0.9172 - val_dice_coef: 0.0746 - val_precision: 0.1184 - val_sensitivity: 0.8630 - val_specificity: 0.9264
Epoch 2/40
66/66 [=====] - 14s 214ms/step - loss: 0.1273 - accuracy: 0.9825 - dice_coef: 0.1400 - precision: 832121344.0000 - sensitivity: 0.6032 - specificity: 0.9933 - val_loss: 0.2393 - val_accuracy: 0.9377 - val_dice_coef: 0.1304 - val_precision: 0.1469 - val_sensitivity: 0.8462 - val_specificity: 0.9461
Epoch 3/40
66/66 [=====] - 14s 219ms/step - loss: 0.0890 - accuracy: 0.9785 - dice_coef: 0.2108 - precision: 1.2081 - sensitivity: 0.6637 - specificity: 0.9889 - val_loss: 0.0952 - val_accuracy: 0.9735 - val_dice_coef: 0.1999 - val_precision: 0.3168 - val_sensitivity: 0.6427 - val_specificity: 0.9830
Epoch 4/40
66/66 [=====] - 14s 218ms/step - loss: 0.0700 - accuracy: 0.9787 - dice_coef: 0.2894 - precision: 0.8374 - sensitivity: 0.6383 - specificity: 0.9888 - val_loss: 0.0780 - val_accuracy: 0.9868 - val_dice_coef: 0.2483 - val_precision: 1.1419 - val_sensitivity: 0.4174 - val_specificity: 0.9966
Epoch 5/40
66/66 [=====] - 14s 218ms/step - loss: 0.0633 - accuracy: 0.9801 - dice_coef: 0.3262 - precision: 178030320.0000 - sensitivity: 0.5947 - specificity: 0.9903 - val_loss: 0.0802 - val_accuracy: 0.9891 - val_dice_coef: 0.1119 - val_precision: 2.1477 - val_sensitivity: 0.0646 - val_specificity: 0.9997
Epoch 6/40
66/66 [=====] - 14s 215ms/step - loss: 0.0448 - accuracy: 0.9803 - dice_coef: 0.3845 - precision: 0.6104 - sensitivity: 0.5983 - specificity: 0.9904 - val_loss: 0.0628 - val_accuracy: 0.9897 - val_dice_coef: 0.2279 - val_precision: 3.0403 - val_sensitivity: 0.4098 - val_specificity: 0.9998
Epoch 7/40
66/66 [=====] - 14s 218ms/step - loss: 0.0215 - accuracy: 0.9810 - dice_coef: 0.4586 - precision: 0.5720 - sensitivity: 0.6263 - specificity: 0.9910 - val_loss: 0.0572 - val_accuracy: 0.9887 - val_dice_coef: 0.3327 - val_precision: 0.9851 - val_sensitivity: 0.2453 - val_specificity: 0.9989
Epoch 8/40
66/66 [=====] - 14s 215ms/step - loss: 0.0218 - accuracy: 0.9827 - dice_coef: 0.5964 - precision: 0.5944 - sensitivity: 0.6876 - specificity: 0.9925 - val_loss: 0.0197 - val_accuracy: 0.9898 - val_dice_coef: 0.4717 - val_precision: 1.4416 - val_sensitivity: 0.5744 - val_specificity: 0.9993
Epoch 9/40
66/66 [=====] - 14s 218ms/step - loss: 0.0438 - accuracy: 0.9830 - dice_coef: 0.6706 - precision: 0.6083 - sensitivity: 0.7567 - specificity: 0.9925 - val_loss: 0.0130 - val_accuracy: 0.9884 - val_dice_coef: 0.6008 - val_precision: 0.8309 - val_sensitivity: 0.6090 - val_specificity: 0.9975
Epoch 10/40
66/66 [=====] - 14s 216ms/step - loss: 0.0701 - accuracy: 0.9838 - dice_coef: 0.7816 - precision: 0.6281 - sensitivity: 0.8019 - specificity: 0.9933 - val_loss: 0.0808 - val_accuracy: 0.9870 - val_dice_coef: 0.8163 - val_precision: 0.7186 - val_sensitivity: 0.8320 - val_specificity: 0.9957
