UNIVERSITY OF LINCOLN

School of Computer Science

MSc Robotics and Autonomous Systems

Brain Tumor Segmentation of MRI Images using 3D U-Net Model Architecture

RESEARCH PROJECT

CMP9140M-2122

SADANAND VITHAL GIRADDI

Student ID: 25895786

Table of Contents

1. Abstract 6

2. Introduction 7

2.1 Rationale/ benefits 7

2.2 Aim and Objectives 7

2.3 Background 7

2.4 Report Structure 8

3. Literature Review 9

4. Methodology 19

4.1 Project Management 19

4.1.1 Gantt chart 19

4.1.2 PERT chart 20

4.1.3 Risk Assessment 20

4.1.4 Supervisor meetings 21

4.1.5 PRINCE2 Technique. 21

4.2 Software Development 22

4.2.1 SCRUM 22

4.3 Toolsets and Machine Environments 23

4.4 Dataset 25

4.5 Research Methods 26

5. Design and Development 30

5.1 Requirements 30

5.2 Design 30

5.3 Implementation 32

5.3.1 Libraries installation. 32

5.3.2 Pre-processing the Dataset. 32

5.3.3 Feature-Extraction of the Dataset. 34

5.3.4 Custom Data generator. 38

5.3.5 Callback Functions. 39

5.3.6 Compiling the model. 40

5.3.7 Training the model. 40

5.3.8 Model Architecture layers 40

6. Experiments and Evaluation 53

6.1 Results of experiment. 53

6.2 Analysis of results. 80

7. Discussions and Reflective Analysis 82
8. Conclusion 83
9. References and Appendices 84
9.1 References 84
9.2 Appendices 86

SL No	List of Figures	Page Number
1	Figure 3.1: The representation of Glioma with respect to its sub-regions	9
2	Figure 3.2 The Distribution of data for the last 10 years of the BraTS Challenge.	10
3	Figure 3.3 The algorithms performed by the authors (Menze et al., 2015)	11
4	Figure 3.4 U-Net model architecture with 64 filters having 3*3 convolution	12
5	Figure 3.5 Dice similarity score generated for the U-Net model with 64 filters	12
6	Figure 3.6 The U-Net architecture with 32*32 pixels	13
7	Figure 3.7 The Segmentation result of light microscopic images	14
8	Figure 3.8 PSPNet architecture	14
9	Figure 3.9 the modified Xception model by adding more layers	15
10	Figure 3.10 (a) The modalities of images provided in the BraTS 2019 dataset	16
11	Figure 4.1 The final Gantt chart for the project	19
12	Figure 4.2 The PERT chart for the Brain tumor segmentation project.	20
13	Figure 4.3 The complete cycle of Agile Scrum Framework.	22
14	Figure 4.4 The training dataset containing 1251 patient record data	26
15	Figure 4.5 The U-NET Model architecture.	28
16	Figure 5.1 The overall Machine Learning pipeline	31
17	Figure 5.2 Installing the Dependencies for the research project.	32
18	Figure 5.3 Loading the images from the training dataset for all the classes.	33
19	Figure 5.4 Visualizing the images in the classes of the dataset.	33
20	Figure 5.5 plotting of t1 image.	34
21	Figure 5.6 plotting of mask image.	35
22	Figure 5.7 plotting the different annotations of flair images	36
23	Figure 5.8 The formulas of evaluation methods chosen for the research project.	37
24	Figure 5.9 Distribution of data in the dataset	39
25	Figure 5.10 Adding Callbacks	39
26	Figure 5.11 : Compiling the model	40
27	Figure 5.12 Training the model	40
28	Figure 5.13 3D U-Net modified architectures 4 filters with Kernel size of 3*3*3	41
29	Figure 5.14 3D U-Net modified architectures 8 filters with Kernel size of 3*3*3	43
30	Figure 5.15 3D U-Net modified architectures 16 filters with Kernel size of 3*3*3	45
31	Figure 5.16 3D U-Net modified architectures 32 filters with Kernel size of 3*3*3	47
32	Figure 5.17 3D U-Net modified architectures 4 filters with Kernel size of 2*2*2	48
33	Figure 5.18 3D U-Net modified architectures 8 filters with Kernel size of 2*2*2	49
34	Figure 5.19 3D U-Net modified architectures 16 filters with Kernel size of 2*2*2	51
35	Figure 6.1 plotting the graphs for training and validation	53
36	Figure 6.2 Training and validation curves for 16 filters with kernel size of (3, 3, 3).	54
37	Figure 6.3 Training and validation curves for 8 filters with kernel size of (3, 3, 3).	57
38	Figure 6.4 Training and validation curves for 4 filters with kernel size of (3, 3, 3).	59

39	Figure 6.5 Training and validation curves for 32 filters with kernel size of (3, 3, 3).	62
40	Figure 6.6 Training and validation curves for 16 filters with kernel size of (2, 2, 2).	65
41	Figure 6.7 Training and validation curves for 8 filters with kernel size of (2, 2, 2).	68
42	Figure 6.8 Training and validation curves for 4 filters with kernel size of (2, 2, 2).	71
43	Figure 6.9 Evaluation scores generated for the Testing set and the Validation set	80
44	Figure 9.1 Flowchart for the source-code.	86

SL No	List of Tables	Page Number
1	Table 1: Comparison of Different Integrated Development Environments.	21
2	Table 2: 3D U-Net modification (started with 4 number of filters and (Kernel sizes-3*3*3))	40
3	Table 3: 3D U-Net modification (started with 8 number of filters and (Kernel sizes-3*3*3))	41
4	Table 4: 3D U-Net modification (started with 16 number of filters and (Kernel sizes-3*3*3))	43
5	Table 5: 3D U-Net modification (started with 32 number of filters and (Kernel sizes-3*3*3))	45
6	Table 6: 3D U-Net modification (started with 4 number of filters and (Kernel sizes-2*2*2))	47
7	Table 7: 3D U-Net modification (started with 8 number of filters and (Kernel sizes-2*2*2))	48
8	Table 8: 3D U-Net modification (started with 16 number of filters and (Kernel sizes-2*2*2))	50

Acknowledgements

I would like to convey my special thanks and sincere gratitude to my supervisor Dr. Lei Zhang, for his wonderful support and the guidance that he has given me in the supervisor meetings and via email to accomplish this research project. I would also like to thank him for allowing me to learn the research skills for my interest in machine learning through this project. I would like to thank the University of Lincoln for providing great lab systems. I would also like to thank my father, mother, and brother who encouraged me to complete the project in time. Finally, I thank the almighty god who showed me the right vision and helped me to clear the obstacles that I have faced during these years and many more to face in my upcoming life to get succeeded in my life.

1. Abstract

In this research project, Brain tumor segmentation uses the deep learning method for segmenting brain tumors from MRI images. Glioma segmentation is more focused on the current research for detecting the presence of tumors in the brain tumor sub-regions. As manual segmentation is difficult and takes more time. Automatic Segmentation helps in saving time for the segmentation of brain tumors in MRI images. The major challenge in brain tumor segmentation is determining the tumor extent. Various state-of-art algorithms like DeepMedic, PSPNet, and V-Net are used on the segmentation of brain tumors on different datasets of the BraTS challenge. This research work focuses on the 3D U-Net model architecture that is going to be implemented on brain tumor segmentation of MRI images of the BraTS 2021 dataset for further research. The different modifications have been done by tuning the hyperparameters to the 3D U-Net model architectures for enhancing the performance of the model and comparing the evaluation results. Based on the evaluation methods like Loss, Accuracy, MeanIoU, Precision, Dice Similarity of Co-efficient, Specificity, and Sensitivity, Generating the dice scores for the tumor sub-regions like the Necrotic core region, Edema region, and enhancing tumor region the results are evaluated. The implementation of code is done on the Python Jupyter notebook script using the TensorFlow frameworks and the Keras modules.

2. Introduction

2.1 Rationale/ benefits

Machine learning consists of a set of techniques that helps machines to do better at doing things by acquiring the skills with or without the help of a human (as stated by Andrew NG). Involves the machine to learn things like the humans do. And plays a major role in the fields like Medical, Banking, business to science, and technology. Deep learning is the subset of machine learning that uses algorithms that are inspired by the neurons in the human brain. Deep learning is a neural network where the network consists of three or more layers and involves learning through layers which enables the computer to perform the tasks to refine the performance of the data. Deep learning helps in handling a large number of datasets in terms of analyzing and providing the output accordingly.

2.2 Aim and Objectives

The major benefit of doing this project is to research the previous works which are done for segmentation of brain tumor sub-regions using various state-of-art methods. And after knowing the limitations of the algorithms used on the BraTS datasets and further trying to improve the algorithm by implementing (our proposed model i.e. 3d U-Net) by modifying the architecture in the segmentation of brain tumors.

Project Aim:

The main aim is to use our proposed model i.e. 3D U-Net to provide better brain tumor segmentation performance and evaluation for the BraTS 2021 dataset for 3D MRI image processing in medical applications.

Project Objectives:

The major objective of this project is to improve the segmentation of the brain tumor sub-regions: The Enhancing tumor region, the tumor core region, and the whole tumor region for the BraTS 2021 Dataset which is taken from the Kaggle competition. And for further research, the 3D U-Net implementation will be carried out by modifying the model architectures based on U-Net architecture for segmenting the brain tumor regions by enhancing the performance and evaluations that will be considered for the project.

Hypothesis: A research study is designed to hypothesize on providing good segmentation of dice similarity scores for the tumor sub-regions of the Necrotic core region, Edema core region, and enhancing tumor core region.

2.3 Background

Brain tumors are divided into two categories. They are 1) Primary tumor type and 2) Secondary tumor type. In the primary tumor type, the tumor cells are originated from the brain, and in the secondary tumor type, the tumor cells are originated from the other organs of the body, and later they are metastasized into the brain. Gliomas are the primary tumor type. And in metastasized situations, in any area of the body, they develop malignant cells which are cancerous in nature and can then spread and cause to the brain (Işın, Direkoğlu and Şah, 2016). The Brain tumor segmentation of MICCAI challenge aims for providing the accurate segmentation of tumors in the MRI images (Menze et al., 2015) and (Bakas et al., 2019). The modern methods are used for segmenting brain tumors are examined using the cutting-edge approaches in the Brain tumor segmentation of BraTS Challenge.

Image segmentation plays a major role and critical step in the segmentation of MRI images used for extracting the tumors in the brain tumor sub-regions (Dong et al., 2017).

Medical Image processing helps to explore the 3D image datasets of the human body, which are commonly obtained using magnetic Resonance Imaging (MRI), where MRI is used as a diagnostic tool

for analysis of Brain tumor BraTS data, also referred to as widely used Imaging techniques and Computed Tomography (CT) scans for diagnosing the patient data for the research purposes (Pereira et al., 2016).

Glioma is the primary tumor that originated in the glial cells of the brain and is classified as low-grade and high-grade glioma. High-grade gliomas are considered as the most aggressive form of disease having a survival rate of two to three years which requires immediate treatment and for low-grade gliomas are less aggressive and slower growing low-grade variants when it is compared with high-grade gliomas, and the aggressive treatment is delayed due to low-grade astrocytomas and life expectancy is several years. Neuroimaging protocols were applied for both groups to evaluate the disease's progression before and after the treatment by applying the success of chosen strategy for treatment. Gliomas research is more prevalent in today's scenario case. The importance of early treatment plays a major role in diagnosing using CT scans, MRI scans are used to determine the size and location of brain tumors (Menze et al., 2015).

The accurate and automatic segmentation of brain tumor sub-regions with respect to their boundaries in Magnetic Resonance Images (MRI) is more important in medical applications such as monitoring the tumor growth of the brain and generating the radiotherapy maps. Earlier radiologists used to do manual detection for MRI scans or CT scans for tracing the tumor sub-regions of the patients. However, this process is time-consuming, tedious, and becomes impractical when radiologists are dealing with numerous patients. This has been highlighted to go for automatic segmentation for detecting the exact location of gliomas present in the tumor regions where this can be achieved using the deep learning of various state-of-the-art tumor segmentation algorithm methods for increasing the performance of detection in the tumor sub-regions (Baid et al., 2021).

This Research project discusses Brain Tumor Segmentation and evaluation of the BraTS 2021 dataset. A 3D U-Net network is used to classify and detect the brain tumor in the brain tumor sub-regions. The major challenging part of this project is to detect the exact location of gliomas in the brain tumor sub-regions by implementing the 3D U-Net algorithms and noting down the performance of an algorithm by modifying the architectures.

2.4 Report Structure

In summary, my contribution to the project is:

- Acquiring a suitably updated dataset for Brain tumor segmentation which should be publicly available and the BraTS 2021 dataset has been taken from the Kaggle competition.
- Performing the background research for the literature review on the brain tumor segmentation of various state-of-the-art methods for knowing the evaluation methods and scope of the proposed algorithm.
- Proposing the deep learning method 3D U-Net architecture based on the original U-Net architecture containing the encoder having the contraction path and the decoder having the expansion path for the brain tumor segmentation for the BraTS 2021 dataset.
- In this work, we are modifying the 3D U-Net architecture by giving different numbers of filters with different kernel sizes and adding dropout layers for the model.
- Detailed analysis of the 3D U-Net model based on design choices and model variants has been provided.
- Outleighing the results by comparing the performance and evaluations scores obtained using various modifications done on 3D U-Net networks and also comparing the results with various state-of-art methods used for segmenting the Brain tumor of the BraTS 2021 dataset comparing the promising results with tumor sub-regions.