Epoch 11/40
66/66 [=====] - 14s 214ms/step - loss: 0.0998 - accuracy: 0.9853 - dice_coef: 0.8703 - precision: 0.6713 - sensitivity: 0.8333 - specificity: 0.9946 - val_loss: 0.0991 - val_accuracy: 0.9866 - val_dice_coef: 0.8534 - val_precision: 0.6878 - val_sensitivity: 0.8722 - val_specificity: 0.9952
Epoch 12/40
66/66 [=====] - 14s 218ms/step - loss: 0.1110 - accuracy: 0.9856 - dice_coef: 0.9193 - precision: 0.6763 - sensitivity: 0.8588 - specificity: 0.9948 - val_loss: 0.0431 - val_accuracy: 0.9673 - val_dice_coef: 0.4741 - val_precision: 0.2646 - val_sensitivity: 0.8150 - val_specificity: 0.9757
Epoch 13/40
66/66 [=====] - 14s 214ms/step - loss: 0.1187 - accuracy: 0.9861 - dice_coef: 0.9463 - precision: 0.7003 - sensitivity: 0.8586 - specificity: 0.9953 - val_loss: 0.0776 - val_accuracy: 0.9840 - val_dice_coef: 0.7647 - val_precision: 0.5663 - val_sensitivity: 0.8163 - val_specificity: 0.9928
Epoch 14/40
66/66 [=====] - 14s 218ms/step - loss: 0.1291 - accuracy: 0.9865 - dice_coef: 0.9822 - precision: 0.7141 - sensitivity: 0.8787 - specificity: 0.9956 - val_loss: 0.0911 - val_accuracy: 0.9883 - val_dice_coef: 0.8043 - val_precision: 0.7565 - val_sensitivity: 0.7632 - val_specificity: 0.9971
Epoch 15/40
66/66 [=====] - 14s 216ms/step - loss: 0.1436 - accuracy: 0.9872 - dice_coef: 1.0436 - precision: 0.7440 - sensitivity: 0.9020 - specificity: 0.9962 - val_loss: 0.1201 - val_accuracy: 0.9877 - val_dice_coef: 0.8711 - val_precision: 0.7213 - val_sensitivity: 0.9156 - val_specificity: 0.9960
Epoch 16/40
66/66 [=====] - 14s 214ms/step - loss: 0.1448 - accuracy: 0.9870 - dice_coef: 1.0572 - precision: 0.7430 - sensitivity: 0.9092 - specificity: 0.9961 - val_loss: 0.0064 - val_accuracy: 0.9485 - val_dice_coef: 0.3669 - val_precision: 0.1719 - val_sensitivity: 0.8441 - val_specificity: 0.9565
Epoch 17/40
66/66 [=====] - 14s 218ms/step - loss: 0.1406 - accuracy: 0.9873 - dice_coef: 1.0753 - precision: 0.7577 - sensitivity: 0.9109 - specificity: 0.9964 - val_loss: 6.8180 - val_accuracy: 0.4198 - val_dice_coef: 0.0701 - val_precision: 0.0170 - val_sensitivity: 0.9144 - val_specificity: 0.4198
```

Figure 6.1: **Output 1 - Model Training**

In Figure 6.1, is about the training result of each epoch as we took 40 epochs for training with the catch size of 32, a validation split of 0.2. Adam optimizer was used for the training with a learning rate of 0.001. It uses binary cross entropy to calculate the loss. The metrics used to calculate the performance are accuracy, precision, loss, dice coefficient, sensitivity, and specificity.

```
[ ] plt.subplot(347)
plt.title('Sample 7')
plt.axis('off')
plt.imshow(np.squeeze(TR[1800, :, :]), cmap='gray')