3. Literature Review

In this literature review, the literature is majorly focused on brain tumor segmentation algorithms that are used using various deep learning state-of-the-art methods for segmentation and detection of tumor sub-regions. And in return, it helps us to compare our proposed model i.e 3D U-Net model, in terms of evaluation and performance with the other segmentation algorithms of model architectures.

The Benchmark of the BraTS 2021 paper, the authors (Baid et al., 2021) describes the BraTS 2021 challenge consists of the multi-parametric Magnetic Resonance Imaging (mpMRI) scans of gliomas in the form of 3D datasets for Brain tumor segmentation and the radiogenomic classification which was jointly organized and provided the dataset by the Radiological Society of North America (RSNA) - American Society of Neuroradiology (ASNR) – Medical Image computing and computer-assisted Interventions (MICCAI). And this challenge is majorly focused on the glioma segmentation algorithms as part of the challenge. Gliomas are the most primary malignancies tissues of the nervous system with various degrees of prognosis and aggressiveness. The BraTS 2021 targets the evaluations of different algorithms for detecting the same tumor characterizations with the baseline mpMRI data containing 2040 patients. The ground truth annotations of tumor sub-regions were approved by the neuroradiologists and methylation status using MGMT promoter was experimented with in the laboratory using the surgical brain tumor specimen. The annotated tumor sub-regions of segmenting classes are based on VASARI features comprising of label 3- No tumor, label 1- necrotic tumor core (NCR), label 2- invaded tissue or the peritumoral edematous tissue, label 4- Gd- enhancing tumor of the brain. The structured mpMRI volumes of BraTS were considered as T1, T2, T1GD, and T2-FLAIR.

This challenge focuses on two tasks they are: 1) The brain tumor segmentation of tumor sub-regions. And 2) The classification of the tumor methylation status using MGMT (O-methylguanine DNA methyltransferase) promoter. And the overall performance of the BraTS 2021 challenge is evaluated through the Sage Bionetworks Synapse platform and Kaggle. The evaluation is done based on the Accuracy, loss, dice similarity co-efficient score, Sensitivity, Precision, and Specificity.

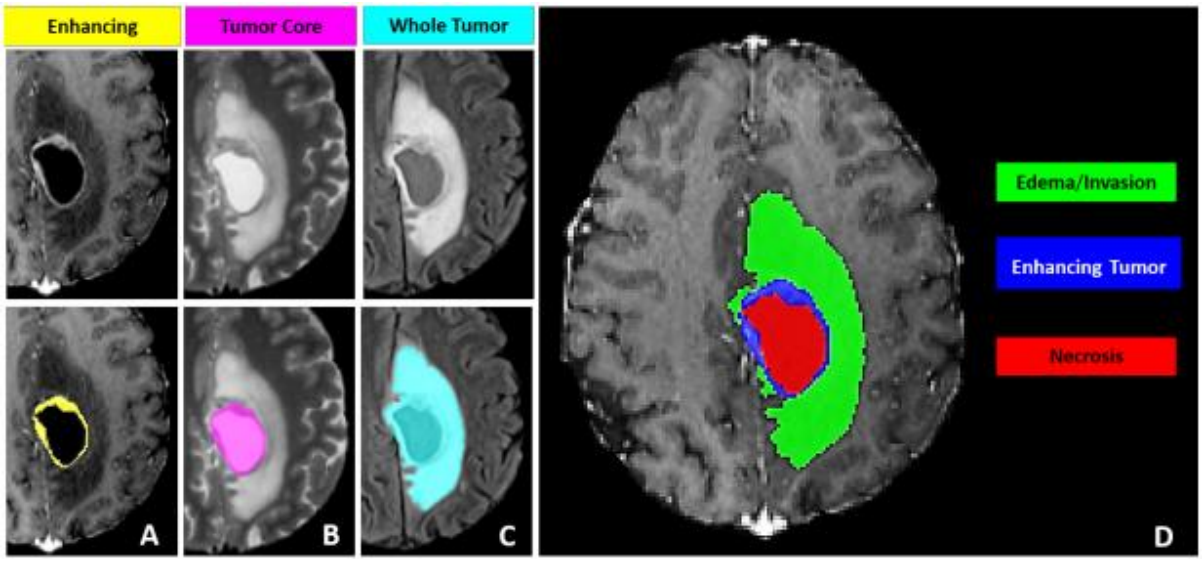


Figure 3.1: The representation of Glioma with respect to its sub-regions where (Enhancing tumor- yellow, Tumor core- Pink, Necrotic core region- Red, and the whole Tumor i.e Edema – light blue and green) that is considered for BraTS 2021 Challenge.

Year	Total Data	Training Data	Validation Data	Testing Data	Tasks	Timepoint
2012	50	35	NA	15	Segmentation	Pre-operative
2013	60	35	NA	25	Segmentation	Pre-operative
2014	238	200	NA	38	Segmentation	Longitudinal
2015	253	200	NA	53	Segmentation Disease progression	Longitudinal
2016	391	200	NA	191	Segmentation Disease progression	Longitudinal
2017	477	285	46	146	Segmentation Survival prediction	Pre-operative
2018	542	285	66	191	Segmentation Survival prediction	Pre-operative
2019	626	335	125	166	Segmentation Survival prediction	Pre-operative
2020	660	369	125	166	Segmentation Survival prediction	Pre-operative
2021	2040	1251	219	570	Segmentation MGMT classification	Pre-operative

Figure 3.2: The Distribution of data across training, validation, and testing for the last 10 years of the BraTS Challenge.

The major limitation is that it needs to be further considered for the segmentation of tumors at each case should be refined by the single annotator by following an iterative process with the group of expert approvers. And some of the MRI data's are having some abnormalities of gliomas that are included in the BraTS challenge.

The Brain tumor image segmentation using multimodal for BraTS paper, the authors (Menze et al., 2015) reported the set-up and evaluated the results of brain tumor image segmentation using multimodal organized with MICCAI 2012 and 2013 conference. And describes the current state-of-art methods used in automated segmentation. The set of 65 multi-contrast magnetic resonance scans of high and low-grade glioma patients was manually annotated by up to four routers and was segmented using the twenty state-of-the-art methods. The quantitative evaluations considerably revealed disagreement for segmentation of the various tumor regions with dice scores of 74% to 85% which was done by the human raters. Some algorithms worked well for different sub-regions when it is compared with the human raters. Unfortunately, none of the algorithms are ranked at the top for the segmentation of sub-regions. The manually annotated data and BraTS image are publicly available online for the evaluation system. The segmentation of tumor sub-regions is manually annotated by the human raters showed significant variations between the structures considering in terms of size, and localization by prohibiting the strong use of priors on shape and location which are the important components for segmentation. In generative probabilistic methods, the models with explicit anatomy and appearance were combined for obtaining the automated segmentation and exhibit good generalizations to unsee the images. In discriminative approach methods, the relationship between the image intensities and the segmentation labels is directly learned. And fusing several algorithms using the majority vote yielded the segmentations which ranked above all the algorithms. The template-moderated approach method has generated the highest accuracy among the algorithm used in this paper. This study shows that the performance has been significantly boosted by fusing different segmenters.

Algorithm	MRI modalities	Approach	Perform. score	Tumor type	trainining/testing (tt/cv)
Fletcher 2001	T_1 T_2 PD	Fuzzy clustering w/ image retrieval	Match (53-91%)	na	2/4 tt
Kaus 2001	T_1	Template-moderated classification	Accuracy (95%)	LG, M	10/10 tt
Ho 2002	T_1 T_1c	Level-sets w/ region competition	Jaccard (85-93%)	G, M	na/5 tt
Prastawa 2004	T_2	Generative model w/ outlier detection	Jaccard (59-89%)	G, M	na/3 tt
Corso 2008	T_1 T_1c T_2 FLAIR	Weighted aggregation	Jaccard (62-69%)	HG	10/10 tt
Verma 2008	T_1 T_1c T_2 FLAIR DTI	SVM	Accuracy (34-93%)	HG	14/14 cv
Wels 2008	T_1 T_{1c} T_2	Discriminative model w/ CRF	Jaccard (78%)	G	6/6 cv
Cobzas 2009	T_{1c} FLAIR	Level-set w/ CRF	Jaccard (50-75%)	G	6/6 tt
Wang 2009	T_1	Fluid vector flow	Tanimoto (60%)	na	0/10 tt
Menze 2010	T_1 T_1c T_2 FLAIR	Generative model w/ lesion class	Dice (40-70%)	G	25/25 cv
Bauer 2011	T_1 T_{1c} T_2 FLAIR	Hierarchical SVM w/ CRF	Dice (77-84%)	G	10/10 cv

Figure 3.3: The algorithms performed by the authors (Menze et al., 2015) using different ways of approaches and methods used for generating the performance score based on the type of tumor they have chosen by separating the dataset into training, testing, and cross-validation.

The major limitation of this paper is none of the algorithms ranked at the top in the competition, and for further considerations, the homogenized datasets should be provided based on image volumes which are registered and interpolated with standard resolution and normalized with the image intensity in order to do the comparisons of segmentation methods over the issue of pre-processing. And then for further work, the methodologies using different approaches of methods should be implemented.

The Automatic brain tumor segmentation and detection are done using the U-Net architecture which is based on fully convolutional networks paper, the authors (Dong et al., 2017) describe the major challenge in brain tumor segmentation planning and the quantitative evaluation based on determining the tumor extent. The non-invasive magnetic resonance imaging method has become a primary diagnostic tool without ionizing the radiation for brain tumors. Manual segmentation takes more time for extracting the 3D MRI Volumes and the performance majorly relies on the experience of an operator. A reliable fully automated segmentation method for the segmentation of tumors is necessary for measuring the tumor extent. For this study, they proposed a fully automated method for the segmentation of brain tumors using U-Net which is based on the deep convolutional network. And later they evaluated their method on BraTS 2015 datasets of multimodal brain tumor segmentation containing 54 low-grade and 220 high-grade tumor cases. And cross-validation is used with the method for obtaining good segmentation efficiency. The sequences of four scans of an MRI were performed using T1-weighted (T1), T1 with enhancing contrast (T1c), T2-weighted, and Flair of each patient. All four scans were registered for the patient having a fine resolution of 1*1*1 mm3 and an image size of 240*240*155 by applying the normalization for each scan and subtracting the mean of each scan sequence and dividing by its standard deviation. And manual segmentation of all four labels was used as ground truth in both training and validation. Flair images are used for segmenting the images. The main reason for building this model architecture is to improve the performance of the tumor and

compared it with the other models. L1, L2, and Dropout layers are used in order to overcome overfitting issues. Overall their proposed method can yield good efficiency and perform good segmentation on comparing with the manual ground truth labels. The proposed model can achieve comparable results for segmenting brain tumor sub-regions.

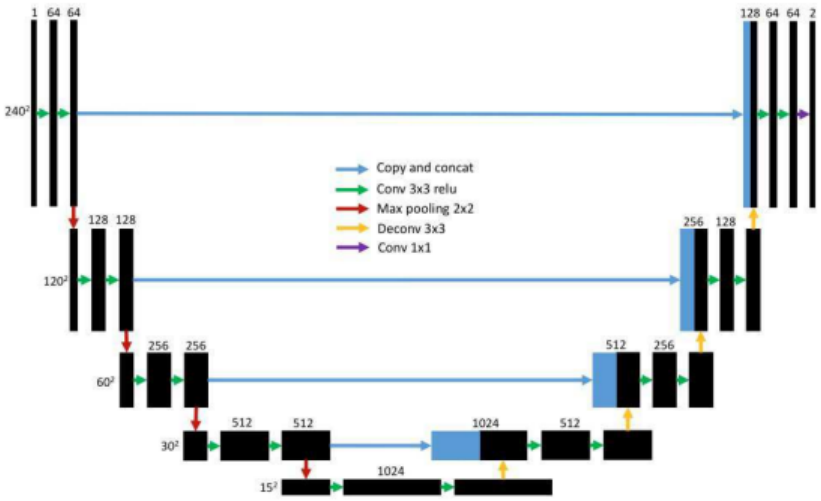


Fig. 1. Our developed U-Net architecture.

Table 2. Parameters setting for the developed U-Net.

Parameters	Value
Number of convolutional blocks	[4, 5, 6]
Number of deconvolutional blocks	[4, 5, 6]
Regularization	L1, L2, Dropout

*Figure 3.4: U-Net model architecture with 64 filters having 3*3 convolution and Deconvolution and 2*2 max pooling.*

			DSC		
Method	**Data**	**Grade**	**Complete**	**Core**	**Enhancing**
Proposed	Cross-Validation on BRATS 2015 Training Datasets	HGG	0.88	0.87	0.81
		LGG	0.84	0.85	0.00
		Combined	0.86	**0.86**	0.65
Pereira16	BRATS 2013 Leaderboard	HGG	0.88	0.76	0.73
		LGG	0.65	0.53	0.00
		Combined	0.84	0.72	0.62
	BRATS 2013 Challenge	HGG	0.88	0.83	0.77
	BRATS 2015 Challenge	Combined	0.79	0.65	0.75
Havaei16	BRATS 2013 Training	Combined	0.88	0.79	0.73
	BRATS 2013 Challenge	Combined	0.88	0.79	0.73
	BRATS 2013 Leaderboard	Combined	0.84	0.71	0.57
	BRATS 2015 Challenge	Combined	0.79	0.58	0.69
Kamnitsas17	BRATS 2015 Training	Combined	**0.90**	0.76	**0.73**
	BRATS 2015 Challenge	Combined	0.85	0.67	0.63

Figure 3.5: Dice similarity score generated for the prosed U-Net model with 64 filters and the evaluation of dice similarity co-efficient scores for grades are HGG- 0.88, LGG- 0.84, and Combined – 0.86.

The major limitation of this paper is they evaluated the segmentation model with cross-validation which may provide an unbiased prediction. And tuning hyperparameters should be done carefully in order to fit the model with good layers. After applying the L1, L2, and Dropout layers they did not get a significant performance based on the proposed model. They are less successful in segmenting the enhanced tumor regions of low-grade glioma.

The convolutional networks used for the biomedical image segmentation paper, the authors (Ronneberger, Fischer and Brox, 2015) describe that they segmented the biomedical image based on U-Net architecture. They presented the training strategy in which they are used for data augmentation of annotated samples using the network. The main architecture consists of a contraction path for capturing the context and an expansion path for enabling precise localization. Their network performed best for very few images and outperforms a sliding window convolution network method on the ISBI challenge. And they have trained the same network on transmitted light microscopy images and won the ISBI challenge conducted in 2015 for cell tracking. The image segmentation of size 512*512 took less time when it is run on the GPU. They built the fully convolution network by modifying the architectural network for working with few training images and yielding precise segmentations. The one modification they have done for this architecture is that they have used a large number of feature channels in the upsampling part for propagating the context to high resolutions. And the other modification done is that they did not use the fully convoluted layers instead they validated each part of convolution, as the segmentation map contains only pixels that help for giving the inputs. For predicting the pixels the context information which is found missing is extrapolated with the help of mirroring the input image. They have resulted the segmentation in the form of EM stacks and won the challenge for 2D transmitted light datasets.

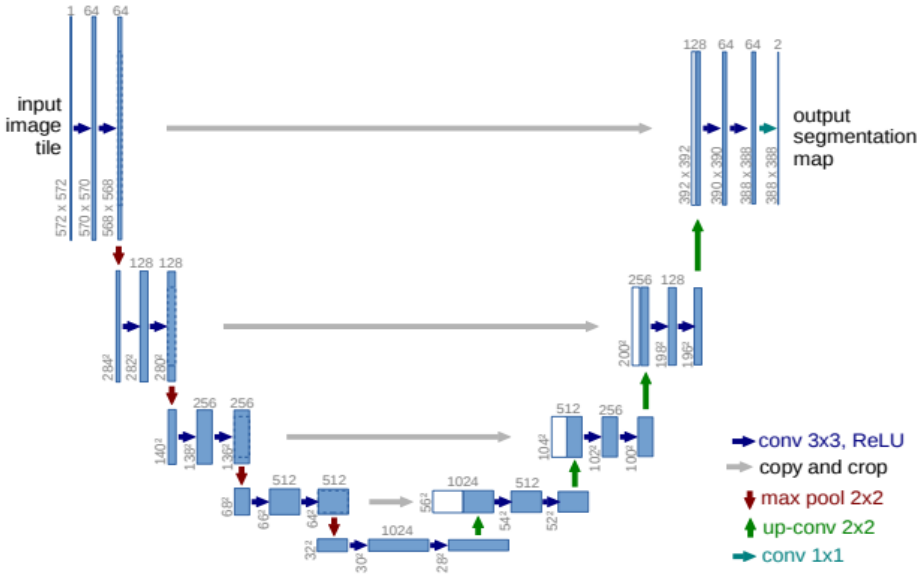


*Figure 3.6: The U-Net architecture with 32*32 pixels where the arrow denotes various operations, the blue box represents the multi-channels feature map, and the white box represents the copied feature maps.*

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

Figure 3.7: The Segmentation result of light microscopic images recorded based on the IOU score for the ISBI cell tracking challenge 2015.

The major limitation found in this paper is that segmenting of cells and the tasks for separation of touching objects for the same class was more challenging for proposing the use of weighted loss. And the task was provided with little training datasets. And the prediction of pixels in the border region images are having some missing context information.

The pyramid Scene Parsing (PSP) Network paper, the authors (Zhao et al., 2017) describes the challenges faced for parsing the scene in unrestricted in the diverse and open vocabulary scenes. And also exploited the global context information used in different regions based on the aggregation of context through the pyramid pooling module which is proposed together with the PSP (Pyramid Scene Parsing) Network. This network produces superior prediction over the pixel level using a good framework. This method is tested on various datasets and this method won the ImageNet scene parsing challenge in 2016, cityscapes, and PASCAL VOC 2012 benchmark. The single PSPNet yielded a new record of mIoU accuracy score of 85.4% on the Pascal Voc 2012 benchmark and achieved 80.2% of accuracy on the cityscapes. The major goal of scene parsing is to assign each pixel in the image over a category label. The scene parsing method predicts labels, shape, and location of the element. They have used the pre-trained ResNet architecture based on deeply supervised loss with dilated network strategy. The module uses four bin sizes of 1*1, 2*2, 3*3, and 6*6 for the pooling operation between the average and max pooling.

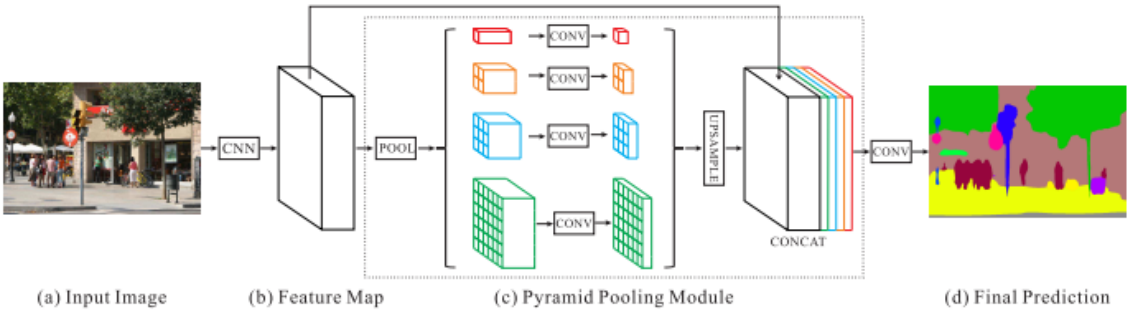


Figure 3.8: PSPNet architecture using the CNN for getting the feature map from the last convolution layer and then the pyramid pooling module is applied for parsing the different sub-region representations which are followed with the Upsampling for the final feature containing both global and local context information and later it is fed into the convolution layer for the final pixel prediction.

The major limitation found in this paper is that scene parsing is more difficult as it is related only to scenes and label variety. Many errors are related to the global context information used for different receptive fields. And the deep network with the appropriate global scene level can improve the scene parsing performance.

The semantic image segmentation paper, the authors (Chen et al., 2018) describes the encoder-decoder structure or the spatial pyramid pooling module used for semantic segmentation tasks in deep neural networks. The networks are used for encoding the multi-scale context information by probing the features with the pooling operations and filters. And later the networks are used for capturing the sharp boundaries of an object by recovering the spatial information. To support this work, they proposed combining both methods. i.e. the proposed model DeepLabv3+ was extended by adding the simple effective decoder module for refining the segmentation over the object boundaries. And for further research work, they explore the Exception model for applying the convolution to both decoder modules and atrous spatial pooling modules by resulting the faster encoder-decoder networks. The proposed model is able to segment the images without doing any post-processing. And demonstrated their evaluation with Cityscapes and PASCAL VOC 2012 datasets by achieving the test generator performance of 82.1% and 89% without doing any post-processing. For the evaluation, they employed the ImageNet-1k method as well as the pretrained ResNet-101 model for extracting the dense feature maps by convolution.

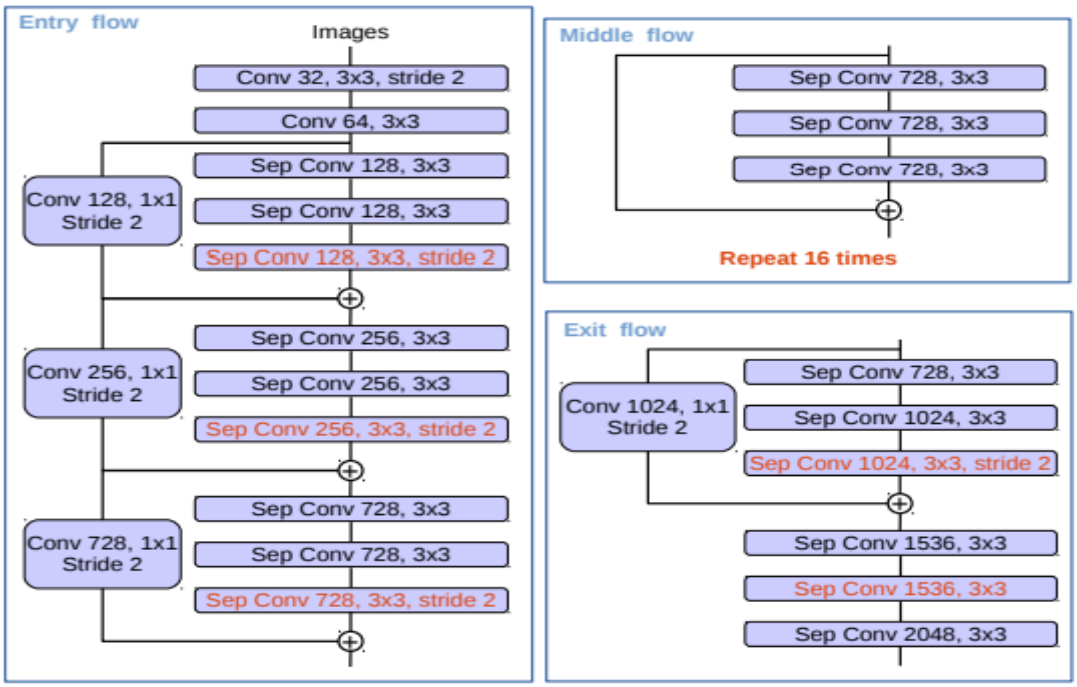
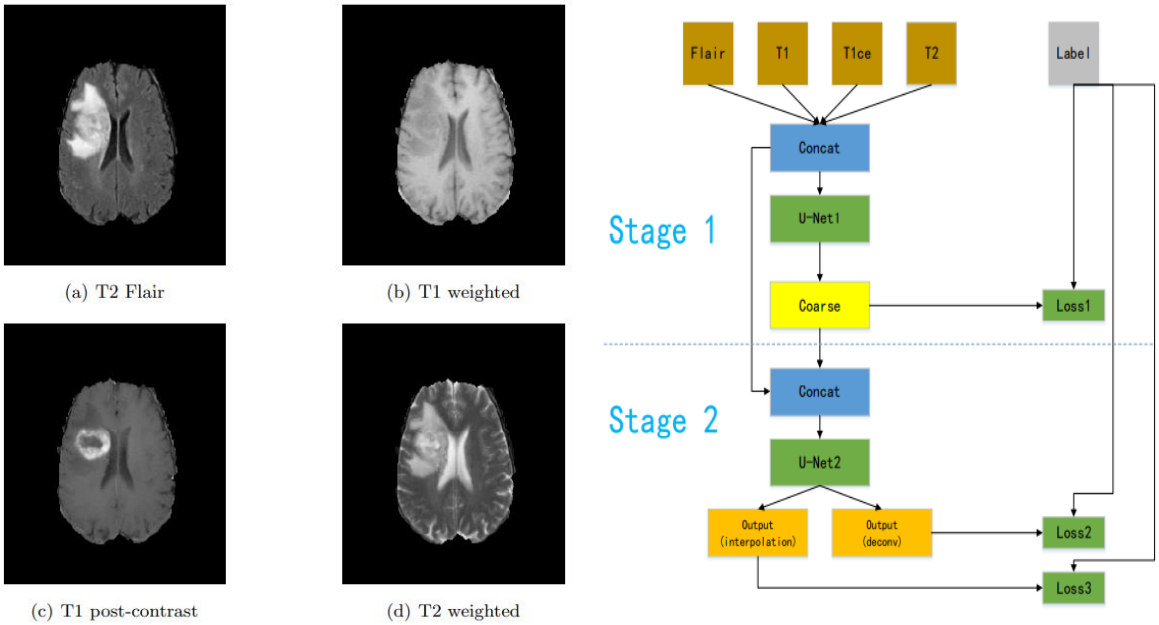


*Figure 3.9: the modified Xception model by adding more layers and max pooling operations are fed with the striding and they added batch normalization and activation function ReLU after each 3*3 depth convolution.*

The major limitation found in this paper is that the model has faced difficulties improvement of object boundaries like segmentation of different images like sofa vs. chair, objects with the rare view, and heavily occluded objects.