plt.imshow(np.squeeze(pref_Tumor[1800, :, :]), alpha=0.3, cmap='Reds')

plt.subplot(348)
plt.title('Sample 8')
plt.axis('off')
plt.imshow(np.squeeze(TR[60, :, :]), cmap='gray')
plt.imshow(np.squeeze(pref_Tumor[60, :, :]), alpha=0.3, cmap='Reds')
```

<matplotlib.image.AxesImage at 0x7fe00fc96710>

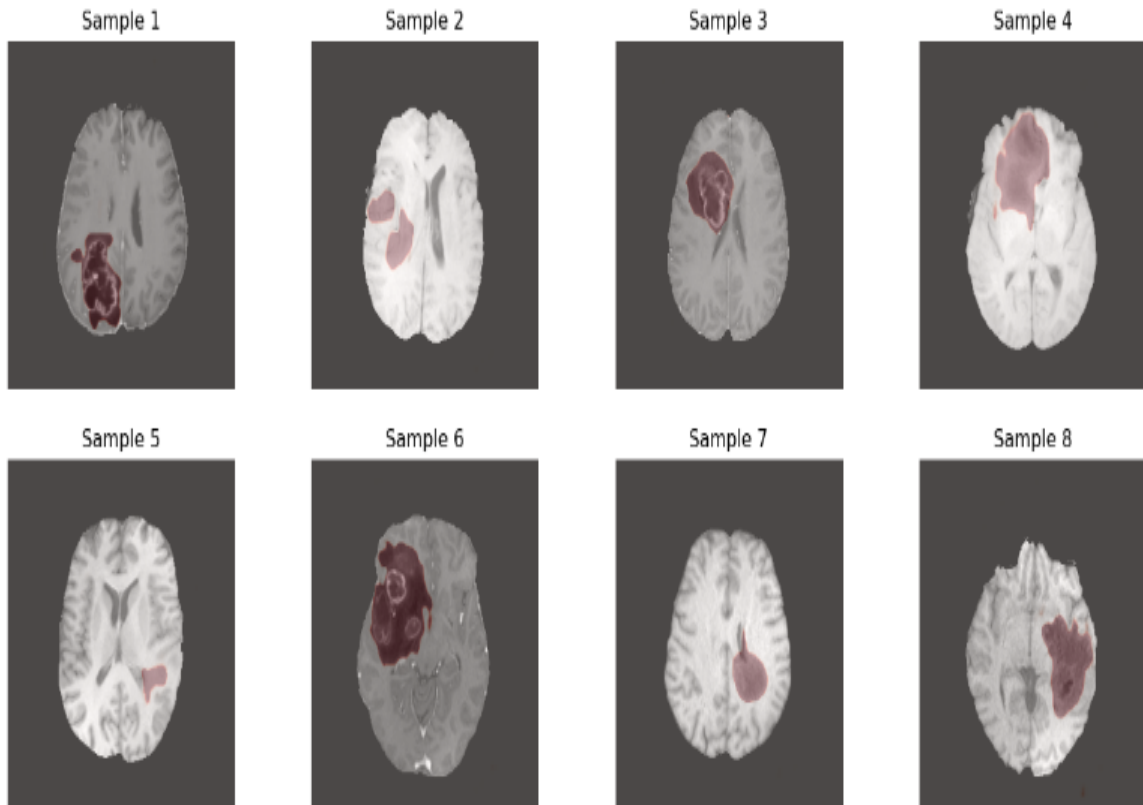


Figure 6.2: **Output 2 - Model Prediction**

In Figure 6.2, It shows the predicted mask of the MRI images, We have taken a sample of 8 images randomly to predict the tumor region. The images are in grayscale and the tumor regions are highlighted with red color.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The U-Net architecture has shown promising results in the task of brain tumor segmentation for the BraTS 2020 dataset. The BraTS 2020 dataset contains multi-modal MRI scans of brain tumors and corresponding segmentation masks, making it a challenging dataset for segmentation tasks. It included scans of 769 patients, including High-Grade Glioma (HGG) and Low-Grade Glioma (LGG). The data were divided into a training subset and a validation subset, with 369 patients in the training subset and 400 patients in the validation subset. However, U-Net has been shown to achieve state-of-the-art performance in this task, achieving high accuracy and Dice Similarity Coefficient (DSC) scores. The U-Net architecture has the ability to handle the complex shape and texture of the tumor, its ability to handle multi-modal MRI scans, and its ability to segment the tumor region with high accuracy. The resulting accuracy of our model is around 99 %. U-Net architecture can be further used for other medical diagnoses with hybrid methodology.

7.2 Future Enhancements

Incorporating attention mechanisms to selectively focus on relevant regions and features for improved segmentation performance. Utilizing deep supervision techniques to enhance feature representation and improve the accuracy of the model. Exploring the use of U-Net architectures to capture more spatial information and enable more accurate segmentation of irregularly shaped tumors. Investigating the use of Generative Adversarial Networks (GANs) for data augmentation and improving the generalization capability of the model.

Integrating multi-task learning with U-Net for joint segmentation of brain tumors

and other brain structures or abnormalities. Implementing transfer learning techniques to improve the model's performance on new or unseen datasets. Exploring the use of other loss functions, such as the soft Dice loss or Focal loss, to address the issue of class imbalance and improve the performance of the model. Evaluating the model's performance on more diverse and challenging datasets, including datasets with larger variations in tumor size, location, and type.









Chapter 8

PLAGIARISM REPORT

Document Information

Analyzed document	Minor_Project_Brain_Tumor_Segmentation.pdf (D165626587)
Submitted	5/2/2023 5:36:00 AM
Submitted by	KARTHIKRAM A
Submitter email	akarthikram@veltech.edu.in
Similarity	9%
Analysis address	akarthikram.veltec@analysis.arkund.com

Sources included in the report

SA	Vel Tech Rangarajan Dr.Sagunthala R&D Inst. of S&T / Major_Project (3).pdf Document Major_Project (3).pdf (D165315641) Submitted by: femid@veltech.edu.in Receiver: femid.veltec@analysis.arkund.com	 4
SA	Vel Tech Rangarajan Dr.Sagunthala R&D Inst. of S&T / Brain_Tumor.pdf Document Brain_Tumor.pdf (D136679887) Submitted by: vvidhya@veltech.edu.in Receiver: vvidhya.veltec@analysis.arkund.com	 28
SA	15_AGD_Project Phase1 Report.pdf Document 15_AGD_Project Phase1 Report.pdf (D121546970)	 1
SA	Final_ProjectReport-COE16B004.pdf Document Final_ProjectReport-COE16B004.pdf (D73360351)	 2
SA	P1_BrainTumorSegmentation_AnatomicalContextualInformation_MDPI_Diagnostic_V3.pdf Document P1_BrainTumorSegmentation_AnatomicalContextualInformation_MDPI_Diagnostic_V3.pdf (D106225376)	 1
SA	Vel Tech Rangarajan Dr.Sagunthala R&D Inst. of S&T / IFS-211879_R1.pdf Document IFS-211879_R1.pdf (D111546457) Submitted by: ksambathkumar@veltech.edu.in Receiver: ksambathkumar.veltec@analysis.arkund.com	 1
SA	Sunayana Paper.docx Document Sunayana Paper.docx (D111665011)	 1
SA	Mini Project Report.docx Document Mini Project Report.docx (D151031678)	 1

Entire Document

OPTIMIZED U-NET ARCHITECTURE FOR BRAIN TUMOR SEGMENTATION Minor
project report submitted
in partial fulfillment of the requirement
for award of the degree of Bachelor of Technology in Computer Science &
Engineering
By

<https://secure.arkund.com/view/158415577-162011-797006#/overview>

1/24

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
1
2 from sklearn.model_selection import train_test_split
3 import os
4 import nibabel as nib
5 import cv2 as cv
6 import matplotlib.pyplot as plt
7 from keras import backend as K
8 import glob
9 import keras
10 from keras.models import Model, load_model
11 from keras.layers import Input, BatchNormalization, Activation
12 from keras.layers.convolutional import Conv2D, UpSampling2D
13 from keras.layers.pooling import MaxPooling2D
14 from keras.layers import concatenate
15 from keras.callbacks import EarlyStopping, ModelCheckpoint
16 from keras import optimizers
17 import skimage.io as io
18 import skimage.color as color
19 import random as r
20 import math
21 import numpy as np
22 import pandas as pd
23 import matplotlib.pyplot as plt
24 import tensorflow as tf
25 from sklearn.model_selection import train_test_split
26 from nilearn import plotting
27
28 Flair= nib.load('../input/brats20-dataset-training-validation/BraTS2020_TrainingData/
    MICCAI-BraTS2020_TrainingData/BraTS20_Training_008/BraTS20_Training_008_flair.nii')
29
30 Seg= nib.load('../input/brats20-dataset-training-validation/BraTS2020_TrainingData/
    MICCAI-BraTS2020_TrainingData/BraTS20_Training_008/BraTS20_Training_008_seg.nii')
31
32 T1= nib.load('../input/brats20-dataset-training-validation/BraTS2020_TrainingData/
    MICCAI-BraTS2020_TrainingData/BraTS20_Training_008/BraTS20_Training_008_t1.nii')
```

```

33