In a two-stage cascaded U-Net paper the authors (Jiang et al., 2020) proposed the two-stage cascaded U-Net model for segmenting the substructures of the brain tumors which was trained on the training dataset provided by the multi-modal brain tumor segmentation challenge 2019. And experimented with the results on the testing set by demonstrating the evaluation metrics based on Hausdorff distance and dice scores, they achieved Hausdorff distance (95%) for enhancing tumor of 2.6505, the whole tumor of 4.6180 and tumor core of 4.1307 and also achieved good dice scores for enhancing tumor of 0.8326, whole tumor of 0.8879 and tumor core of 0.8369 respectively. Their approach won 1st place for the segmentation task in BraTS 2019 challenge. The training dataset which was taken from the BraTS 2019 contains 259 cases of HGG (High-grade glioma) and 76 cases of LGG (Low-grade glioma) which as

manually annotated and certified by the radiologists. In their first stage network, they used a variant of U-Net for training the coarse prediction. And in the second stage, they increased the width of the network and used two decoders for boosting the performance and also stage helped them to refine the prediction map by concatenating the layers with the encoder path. The Dice score, Sensitivity, Specificity, and Hausdorff Distance (95%) number of metrics are used for measuring the segmentation performance of algorithms.



(a) (b)

Figure 3.10: (a) The modalities of images provided in the BraTS 2019 dataset. (b) Represent the overview of a two-stage cascaded network.

During pre-processing of the data they applied the intensity normalization for each modality of the patient as the intensity values for MRI are non-standardized and for preventing the overfitting issue they cropped the MRI data from 240*240*155 to 128*128*128 voxels for memory limitations. The model has been trained for 405 iterations of the epoch. And modified the network architecture and tested the model for the BraTS 2019 dataset.

The major limitation found in this paper is some overfitting issues with different models trained on the same datasets and their proposed method requires more computing power for training the models and faced a lot of time taken for training as they are running it on 12 Gb memory.

In the Brain tumor segmentation for identifying the best machine learning algorithm and predicting the overall survival rate paper, the authors (Bakas et al., 2019) study the various state-of-art machine learning methods used for analyzing the MRI images of the BraTS dataset from 2012 to 2018. They focused on the segmentation of gliomas sub-regions of mpMRI scans, assessing the tumor progression, and predicting the survival rate for the scans of the patient. And investigated the machine learning algorithms for each task. And explained the difference of annotation done to the BraTS MRI images. The training dataset has been increased with size and machine learning architectures from year to year challenge of BraTS. The accuracy has been improved by the fusion of segmentation labels of various individual methods. For handling large patient information the development of machine learning approaches are considered.

In the brain tumor segmentation using CNN paper, the authors (Pereira et al., 2016) proposes the CNN with small 3*3 kernels, they used small kernels because of preventing the overfitting of the model and investigated the use of intensity normalization in the pre-processing step and proved to have effective in the data augmentation. They applied N41TK method and intensity normalization method for making the contrast. Normalized the patches to have zero mean and unit variance on each sequence. The proposed network was validated on the BraTS 2013 dataset and obtained 0.88, 0.83, and 0.7 dice similarity scores for the complete, core, and enhancing regions. And also participated on the BraTS 2015 dataset and obtained 0.78, 0.65, and 0.75 dice similarity scores for the complete, core, and enhancing region.

The main advantage found in this paper is they used small kernels to prevent overfitting, and disadvantage found is that data augmentation was not thoroughly explored in DL technique of brain tumor segmentation.

In the brain tumor segmentation using the triplanar method paper, the authors (Sundaresan, Griffanti and Jenkinson, 2021) describe the triplanar ensemble network using U-Net with independent tumor core prediction. They cropped images to 192*192*160 size and extracted 2D slices from the volume. They used 3*3 convolutional kernels in the first layer and uses 5*5 kernels in the next layer. They evaluated their method on MICCAI Validation dataset and they achieved dice scores of 0.77 for both tumor core and the enhancing tumor core and for whole tumor they achieved 0.89 dice score. And achieved 0.81 for enhancing tumor, 0.89 for the whole tumor, and 0.84 on tumor core region of BraTS 2020 dataset.

In the review of MRI images using deep learning methods for brain tumor segmentation paper, the author (Işın, Direkoğlu and Şah, 2016) made reviewed on segmentation methods used in the brain tumor segmentation and automatic segmentation methods achieved state-of-art results and addressed the use of deep learning methods for segmentation their problems. Deep learning methods enabled the processing of large amounts of MRI image data. They used the pre-processing step for removing the noise and skull stripping, after this process they used the image processing techniques for extracting the features like textons, discrete wavelet transforms, and raw intensities. They proposed 3D CNN architecture for the segmentation of multi-modality of 3D patches giving a filter shape of 5*5*5*4. As per architecture they proposed two networks, and they achieved the dice scores for the whole tumor region, core tumor region, and active tumor region is 87%, 77%, and 73%.

In the muniecon thesis paper, the author (Lloyd and Macrae, 2020) implemented the prince2 method of technique in his project for project management. In which each selection of activities should be approved by the supervisor, following the supervisor's instructions carefully, a detailed comparison between the company policies and the prince2 method has been provided in his paper. Using the suitable method, the paper discussed the timelines, costs, financial and non-financial benefits, analysed the risks, doing the background research for the project has been done.

In the scrum development process paper, the authors (Schwaber, 1997) describe how the development process takes place and also stated the philosophy for developing the systems through planning and estimating the project. Scrum assumes the development in the system is unpredictable and defines the scrum as the set of activities that combines the techniques and tools that the development team can build the design of the system, as the activities are inherent to the risk, controlling the management of the process, and says that scrum is commonly used in the development cycle for the process of time pressure, quality, vision, resource, and enhancing the customer requirements.

Summary of Literature work:

From the above literature review, the project has reviewed the literature based on the CNN architectures, various state-of-art methods like DeepMedic, PSPNet, ImageNet, DeepLabv3+, Volumetric segmentation, U-Net used for segmentation of brain tumor sub-regions, pre-processing and post-processing methods used for refining the evaluation methods, and prediction of results based on the model architectures. And also what type of project management technique is used and choosing the scrum methodology for the software development process for processing the project.

And for further research, we are dealing with and modifying the 3D U-Net model which is originally based on the U-Net model, and comparing our evaluation scores with the general methods used in the state-of-art algorithms for the BraTS 2021 dataset.

4. Methodology

This section briefs into sub-sections for completion of the project.

1) Project Management explains the method chosen in project management techniques for doing the project. And overall planning done for the project has been outlined using the Gantt chart.

2) Software development process briefs about the type of methodology used is scrum, which is used for completing the project by implementing the phases and principles of scrum methodology.

3) Toolsets and Machine Environments briefs about the tools and software used for handling the project and explaining about the libraries taken and installed for loading the dataset, training the model, and evaluating the model results with evaluation methods.

4) Dataset gives a brief description of the data we are using.

5) Research methods give a brief description of the end-to-end analysis of the dataset, Segmentation methods selected for segmenting the 3d MRI images of Brain tumor segmentation, how the modifications are done on the proposed 3d U-Net model, and evaluation methods that are going to be implemented for the predictions and evaluating the results.

4.1 Project Management

This section focuses mainly on the project management techniques used for this project and also discussed why some of the other techniques are not used for the project and also says about the effective of the method chosen for developing the brain tumor segmentation project.

4.1.1 Gantt chart

The Gantt chart was created for outlining the plan of the project for completing the stages performed in the project.

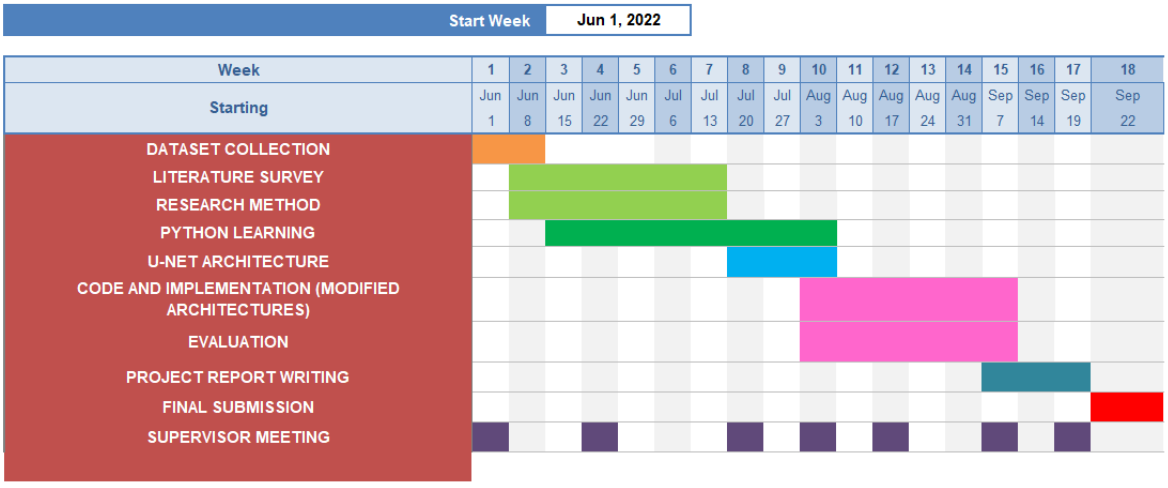


Figure 4.1: The final Gantt chart for the project.

The final Gantt chart has been created based on the activities carried out for the completion of each stage in the project. The major advantage of this Gantt chart is displaying the stages that needed to be completed within the deadline. Some tasks have been completed within the specified deadline like handling the dataset and learning about the proposed model. But, In the code and implementation stage, modifying the model architectures took some more time for training the models. Usually, the code and implementation stage needs more time for all the projects.

4.1.2 PERT chart

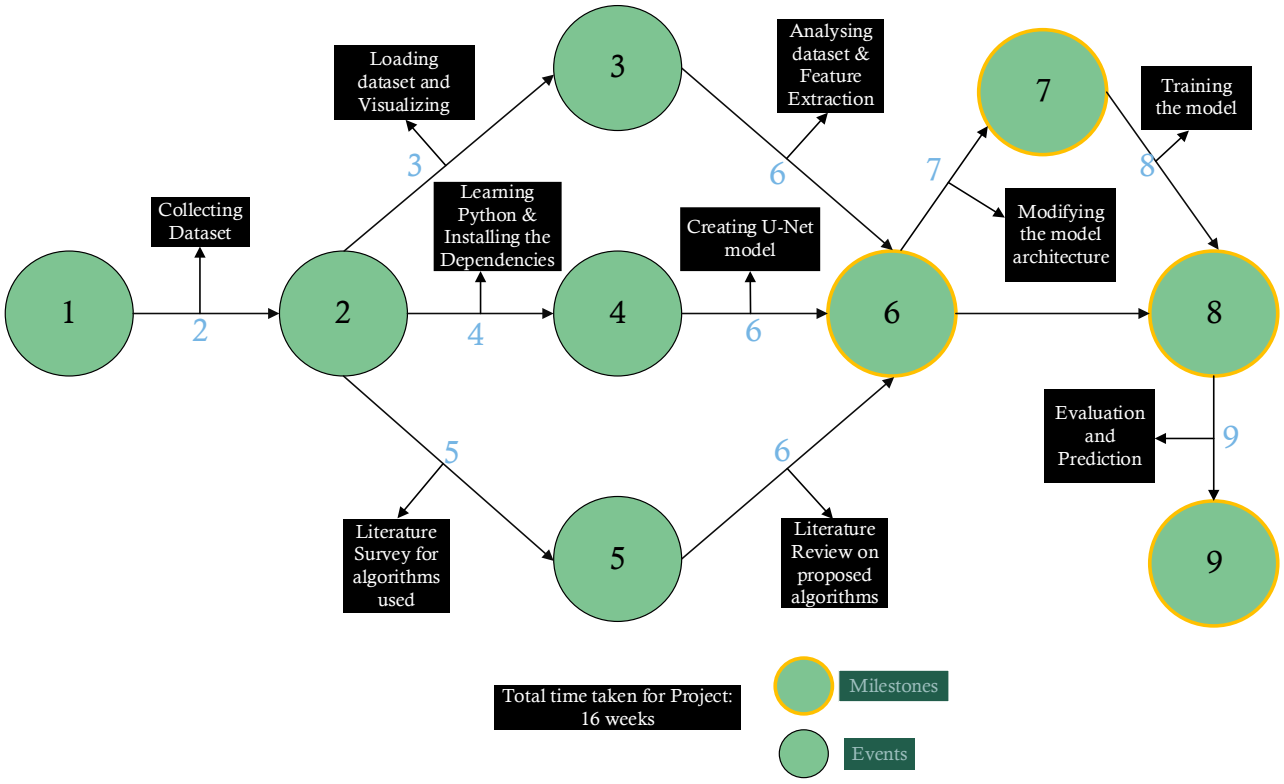


Figure 4.2: The PERT chart for the Brain tumor segmentation project.