34 T1ce= nib.load( '../input/brats20-dataset-training-validation/BraTS2020-TrainingData/
MICCAI.BraTS2020-TrainingData/BraTS20-Training_008/BraTS20-Training_008-t1ce.nii')
35
36 T2= nib.load( '../input/brats20-dataset-training-validation/BraTS2020-TrainingData/
MICCAI.BraTS2020-TrainingData/BraTS20-Training_008/BraTS20-Training_008-t2.nii')
37
38 plotting.plot_anat(Flair,cut_coords=[-100, 100, 100],title="Flair",draw_cross =False)
39
40 plotting.plot_anat(Seg,cut_coords=[-100, 100, 100],title="Seg",draw_cross =False)
41
42 plotting.plot_anat(T1,cut_coords=[-100, 100, 100],title="T1",draw_cross =False)
43
44 plotting.plot_anat(T1ce,cut_coords=[-100, 100, 100],title="T1ce",draw_cross =False)
45
46 plotting.plot_anat(T2,cut_coords=[-100, 100, 100],title="T2",draw_cross =False)
47
48 Path= '../input/brats20-dataset-training-validation/BraTS2020-TrainingData/
MICCAI.BraTS2020-TrainingData'
49 p=os.listdir(Path)
50 Input_Data= []
51 def Data_Preprocessing(modalities_dir):
52
53     all_modalities = []
54     for modality in modalities_dir:
55         nifti_file = nib.load(modality)
56         brain_numpy = np.asarray(nifti_file.dataobj)
57         all_modalities.append(brain_numpy)
58     brain_affine = nifti_file.affine
59     all_modalities = np.array(all_modalities)
60     all_modalities = np.rint(all_modalities).astype(np.int16)
61     all_modalities = all_modalities[:, :, :, :]
62     all_modalities = np.transpose(all_modalities)
63     return all_modalities
64
65 for i in p[:20]:
66     brain_dir = os.path.normpath(Path+'/'+i)
67     flair = glob.glob(os.path.join(brain_dir, '*_flair*.nii'))
68     t1 = glob.glob(os.path.join(brain_dir, '*_t1*.nii'))
69     t1ce = glob.glob(os.path.join(brain_dir, '*_t1ce*.nii'))
70     t2 = glob.glob(os.path.join(brain_dir, '*_t2*.nii'))
71     gt = glob.glob(os.path.join(brain_dir, '*_seg*.nii'))
72     modalities_dir = [flair[0], t1[0], t1ce[0], t2[0], gt[0]]
73     P_Data = Data_Preprocessing(modalities_dir)
74     Input_Data.append(P_Data)
75
76 fig = plt.figure(figsize=(5,5))
77 immmg = Input_Data[1][120,:,:,3]
78 imgplot = plt.imshow(immmg)
79 plt.show()

```

```

80
81 def Data_Concatenate (Input_Data):
82
83     counter=0
84     Output= []
85     for i in range(5):
86         print('$')
87         c=0
88         counter=0
89         for ii in range(len(Input_Data)):
90             if (counter != len(Input_Data)):
91                 a= Input_Data[counter][:,:, :, i]
92                 #print('a={}'.format(a.shape))
93                 b= Input_Data[counter+1][:,:, :, i]
94                 #print('b={}'.format(b.shape))
95                 if (counter==0):
96                     c= np.concatenate((a, b), axis=0)
97                     print('c1={}'.format(c.shape))
98                     counter= counter+2
99                 else:
100                     c1= np.concatenate((a, b), axis=0)
101                     c= np.concatenate((c, c1), axis=0)
102                     print('c2={}'.format(c.shape))
103                     counter= counter+2
104             c= c[:,:,: , np.newaxis]
105             Output.append(c)
106     return Output
107
108 InData= Data_Concatenate (Input_Data)
109
110 AIO= concatenate (InData , axis=3)
111 AIO=np.array (AIO, dtype='float32')