The PERT Chart has been created in addition to the Gantt Chart which helps in determining the duration of each task to be completed in each stage of completing the project. And also shows that the project can only be focused on a single task or that multiple tasks can be done simultaneously or not. And it was created before starting the project for giving a rough estimation of the time taken to complete the brain tumor segmentation project.

4.1.3 Risk Assessment

For completion of the project, the maximum time has been allotted for the code and implementation stage, where in this stage the proposed model has been tested by varying the model architectures. The main issue of the project is that the whole project has been carried out in the lab systems provided by the University of Lincoln. As the project is dealing with the 3D dataset and the overall size of the dataset is around 13GB. Even though, the machines in the lab are having specifications of (Specifications – 11th Gen Intel core i5 1160k, having Nvidia 3060 GPU graphics card with 16GB DDR4 memory and 1TB SSD) the modified models in this project are throwing some memory errors for training the model at the cropped size of the image. So, taking into consideration of memory error the modifications needs to be done without causing memory issues or out-of-shape issues. And for training 100 epochs of each modified architecture model took around 10 to 12 hours of time. And taking into consideration of time, every day the machines in the lab get automatically rebooted and reimaged, and on every week there will be a software maintenance period. So, every time when you start the machines you have to re-download the dataset and reinstall the libraries. These were the risks and planning and management of time is done on this project.

4.1.4 Supervisor meetings

Throughout the project, we are under the guidance of the supervisor. The most important part is to complete the project by having frequent meetings with the supervisor as this was considered one of the most valuable project management techniques in completing each stage of the project according to the specified deadline. These meetings should be taken on every week and know when the supervisor is available and request them to schedule the meeting by showing the progress of our work and deciding the further research in completing the stages of the project. These meetings helped me to complete the research project in time and learnt the research skills from the supervisor. And also known how to structure the reported work.

4.1.5 PRINCE2 Technique.

Out of many methodologies in project management, this research project is planning to use the PRINCE2 technique (Lloyd and Macrae, 2020) and it is elaborated as the projects in controlled environments. This method is widely used in the United Kingdom in comparison with other countries. This methodology is used by the local government, and National Health Service (NHS) because this method is well-structured and more effective in nature for managing any size or type of the project. This technique is specially built keeping IT in the mind and uses its own framework as a principle and is intended for the project needs by not setting the hard rules.

PRINCE2 technique (Lloyd and Macrae, 2020) consists of six characteristics where the project has been tracked and well progressed they are: 1) Project scope, 2) Costs, 3) Timescales, 4) Risk Assessment, 5) Quality, and 6) Benefits.

1) Project scope – The major scope of the project is to develop the machine learning algorithms by using the advanced technology model i.e., 3d U-Net for the Brain tumor segmentation of the BraTS dataset by modifying the architectures and testing the model for helping the research work.
2) Costs – Designing the cost-efficient for the project, the project model is trained and tested in the lab systems having a specification of i5 core 11^{th} generation systems with 16GB ram, Nvidia 3060 GPU, and 1TB SSD provided by the University of Lincoln by making use of Lab infrastructure. This includes zero hardware cost and talking about the software part, the project uses the free licensed version open-sourced software i.e., python for evaluating the whole project in an end-to-end fashion.
3) Timescales – On talking about the timescales, the amount of time taken for completing the research project work is three months. Starting from the project's first month, it dealt with the collection of the dataset and finding suitable literature work, and reviewing the literature. In the second month, the project was known to go with which machine learning algorithm. And learning about the algorithm and coding the work and in the final month, the research project deals with implementation, evaluation work, and writing the report work.
4) Risk Assessment – this section is explained in the section 4.1.3 Risk Assessment
5) Quality – In this project, the quality of the images are checked by cropping the image size to 128*128*128 for predicting the precise size and location of tumors present in the tumor sub-regions. Further, it can be deployed in real-time and checked the quality of the research work done for further meeting the standards.
6) Benefits – The major benefit of adopting this project management framework technique for doing this project is to keep tracking and progressing the tasks done in each stages, managing the cost-efficient of the project, and reviewing the supervisor meetings by improving the further research work by updating the research to the medical image processing field.

4.2 Software Development

This section briefs about the software development method and techniques used for this project and says how the chosen methodology (SCRUM) has helped the project.

4.2.1 SCRUM

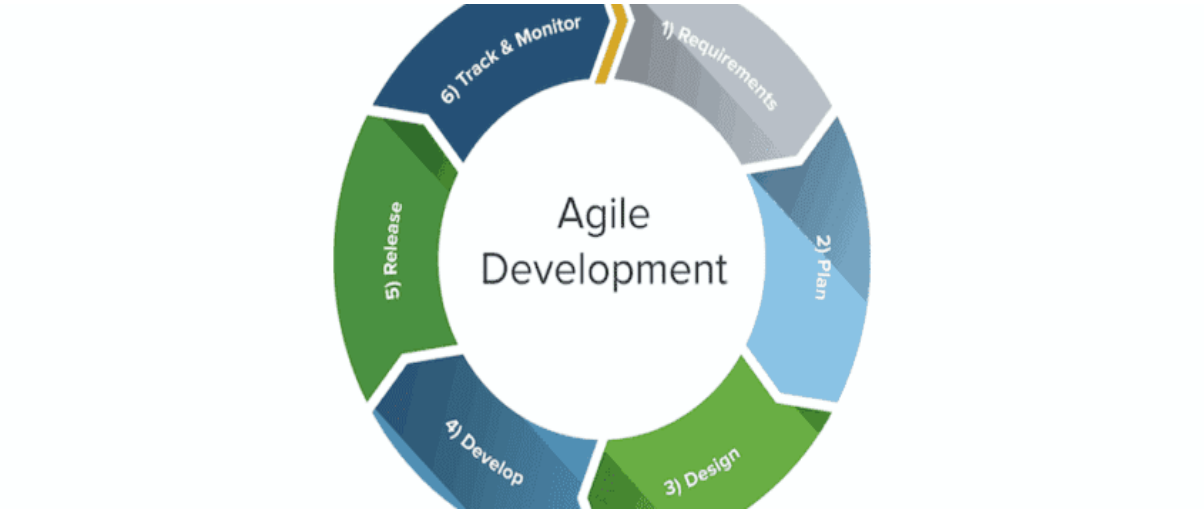


Figure 4.3: The complete cycle of Agile Scrum Framework.

The Scrum Methodology is the most famous and popular framework which is a part of agile methodology and this research project follows the scrum methodology framework which is derived from the agile methodology for completing the project at a shorter period of time. Scrum is a commonly used iterative object oriented development of cycle and this approach increases the flexibility which is responsible for initial and additional requirements that are discovered during the development of the project (Schwaber, 1997). The project is dealt with managing the time pressure, producing quality work, and taking use of resources of the related works (Schwaber, 1997). The process involved at each stage in the scrum methodology is called sprints having fixed-length iterations that last for two to three weeks. The overall project is led by the scrum master who clears all the obstacles for executing the project's day-to-day work. The main supervisor of the research project acts as a scrum master for clarifying the difficulties faced at each stage of the project by having teams meetings and helping the project to complete all the stages.

The research project is further divided according to the plan into stages which are called as sprints for completing the project. As the project is having only the main developer taking all the responsibility for completing the stages in the project under the guidance of the scrum master – The Supervisor. The main reason for selecting the scrum methodology framework is that every stage in the project lasts only for two to three weeks and the sprints used in the scrum methodology help the project by making the stages into sprints and each sprint have two to three weeks for completion of tasks. The project started with the collection of the dataset and further researching the algorithms performed on the dataset is considered as one sprint which usually takes to complete the task in two to three weeks. And further, each stage or sprint has separate tasks which help to complete the activities by developing the project. At each sprint, there will be meetings held by the scrum master for knowing the progress of the research work and reporting the work. The scrum methodology could be used without having the daily scrums. The team meetings in the project helped in keeping the track of the project work and collecting feedback and further improving the research in the next sprint meeting.

The testing and modifying the model work has been carried out in the training phase. Deploying the tested and modified models works are not done in the real-time production field environment and is not part of the research scope. Overall the project management helped the project to train the algorithms and test the model was beneficial for further evaluating the results. And from the project point of view,

the project implemented the scrum methodology and had a major benefit in completing the tasks by making sprints and completing the research project, and experienced the best practice for the software development process.

4.3 Toolsets and Machine Environments

This section briefs about the tools that are used for the development of the Brain tumor segmentation research project such as using the IDE (Integrated development environments) for writing the code, and model training, the libraries used for handling the dataset, and evaluating the results. Talking about the version controls like GitHub is used for storing the implementation code work.

This project could have used several integrated development environments like Jupyter notebook, Spyder, Google colab, Pycharm, and more.

Integrated Development Environment (IDEs)	Google Colab	Jupyter Notebook	PyCharm	Spyder	Visual Studio Code	IDLE
SNMP Support (Simple Network Management Protocol)	Available	Available	Available	Available	Available	Available
Auto completion	Yes	Yes	Yes	No	Yes	No
Libraries support of the third party	Extensive Support without installation	Good Support but we have to install	Good Support but we have to install	Good Support but we have to install	Good Support but we have to install	Good Support but we have to install
Customized Layout	Best	Best	Good	Good	Good	Average
Features Handling	Easy to use	Easy to use	Little confusing	Little confusing	Little confusing	Confusing
Integration of Git	Yes	Yes	Yes	NO	Yes	NO
Statistic program analysis	Yes	Yes	Yes	Yes	Yes	NO
Debugging	Yes	Yes	Yes	Not ideal for work	Yes	NO
Compatibility	More	More	Medium	Medium	Medium	Average
Design of software	Best	Best	Good	Good	Good	Average
Cost	Free but limited and involves buying premium	Freely available	Freely available for having a yearly licensed version	Freely available	Freely available	Freely available
Command line integration	Run on the Command prompt	Run on the Command prompt	Yes, it's there	Available via plugin	Yes, it's there	Yes, it's there

Open-Sourced Software	NO	Yes	Yes	Yes	Yes	Yes
Programming Languages	Python	Python, java, Julia, Matlab, Scala etc..	Python, Django	Python	Typescript, HTML, Javascript, Extensions (python, C/C++)	Python

Table 1: Comparison of Different Integrated Development Environments.

From the above table, on comparing with all the IDEs, we can see visual studio needs some installations of extensions for using python even though it is provided with a lot of required features. And Spyder lacks some functional features while selecting the code and running the code on comparing with the other IDE's. The remaining IDE's like Google colab, Pycharm, and Jupyter notebook can be used for doing this research project.

The Jupyter notebook and Google colab are having the same features, The Google colab is run on the cloud resources using high GPU's. The Google colab has provided free 15 Gb space and use of GPU's on creating the G-mail accounts. As in our project the dataset size is around 13 Gb for uploading and loading it in the drive, on further training the model and saving the weights of the model in the hdf5 format, it needs more memory to store. The project could have gone with the google colab for faster usage of GPUs. As the google colab is limited for certain memory, after completing the free memory provided by the Google colab we have to buy the google colab pro costs around 10 pounds per month (giving extra 50Gb storage) or google colab pro+ cost around 100 GBP pounds (providing 150Gb of disk space). As the Jupyter notebook is having the same features and using the software is free as it is open-sourced. For using the Jupyter notebook any systems are fine, it all depends on the system configurations. But, overall we have to think about the completion of the project in time. As the researcher is going with the Jupyter notebook having all the specific required features and by making use of the lab systems infrastructure having specifications of (Specifications – 11th Gen Intel core i5 1160k, having Nvidia 3060 GPU graphics card with 16GB DDR4 memory and 1TB SSD) provided by the University of Lincoln. The training of model and modified models has been done on the university premises using the LAB Systems.

The key libraries used for completing the Brain tumor segmentation project are:

1) TensorFlow – An open-sourced platform used for creating the models and used as the library for the machine learning productionize. Its major focuses on the part of training and inferencing the deep neural networks.
2) Keras – Keras is a highly-productive interface and high-level API of the TensorFlow. Keras is used for training deep learning models.
3) Itkwidgets – This library helps to visualize the images of 2D and 3D and also helps in spatial analysis of an image.
4) nilearn – This library helps in analyses of brain volumes by providing machine learning and statistical tool for performing multivariate statistics for classification, and modeling.
5) nibabel – It is open-source software that helps to load the nifty file formats used in the medical imaging field. Annotation files can be viewed using this package which is licensed under MIT license.
6) OpenCV – It is a real-time computer vision library that supports in the execution of the model for the machine learning and open-sourced software.
7) sklearn – It is the package of scikit-learn which is simple and used for the analysis of the data and built on the Numpy and matplotlib, licensed by BSD.

8) seaborn – It is a data visualization library which is based on the matplotlib and gives informative statistical graphics.
9) pandas – It is open-sourced and widely used data analysis tool and the library is written for the python language. Data structures and operating the tables and time series operations are done.
10) NumPy– It is a mathematical functions tool, that generates a random number and other mathematical operations are used on the 2D or multi-dimensional arrays and matrices.
11) Microsoft Visio professional – This tool helped the project to design the model modified architectures network diagrams and also helped to design the PERT Chart of network diagram of the project.
12) Microsoft word – This tool helped to structure the reported work of the Brain tumor segmentation project.