112 TR=np.array (AIO[:,:,: ,1], dtype='float32')
113 TRL=np.array (AIO[:,:,: ,4], dtype='float32')
114
115 X_train , X_test , Y_train , Y_test = train_test_split(TR, TRL, test_size=0.15, random_state=32)
116 AIO=TRL=0
117
118 def Convolution(input_tensor , filters):
119
120     x = Conv2D(filters=filters ,kernel_size=(3, 3),padding = 'same',strides=(1, 1))(input_tensor)
121     x = BatchNormalization()(x)
122     x = Activation('relu')(x)
123     return x
124
125 def model(input_shape):
126
127     inputs = Input((input_shape))
128
129     conv_1 = Convolution(inputs ,32)

```

```

130     maxp_1 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_1)
131
132     conv_2 = Convolution(maxp_1,64)
133     maxp_2 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_2)
134
135     conv_3 = Convolution(maxp_2,128)
136     maxp_3 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_3)
137
138     conv_4 = Convolution(maxp_3,256)
139     maxp_4 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'same') (conv_4)
140
141     conv_5 = Convolution(maxp_4,512)
142     upsample_6 = UpSampling2D((2, 2)) (conv_5)
143
144     conv_6 = Convolution(upsample_6,256)
145     upsample_7 = UpSampling2D((2, 2)) (conv_6)
146
147     upsample_7 = concatenate([upsample_7, conv_3])
148
149     conv_7 = Convolution(upsample_7,128)
150     upsample_8 = UpSampling2D((2, 2)) (conv_7)
151
152     conv_8 = Convolution(upsample_8,64)
153     upsample_9 = UpSampling2D((2, 2)) (conv_8)
154
155     upsample_9 = concatenate([upsample_9, conv_1])
156
157     conv_9 = Convolution(upsample_9,32)
158     outputs = Conv2D(1, (1, 1), activation='sigmoid') (conv_9)
159
160     model = Model(inputs=[inputs], outputs=[outputs])
161
162     return model
163
164 model = model(input_shape = (240,240,1))
165 model.summary()
166
167 def dice_coef(y_true, y_pred, smooth=1.0):
168
169     y_true_f = K.flatten(y_true)
170     y_pred_f = K.flatten(y_pred)
171     intersection = K.sum(y_true_f * y_pred_f)
172     return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
173
174 def precision(y_true, y_pred):
175
176     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
177     predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
178     precision = true_positives / (predicted_positives + K.epsilon())
179     return precision

```

```

180
181 def sensitivity(y_true, y_pred):
182
183     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
184     possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
185     return true_positives / (possible_positives + K.epsilon())
186
187 def specificity(y_true, y_pred):
188
189     true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
190     possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
191     return true_negatives / (possible_negatives + K.epsilon())
192
193 Adam=optimizers.Adam(learning_rate=0.001)
194 model.compile(optimizer=Adam, loss='binary_crossentropy', metrics=['accuracy',dice_coef,precision,
195     sensitivity, specificity])
196
197 history = model.fit(X_train, Y_train, batch_size=32, epochs=40, validation_split=0.20, verbose=1,
198     initial_epoch=0)
199
200 model.evaluate(x=X_train, y=Y_train, batch_size=32, verbose=1, sample_weight=None, steps=None)
201 model.evaluate(x=X_test, y=Y_test, batch_size=32, verbose=1, sample_weight=None, steps=None)
202
203 def Accuracy_Graph(history):
204
205     plt.plot(history.history['accuracy'])
206     plt.plot(history.history['val_accuracy'])
207     #plt.title('Model accuracy')
208     plt.ylabel('Accuracy')
209     plt.xlabel('Epoch')
210     plt.legend(['Train', 'Validation'], loc='upper left')
211     plt.subplots_adjust(top=1.00, bottom=0.0, left=0.0, right=0.95, hspace=0.25,
212         wspace=0.35)
213
214     plt.show()
215
216 def Dice_coefficient_Graph(history):
217
218     plt.plot(history.history['dice_coef'])
219     plt.plot(history.history['val_dice_coef'])
220     #plt.title('Dice Coefficient')
221     plt.ylabel('Dice Coefficient')
222     plt.xlabel('Epoch')
223     plt.legend(['Train', 'Validation'], loc='upper left')
224     plt.subplots_adjust(top=1.00, bottom=0.0, left=0.0, right=0.95, hspace=0.25,
225         wspace=0.35)
226
227     plt.show()
228
229 def Loss_Graph(history):

```

```

228     plt.plot(history.history['loss'])
229     plt.plot(history.history['val_loss'])
230     #plt.title('Model loss')
231     plt.ylabel('Loss')
232     plt.xlabel('Epoch')
233     plt.legend(['Train', 'Validation'], loc='upper left')
234     plt.subplots_adjust(top=1.00, bottom=0.0, left=0.0, right=0.95, hspace=0.25,
235                        wspace=0.35)
236     plt.show()
237
238 Accuracy_Graph(history)
239 Dice_coefficient_Graph(history)
240 Loss_Graph(history)
241
242 model.save('./BraTs2020.h5')
243
244 model.load_weights('./BraTs2020.h5')
245
246 X_train=X_test=Y_train=Y_test=0
247
248 fig = plt.figure(figsize=(5,5))
249 immmg = TR[450,:,:]
250 imgplot = plt.imshow(immmg)
251 plt.show()
252
253 pref_Tumor = model.predict(TR)
254
255 fig = plt.figure(figsize=(5,5))
256 immmg = pref_Tumor[450,:,:]
257 imgplot = plt.imshow(immmg)
258 plt.show()
259
260 plt.figure(figsize=(15,10))
261
262
263 plt.subplot(341)
264 plt.title('Sample 1')
265 plt.axis('off')
266 plt.imshow(np.squeeze(TR[200,:,:]), cmap='gray')
267 plt.imshow(np.squeeze(pref_Tumor[200,:,:]), alpha=0.3, cmap='Reds')
268
269 plt.subplot(342)
270 plt.title('Sample 2')
271 plt.axis('off')
272 plt.imshow(np.squeeze(TR[110,:,:]), cmap='gray')
273 plt.imshow(np.squeeze(pref_Tumor[110,:,:]), alpha=0.3, cmap='Reds')
274
275 plt.subplot(343)
276 plt.title('Sample 3')
277 plt.axis('off')

```

```

278 plt.imshow(np.squeeze(TR[880,:,:]),cmap='gray')
279 plt.imshow(np.squeeze(pref_Tumor[880,:,:]),alpha=0.3,cmap='Reds')
280
281 plt.subplot(344)
282 plt.title('Sample 4')
283 plt.axis('off')
284 plt.imshow(np.squeeze(TR[1011,:,:]),cmap='gray')
285 plt.imshow(np.squeeze(pref_Tumor[1011,:,:]),alpha=0.3,cmap='Reds')
286
287 plt.subplot(345)
288 plt.title('Sample 5')
289 plt.axis('off')
290 plt.imshow(np.squeeze(TR[999,:,:]),cmap='gray')
291 plt.imshow(np.squeeze(pref_Tumor[999,:,:]),alpha=0.3,cmap='Reds')
292
293 plt.subplot(346)
294 plt.title('Sample 6')
295 plt.axis('off')
296 plt.imshow(np.squeeze(TR[1450,:,:]),cmap='gray')
297 plt.imshow(np.squeeze(pref_Tumor[1450,:,:]),alpha=0.3,cmap='Reds')
298
299 plt.subplot(347)
300 plt.title('Sample 7')
301 plt.axis('off')
302 plt.imshow(np.squeeze(TR[1530,:,:]),cmap='gray')