GitHub is an open-sourced software and version control using Git which also helps in having the continuous integration of the work by pulling the work for updating and making commits. All the project implementation of coding work has been stored on GitHub by making a separate repository.

4.4 Dataset

Description of Data

The dataset has been taken from the Kaggle Competition for brain tumor Segmentation of BraTS 2021 https://www.kaggle.com/datasets/dschettler8845/brats-2021-task1

The dataset consists of a collection of mpMRI scans of brain tumors in glioma which is taken from different institutions under standardized clinical conditions with available methylation status of MGMT promoter and confirmed diagnosis, which is used as training, testing, and validation data for BraTS 2021 challenge. In comparison with last year's BraTS 2020 dataset, this year dataset has been updated with the ground truth annotations for tumor sub-regions are created for training, testing, and validation datasets which are approved for every subject by the expert neuroradiologists to evaluate the predicted tumor Segmentations. And also, the dataset is provided with the naming conventions and name mapping between the BraTS 2021- 17 dataset. And main consideration taken for this challenge is the participants are not allowed to use the different datasets of their own by extending the BraTS data for training the algorithm and also using the pre-trained models on their own created dataset is not allowed.

Imaging data description

The BraTS 2021 dataset contains mpMRI scans which are available in the format type of NIfTI files having (.nii.gz) describes a) T1-native, b) T1Gd-post-contrast T1-weighted, c) T2- weighted, d) T2-Flair i.e., T2 Fluid Attenuated inversory recovery volumes were obtained using diverse clinical procedures and various scanners from several data providing institutions. When the challenge gets over, they are planning to release the associated de-identified DICOM (.dcm) files.

The imaging dataset has been annotated manually from one to four raters by following the same annotation protocol, and their annotations were validated by knowledgeable neuroradiologists. Annotations include the necrotic tumor core (Label 1- NCR), the peritumoral edematous/invaded tissue (Label 2- ED) and GD- enhancing tumor (Label 4- ET) which is described in the BraTS 2012 and BraTS 2013 paper. After their pre-processing the ground truth data has been created which is co-registered with same anatomical template, interpolating to the same resolution (1 mm3), and skull-stripping. With the help of Cancer Imaging Phenomics Toolkit (CaPTk) and Federated Tumor Segmentation (FeTS) toolkit, the pre-processing pipeline is publicly available. For skull stripping they have used the Deep Learning approach for brain shape prior which is agnostic to MRI input sequence.

4.5 Research Methods

From the overall literature review, the appropriate methods are chosen and implemented through the steps taken for doing this project are:

Step 1: Setting up the data.

- Downloading and unzipping the Brain Tumor Segmentation of BraTS 2021 dataset from the Kaggle website - https://www.kaggle.com/datasets/dschettler8845/brats-2021-task1.
- Extracting the tar files containing .nii.gz format files.
- Installing the nibabel library for handling and dealing with nii files. (https://nipy.org/nibabel/).
- Importing the related dependencies for plotting the images, and evaluating the matrix, as the python script written here is using TensorFlow Keras.
- The given size of an image is 240*240*155 and for further extracting the features from the image is resized to 128*128*128 by scaling all the volumes using the MinMaxScaler method.
- Dropping the volumes where the percentage of annotated data is under a specific threshold (This is carried out in order to maximize the training on actual labeled volumes).
- Segmenting the classes based on the tumor sub-regions which are provided by the specific annotations they are: 0- Normal - means no tumor, 1- Necrotic/core region (Label 1), 2-Edema region (Label 2). 3-Enhancing tumor region (Label 4).
- Scaling the volumes by doing the slices per volume, as there are 155 slices per volume. We are doing the volumetric segmentation by slicing the volume to 128 and the image size is set to 128.
- The dataset is provided with the training data containing 1251 patient records. Splitting the data and masking the volumes into training (0.65), testing (0.15), and validation (0.20).

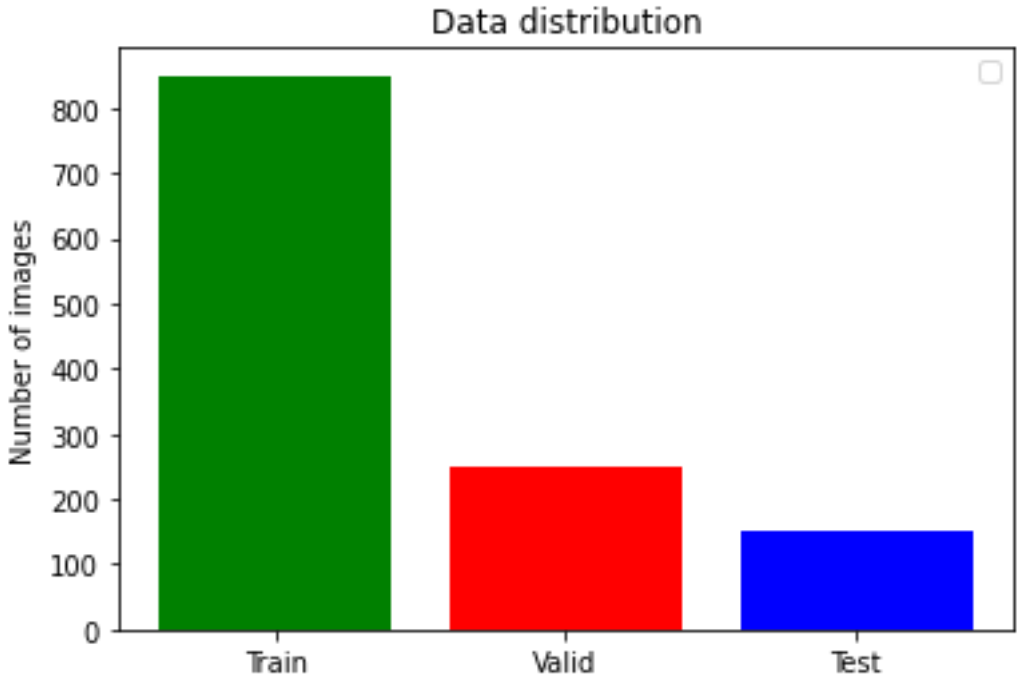


Figure 4.4: The training dataset containing 1251 patient record data has been separated into train -containing 850 data, valid - containing 251 data, and test - containing 150 data for training the model with the proposed method.

Step 2: Defining the custom data generator.

- The Keras image data generator only works with tif, jpeg, and png image formats. As the provided images in the dataset are in the form of .nii.gz format and this format cannot be recognized by the Keras image data generator. Therefore, for this sake, we should define the custom data generator for loading the data from the dataset.

Step 3: Formulas for the evaluation methods.

- Loss function:
  The evaluation metrics for the BraTS dataset has been derived from the (Dong et al., 2017) paper and (Sundaresan, Griffanti and Jenkinson, 2021) paper.

  1) Dice Similarity Co-efficient loss is the measurement of overlap between the two samples. The dice loss measurement will be ranging between 0 to 1, where 1 denotes the perfect and overlap of dice co-efficient of the sample.
  The dice coefficient is calculated as:

$$DSC = \frac{2 * |X \cap Y|)}{(|X| + |Y|).}$$

$$DSC = 2 * \text{sum}(|A * B|) / (\text{sum}(A\text{^}2) + \text{sum}(B\text{^}2))$$

  2) For calculating the segmentation scores of Dice co-efficient of necrotic, Dice co-efficient of Edema, and Dice co-efficient of Enhancing tumor of each region, the segmentation has been evaluated using the dice similarity coefficient for each epoch.

  3) MeanIoU (Intersection over Union) score is the common metric of the evaluation method generated for the Semantic Image Segmentation. The IoU metric score generated for an individual class is:

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

  4) Sensitivity is the measurement of the test for correctly identifying the patients with the disease or correctly identifying the number of true positives (TP) and false negatives (FN). For the segmented regions, it is used to measure the sensitivity of the model.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

  5) Specificity is the measurement of a test for correctly identifying the patients without disease and used for evaluating or identifying the number of true negatives (TN) and false positives (FP). And also used to measure the ability of the model for predicting the background area.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

  6) Precision is measured by the number of positive class predictions that actually fall into the positive class. Also stated as the quality of positive prediction produced by the model, the number of true positives which are divided by the total number of positive predicted from the model.

$$Precision = \frac{TP}{TP + FP}$$

Step 4: Defining the proposed model.

- In this stage, we are using the 3D U-Net architecture algorithm to test the model which is originally designed based on the original U-Net architecture model.
- And, several modifications are done for the proposed model architecture in terms of adding the convolution layer, changing the number of filters to 4,8,16, and 32 with a kernel size of 2*2*2 and 3*3*3 also adding and removing the dropout layers to prevent overfitting.
- The proposed method was validated and tested on the BraTS 2021 datasets containing 1251 training data.

3D U-Net MODEL

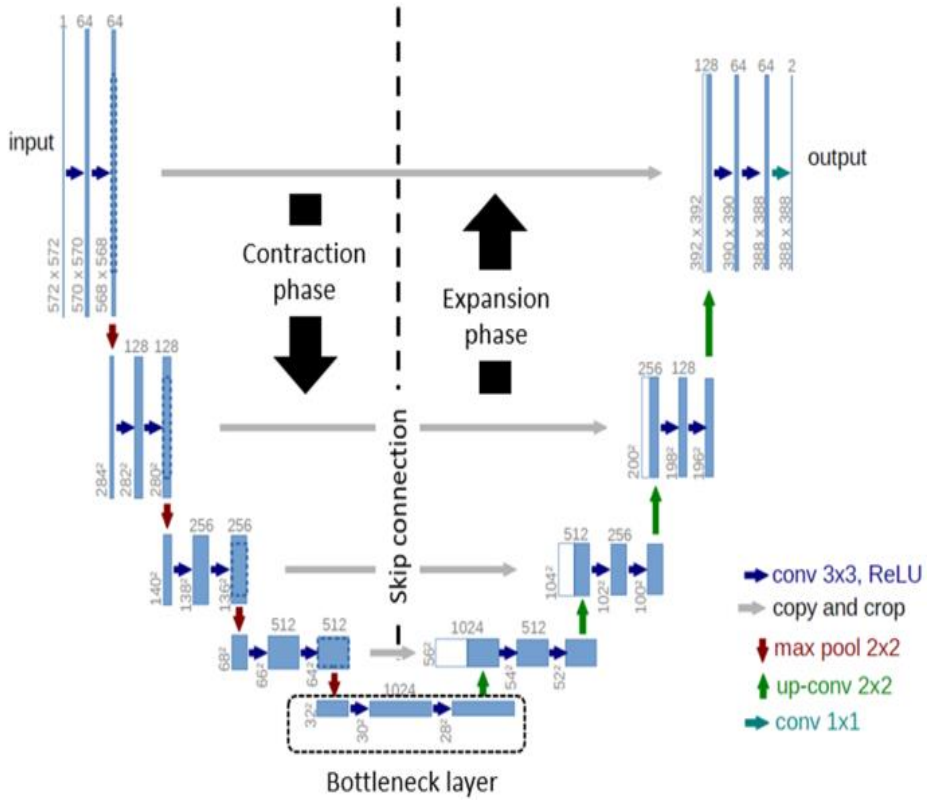


Figure 4.5: The U-NET Model architecture.

The model is designed and modified based on the original U-Net network. The model architecture has been divided into encoder and decoder paths. The encoder path also called as contraction path, where the (MRI) Magnetic resonance images (2×128×128×128) were passed in the first stage of 3D U-Net for the prediction of Segmentation maps roughly and the segmentation map is fed together with the raw images in the decoder path which is also called as expansion path i.e. in the second stage of 3D U-Net for prediction of accurate segmentation map with the network parameters. The 3D U-Net is trained in an end-to-end fashion. Due to the limitation of GPU memory, our proposed network is designed to take the inputs of an image size of 128×128×128 voxels using batch size 1.

It consists of the application repeated of two 3*3*3 Convolutions which are unpadded where each convolution is followed by the ReLU linear activation function with 2*2*2 Max pooling operations for downsampling in the encoder path i.e., contraction path. In each downsampling step, the number of feature channels is doubled (Ronneberger, Fischer and Brox, 2015).

And In the Decoder path i.e., the expansion path, every step consists of an Upsampling of the feature map which is followed by the 3*3*3 Convolution which is up-convoluted, and each convolution layer is followed by the ReLU linear activation function. In each Upsampling step, the number of feature

channels is halved and concatenated with the corresponding encoder blocks added to the encoder path. And later the convolution 3D layer is transposed with the number of feature channels with 2*2*2 Convolutions followed by the same padding and with the stride of 2 for Upsampling (Ronneberger, Fischer and Brox, 2015).

The cropping is done at every convolution layer due to the loss of border pixels. In the final layer, a 1*1*1 convolution is used with the desired number of classes present in the dataset to feature the map with 64 component feature vectors and the overall network consists of 23 convolution layers (Ronneberger, Fischer and Brox, 2015). When the operations are set, the model is further used for training the model and evaluating the results. While remodifying the architectures, batch normalization is set to true for faster training and standardizes the inputs to each layer by making mini-batches. And remodifying the model architectures for further testing the model performance and evaluating the results for final comparison and comparing with other state-of-art algorithms trained on the Brain tumor segmentation of the BraTS dataset.

3D U-Net Modified Model Architectures

Further, the model is modified by tuning the hyperparameters for improving the performance and evaluating the evaluation methods for comparing the results. The model is hyper tuned by changing the convolution layers, updating the kernel sizes of each layer, changing the number of filters in the encoder and decoder blocks, max pooling operations are done after each encoder block, and in the decoder block the convolution 3D Transpose is used with the strides, each convolution layer has activation ReLU function of each encoder and decoder block, and in the final layer, the 1*1*1 convolution is added with the Softmax activation function. Furthermore, each modification done to the model is clearly explained in the implementation part of the Design and development section (Refer to the section 5.3, 5.3.8).

Step 5: Training and prediction of BraTS 2021 dataset.

- In this phase, we are going to load the images of the data in batches using the custom data generator and fed them into the training.
- Generating or Predicting the Dice similarity co-efficient score, Precision, Sensitivity, Specificity, and predicting the tumor sub-region scores with respect to Dice Co-efficient of Necrotic region, Dice Co-efficient of Edema region, Dice Co-efficient of Enhancing tumor region for each epoch, evaluating and comparing the final overall generated scores for the prediction of model with other state-of-art-methods algorithm scores.
- Plotting the graphs of data with visualization generated for training and validation sets, plotting the Training Accuracy with Validation Accuracy, Training Loss with Validation Loss, Training Dice Similarity Co-efficient with Validation Dice Similarity Co-efficient, and Training MeanIoU score with the Validation MeanIoU score.

5. Design and Development

This section, explains how we are planning the design of the project and the activities carried out in the development stage for executing the design efficiently. And also says how the planning meets the design and development objectives. As this project involves the designing of the model and the modifications done in the development stage for the functioning of the machine learning algorithm (3D U-Net). This project describes the software development process carried out in each stage will be discussed below. And what all the implementations have done to the model by modifying the architectures and training the model for further evaluating the results.

5.1 Requirements

The functional requirements for the design are supported by the outline of the design process carried out for completing the project using the local systems in the lab provided by the University of Lincoln. The major requirement for this project is to complete all the planned design and development activities like key aspects and the phases process involved in the project. And the following essential requirements for this project:

1) loading the MRI images and installing the suitable libraries;
2) Simplifying the data for further research work;
3) Handling the modifications done to model architectures without causing memory issues.
4) Designing the cost-efficient infrastructure to train the model by utilizing the Lab systems provided by the University of Lincoln.
5) Software requirements – Python, installing the related libraries like TensorFlow, Keras, sklearn, nibabel, matplotlib, itkwidgets, nilearn, and GitHub.
6) Hardware Requirements – Systems with High GPU and High computing power for withstanding the load of the project model work.

5.2 Design

The overall pipeline is built for processing the data flowing through the machine learning project solution. In each step of the pipeline, it is fed with the processed data from the preceding step. The ML pipeline helps in automating the machine learning workflow by transforming the data and correlating it to the model for further analysis to produce the outputs. Based on the results achieved, further, it can be deployed in the real case environment in the field of production for testing the model. The key stages involved in the pipeline are: 1) Problem Definition, 2) Data Ingestion, 3) Data Preparation, 4) Data Segregation, 5) Model Training, 6) Model Evaluation, 7) Model Deployment, and 8) Performance and Monitoring.

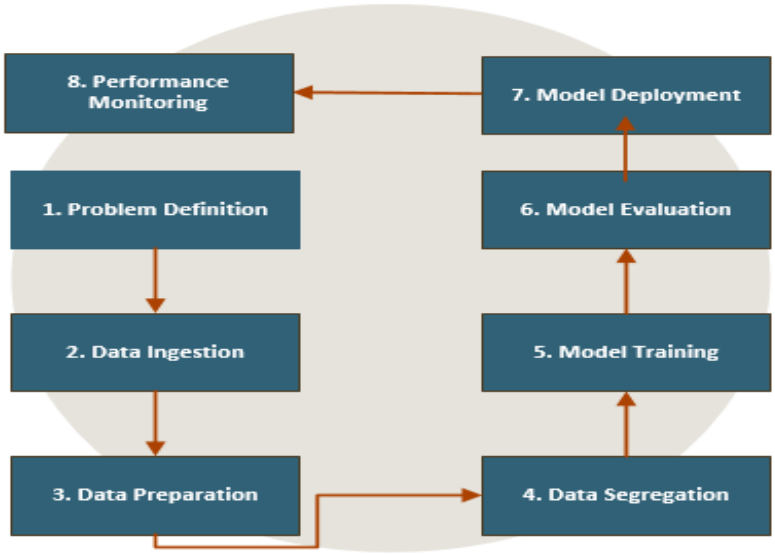


Figure 5.1: The overall Machine Learning pipeline consists of different phases provided for the project solution.

1) Problem Definition – In this stage, the major problem definition of the project is to accurately segment and detect the regions prior to their size, and location of tumors in tumor sub-regions of Brain Tumor Segmentation of the BraTS 2021 dataset.

2) Data Ingestion – In this stage, the BraTS 2021 dataset stored in the local systems has been fed to the model in the batch processing method, where the MRI images are loaded in the batches and sent to the algorithm for evaluating the results.

3) Data Preparation – In this stage, the correct form of data has been processed by unzipping the original BraTS 2021 dataset containing the tar files and checking in which format the images are, the images provided are in the .nii.gz format. To load the data nibabel library has been used for visualizing the dataset. And here the images are resized for further segmentation.

4) Data Segregation – In this stage, the provided dataset has been Split into training (0.65), testing (0.15), and validation (0.20) for further training of the model to perform the evaluation methods.

5) Model Training – In this stage, the images in the dataset are fed into the proposed algorithm for recognizing the patterns. The 3d U-Net model has been designed and also modified the 3D U-Net model architectures for training and generating the results at each epoch.

6) Model Evaluation – In this stage, the performance of the proposed model and modified model architecture algorithms has been assessed using the test generator and validation generator for understanding how accurate the prediction is.

7) Model Deployment – In this stage, the proposed model and modified model architectures are produced and deployed in the local lab systems (Specifications – 11th Gen Intel core i5 1160k, having Nvidia 3060 GPU graphics card with 16GB DDR4 memory and 1TB SSD) in the lab which is provided by the University of Lincoln.

8) Performance and Monitoring – In this stage, the modified 3D U-Net models are continuously monitored and observed their behavior in terms of generating the scores for tumor sub-regions

and other evaluation methods chosen for the project. And later, comparing the generated scores of each modified model architecture.

5.3 Implementation

This section briefs about the implementations done for completing the tasks in the coding work from loading the dataset, feature extraction of the images, visualizing the datasets, creating the model and training and evaluating the results, remodifying the model architectures, and training the model for evaluating the performance of the model and comparing the results.

5.3.1 Libraries installation.

Step 1: Installing the Dependencies.

Installing the dependencies

```
import numpy as np
import nibabel as nib
import itk
import itkwidgets
from ipywidgets import interact, interactive, IntSlider, ToggleButtons
import matplotlib.pyplot as plt
from skimage.util import montage
from skimage.transform import rotate

%matplotlib inline
import seaborn as sns
```

```
import keras
import keras.backend as K
from keras.callbacks import CSVLogger
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, TensorBoard
from tensorflow.keras.layers.experimental import preprocessing
import cv2
# Make numpy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
```

```
import nilearn as nl
import nibabel as nib
import nilearn.plotting as nlplt
```

Figure 5.2: Installing the Dependencies for the research project.

This step involves the installation of related libraries for doing the research work for loading the dataset, visualizing the dataset, plotting the model, training the model, and evaluating the model.

5.3.2 Pre-processing the Dataset.

Step 2: Loading the dataset.

```
image_path = r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/BraTS2021_00005/BraTS2021_00005_flair.nii.gz"
image_obj = nib.load(image_path)
type(image_obj)
```

```
nibabel.nifti1.Nifti1Image
```

```
image_data = image_obj.get_fdata()
print(type(image_data))
image_data.shape
```

```
<class 'numpy.ndarray'>
(240, 240, 155)
```

```
TRAIN_DATASET_PATH = r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/"
```

```
import os
nSample = os.listdir(TRAIN_DATASET_PATH + 'BraTS2021_00005')
nSample
```

```
['BraTS2021_00005_flair.nii.gz',
 'BraTS2021_00005_seg.nii.gz',
 'BraTS2021_00005_t1.nii.gz',
 'BraTS2021_00005_t1ce.nii.gz',
 'BraTS2021_00005_t2.nii.gz']
```

```
test_image_flair=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_flair.nii.gz').get_fdata()
test_image_t1=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t1.nii.gz').get_fdata()
test_image_t1ce=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t1ce.nii.gz').get_fdata()
test_image_t2=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t2.nii.gz').get_fdata()
test_mask=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_seg.nii.gz').get_fdata()
```

Figure 5.3: Loading the images from the training dataset for all the classes.

This step involves loading the classes of the dataset and checking the image format and size of the images shown are 240*240*155. And checking all the data formats and loading the images using the nibabel library.

Step 3: visualizing the images in the dataset.

```
def visualize_3d(layer):
    plt.figure(figsize=(10, 5))
    plt.imshow(image_data[:, :, layer], cmap='gray');
    plt.axis('off')
    return layer

interact(visualize_3d, layer=(0, image_data.shape[2] - 1));
```

```
interactive(children=(IntSlider(value=77, description='layer', max=154), Output()), _dom_classes=('widget-inte…
```

```
classes_dict = {
    'Normal': 0.,
    'Edema': 1.,
    'Non-enhancing tumor': 2.,
    'Enhancing tumor': 3.
}

def visualize_3d_labels(layer):
    mask = nib.load(r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/BraTS2021_00005/BraTS2021_00005_seg.nii.gz").get_fdata()
    plt.imshow(mask[:,:,layer])
    plt.axis('off')
    plt.tight_layout()

interact(visualize_3d_labels, layer=(0, image_data.shape[2] - 1));
```

```
interactive(children=(IntSlider(value=77, description='layer', max=154), Output()), _dom_classes=('widget-inte…
```

```
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1,5, figsize = (20, 10))
slice_w = 25
ax1.imshow(test_image_flair[:,:,test_image_flair.shape[0]//2-slice_w], cmap = 'gray')
ax1.set_title('Image flair')
ax2.imshow(test_image_t1[:,:,test_image_t1.shape[0]//2-slice_w], cmap = 'gray')
ax2.set_title('Image t1')
ax3.imshow(test_image_t1ce[:,:,test_image_t1ce.shape[0]//2-slice_w], cmap = 'gray')
ax3.set_title('Image t1ce')
ax4.imshow(test_image_t2[:,:,test_image_t2.shape[0]//2-slice_w], cmap = 'gray')
ax4.set_title('Image t2')
ax5.imshow(test_mask[:,:,test_mask.shape[0]//2-slice_w])
ax5.set_title('Mask')
```

```
Text(0.5, 1.0, 'Mask')
```

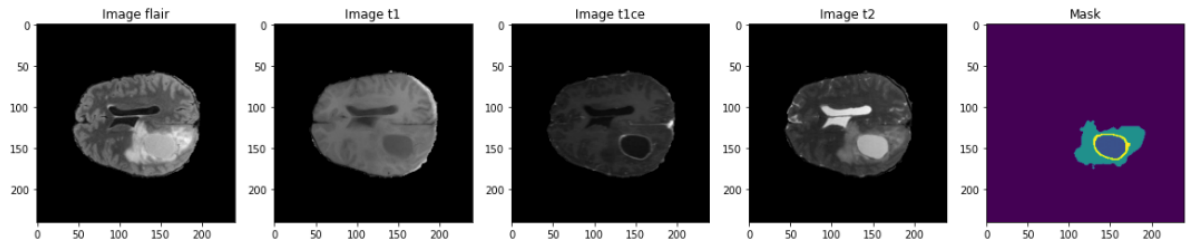


Figure 5.4: Visualizing the images in the classes of the dataset.

This step involves visualizing the images of each class like flair image, t1 image, t1ceimage, t2 image, and mask (segmented) image by plotting it and checking the image pixel size.

5.3.3 Feature-Extraction of the Dataset.

Step 4: Segmentation of images in the dataset.

```
# DEFINE seg-areas
SEGMENT_CLASSES = {
    0 : 'NOT tumor',
    1 : 'NECROTIC/CORE', # or NON-ENHANCING tumor CORE - RED
    2 : 'EDEMA',  # Green
    3 : 'ENHANCING' # original 4 -> converted into 3 later, Yellow
}

# there are 155 slices per volume
# to start at 5 and use 145 slices means we will skip the first 5 and last 5
VOLUME_SLICES = 128
VOLUME_START_AT = 22 # first slice of volume that we will include

IMG_SIZE=128
```

This step involves volumetric segmentation of the images according to their classes by slicing the image to 128 out of 155 slices per volume and giving the image size to 128.