303 plt.imshow(np.squeeze(pref_Tumor[1530,:,:]),alpha=0.3,cmap='Reds')
304
305 plt.subplot(348)
306 plt.title('Sample 8')
307 plt.axis('off')
308 plt.imshow(np.squeeze(TR[29,:,:]),cmap='gray')
309 plt.imshow(np.squeeze(pref_Tumor[29,:,:]),alpha=0.3,cmap='Reds')

```


9.2 Poster Presentation

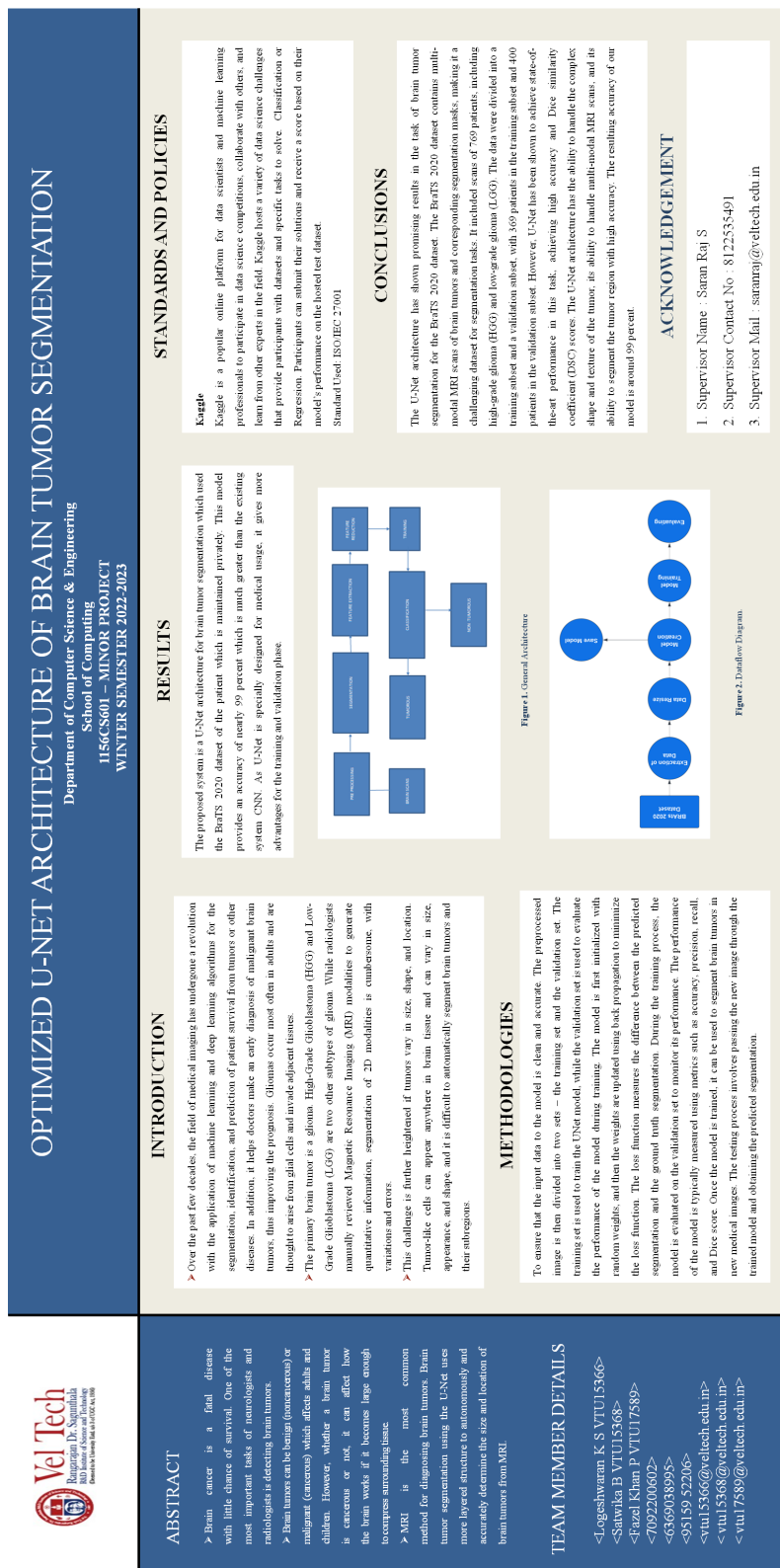


Figure 9.1: Poster Presentation

References

- [1] Tejas Shelatkar, Dr. Urvashi, Mohammad Shorfuzzaman, Abdulmajeed Alsufyani, Kuruva Lakshmana, “Diagnosis of Brain Tumor Using Light Weight Deep Learning Model with Fine-Tuning Approach”, Computational and Mathematical Methods in Medicine, 2022.
- [2] Nahian Siddique, Sidike Paheding, Colin P. Elkin, Vijay Devabhatuni, ”U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications”, Digital Object Identifier, 2021.
- [3] Samia Mushtaq, Apash Roy, Tawseef Ahmed Teli, “A Comparative Study on Various Machine Learning Techniques for Brain Tumor Detection Using MRI”, Global Emerging Innovation Summit, 2021.
- [4] P. Khan, M. F. Kader, S. M. R. Islam et al., “Machine learning and deep learning approaches for brain disease diagnosis: principles and recent advances,” IEEE Access, vol. 9, pp. 37622–37655, 2021.
- [5] Getao Du, Xu Cao, Jimin Liang, Xueli Chen, and Yonghua Zhan, “Medical Image Segmentation based on U-Net: A Review”, Journal of Imaging Science and Technology, 2020.
- [6] A. Rehman, S. Naz, M. I. Razzak, F. Akram, and M. Imran, “A deep learning-based framework for automatic brain tumors classification using transfer learning,” Circuits, Systems, and Signal Processing, vol. 39, no. 2, pp. 757–775, 2020.
- [7] Michal Futrega, Alexandre Milesi, Michal Marcinkiewicz, Pablo Ribalta, “Optimized U-Net for Brain Tumor Segmentation”, NVIDIA, Santa Clara, 2021.

- [8] Ali Ari, Davut Hanbay, “Deep learning based brain tumor classification and detection system”, Turkish Journal of Electrical Engineering and Computer Sciences, 2018.
- [9] T.M. Shahriar Sazzad, K.M. Tanzibul Ahmmed, M.U. Hoque, and M. Rahman, ”Development of Automated Brain Tumor Identification Using MRI Images”, 2nd Int. Conf. Electr. Comput. Commun. Eng. ECCE, 2019 pp. 1-4.
- [10] Aboli Kapadnis, “Brain Tumor Detection using Transfer Learning with AlexNet and CNN”, National College of Ireland, 2021.
- [11] Ramin Ranjbarzadeh, Abbas Bagherian Kasgari, Saeid Jafarzadeh Ghoushchi, ShokofehAnari, Maryam Naseri, Malika Bendechange, “Brain tumor segmentation based on deep learning and an attention mechanism using MRI multi-modalities brain images”, Scientific Reports, 2021.
- [12] Sérgio Pereira, Adriano Pinto, Victor Alves, and Carlos A. Silva, “Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images” , IEEE TRANSACTIONS ON MEDICAL IMAGING, 2016.
- [13] Sourodip Ghosh, Aunkit Chaki, KC Santosh, “Improved U-Net architecture with VGG-16 for brain tumor segmentation”, Physical and Engineering Sciences in Medicine, 2021.
- [14] Li Sun, Songtao Zhang, Hang Chen and Lin Luo¹, “Brain Tumor Segmentation and Survival Prediction Using Multimodal MRI Scans With Deep Learning”, Frontiers in Neuroscience, 2019.
- [15] M Malathi, P Sinthia, “Brain Tumour Segmentation Using Convolutional Neural Network with Tensor Flow”, Asian Pacific Journal of Cancer Prevention, 2019.