Step 5: Plotting the annotation images and masking them.

```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_image_t1[50:-50,:,:]), 90, resize=True), cmap ='gray')
```

```
<matplotlib.image.AxesImage at 0x19281621370>
```

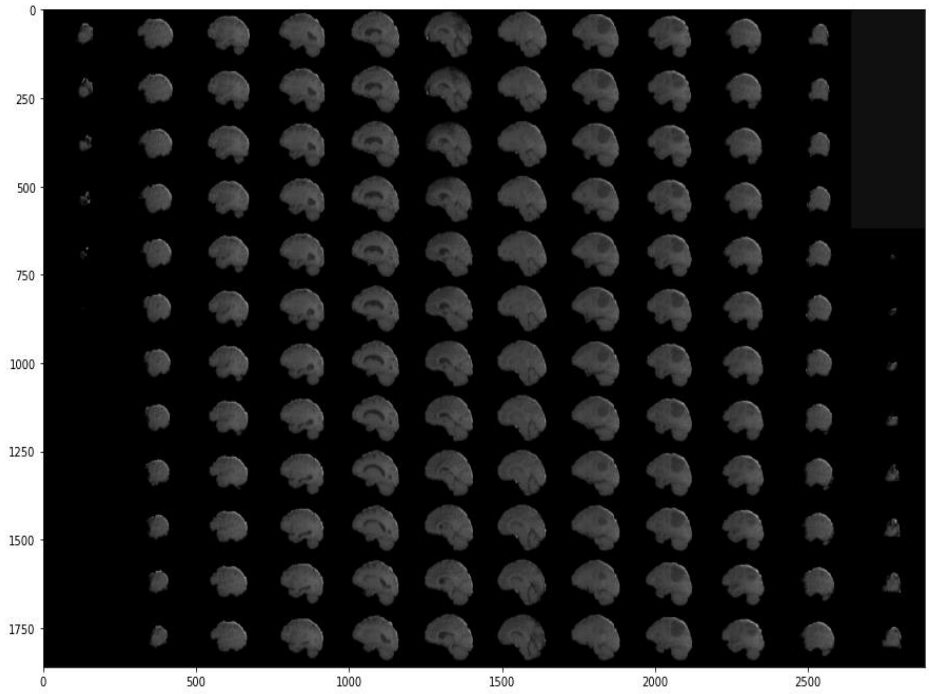


Figure 5.5: plotting of t1 image.

The above figure shows the rotation of annotated t1 images at different pixels at 90 degrees.

```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_mask[60:-60,:,:]), 90, resize=True), cmap ='gray')
```

```
<matplotlib.image.AxesImage at 0x192802b7fd0>
```

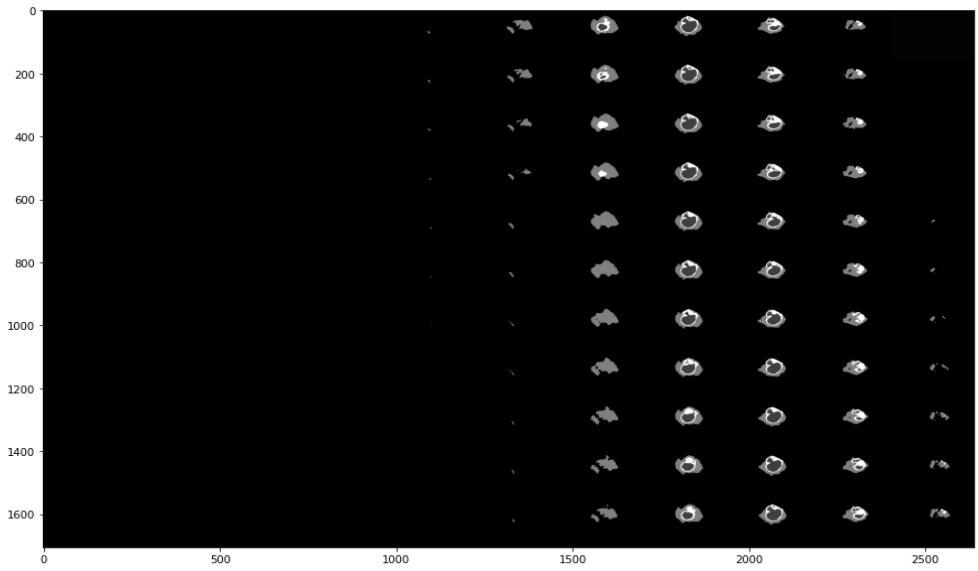
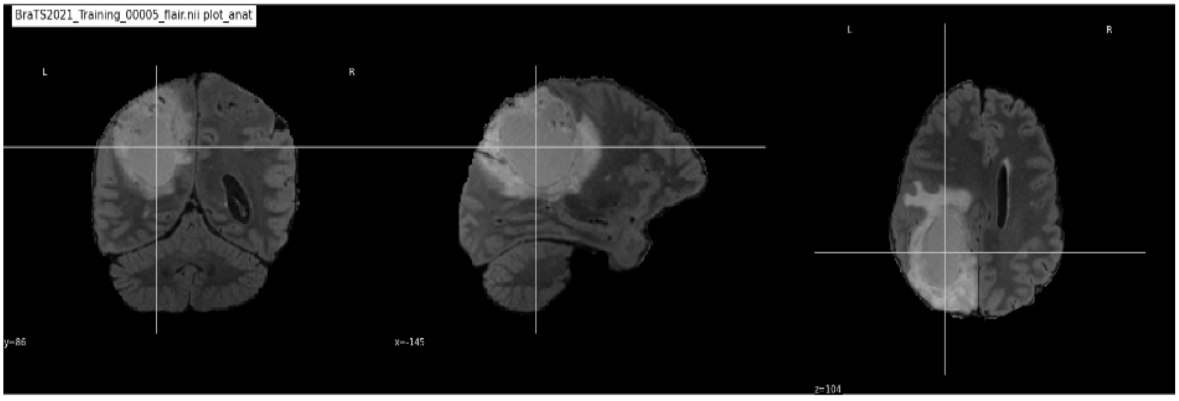


Figure 5.6: plotting of mask image.

The above figure shows the rotation of annotated of mask images at different pixels at 90 degrees.

```
niimg = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_flair.nii.gz')
nimask = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_seg.nii.gz')

fig, axes = plt.subplots(nrows=4, figsize=(30, 40))


nlplt.plot_anat(niimg,
                title='BraTS2021_Training_00005_flair.nii plot_anat',
                axes=axes[0])

nlplt.plot_epi(niimg,
               title='BraTS2021_Training_00005_flair.nii plot_epi',
               axes=axes[1])

nlplt.plot_img(niimg,
               title='BraTS2021_Training_00005_flair.nii plot_img',
               axes=axes[2])

nlplt.plot_roi(nimask,
               title='BraTS2021_Training_00005_flair.nii with mask plot_roi',
               bg_img=niimg,
               axes=axes[3], cmap='Paired')

plt.show()
```



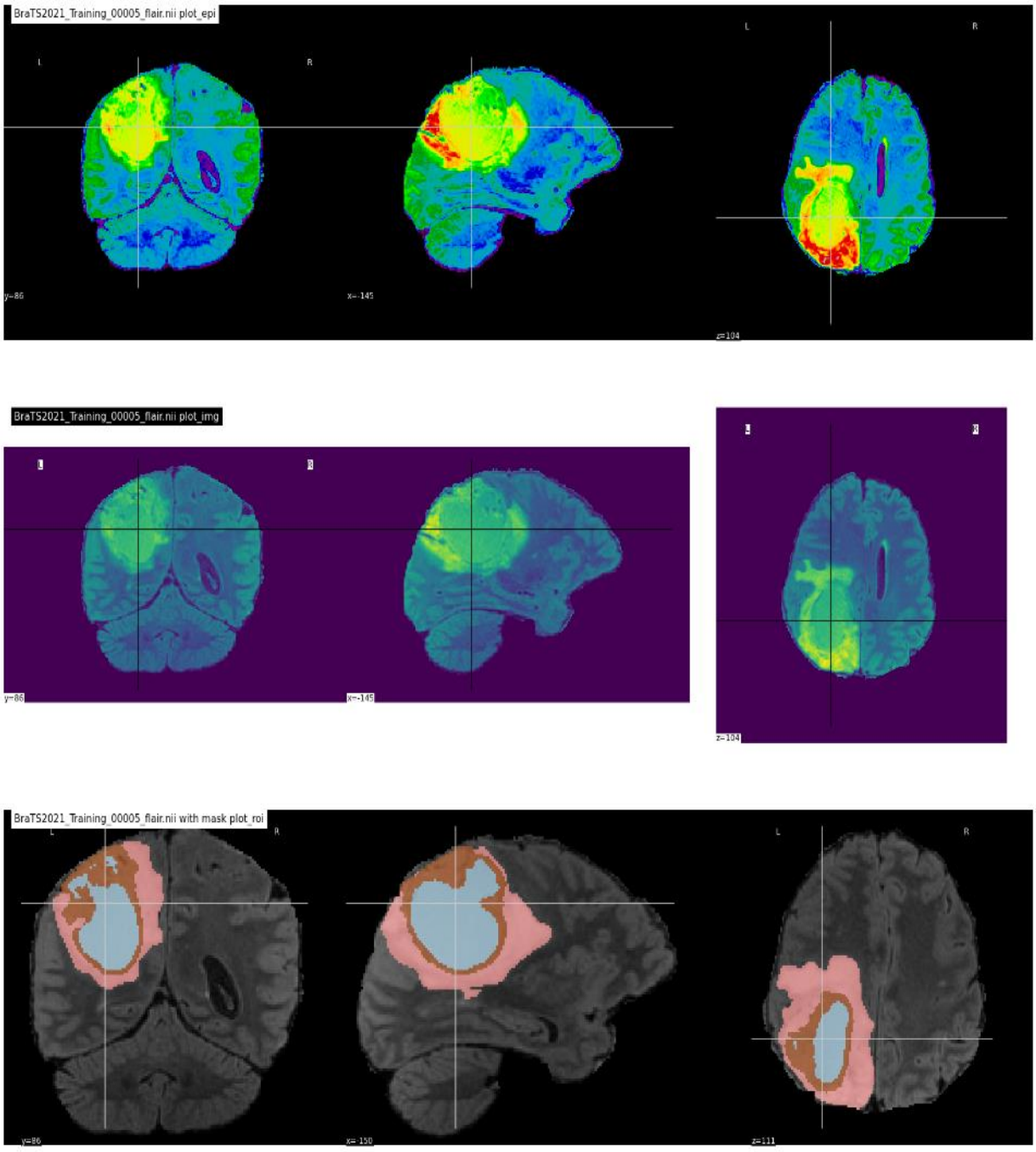


Figure 5.7: plotting the different annotations of flair images.

This step involves the plotting of all the annotated images at different rotation angles of flair, t1, and mask images.

Step 6: Defining the Evaluation method formulas.

This step involves deciding the evaluation methods like Dice Similarity Co-efficient, generating scores for the tumor sub-regions like Dice co-efficient of necrotic, Dice-coefficient of edema, and Dice co-efficient of enhancing tumor, sensitivity, precision, and specificity for Brain tumor segmentation by deriving their formulas for generating the scores for each method which help to compare the results of the modified model architectures.

```
# dice loss as defined above for 4 classes
def dice_coef(y_true, y_pred, epsilon=0.00001):
    """
    Dice = (2*|X & Y|)/ (|X|+ |Y|)
         =  2*sum(|A*B|)/(sum(A^2)+sum(B^2))
    ref: https://arxiv.org/pdf/1606.04797v1.pdf

    """
    axis = (0,1,2,3)
    dice_numerator = 2. * K.sum(y_true * y_pred, axis=axis) + epsilon
    dice_denominator = K.sum(y_true*y_true, axis=axis) + K.sum(y_pred*y_pred, axis=axis) + epsilon
    return K.mean((dice_numerator)/(dice_denominator))


# define per class evaluation of dice coef
# inspired by https://github.com/keras-team/keras/issues/9395
def dice_coef_necrotic(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,1] * y_pred[0,:,:,:,1]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,1])) + K.sum(K.square(y_pred[0,:,:,:,1])) + epsilon)

def dice_coef_edema(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,2] * y_pred[0,:,:,:,2]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,2])) + K.sum(K.square(y_pred[0,:,:,:,2])) + epsilon)

def dice_coef_enhancing(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,3] * y_pred[0,:,:,:,3]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,3])) + K.sum(K.square(y_pred[0,:,:,:,3])) + epsilon)

def Hausdorff_distance(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)

    print(type(y_pred_f))
    # first method
    d2_matrix = pairwise_distances(y_true_f,y_pred_f, metric ='euclidean')
    result = tf.math.reduce_mean(tf.math.minimum(d2_matrix, axis=0)) + tf.math.reduce_mean(tf.math.minimum(d2_matrix, axis=1))
    return result

# Computing Precision
def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision


# Computing Sensitivity
def sensitivity(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    return true_positives / (possible_positives + K.epsilon())


# Computing Specificity
def specificity(y_true, y_pred):
    true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
    possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
    return true_negatives / (possible_negatives + K.epsilon())
```

Figure 5.8: The formulas of evaluation methods chosen for the research project.

Step 7: Splitting the data.

```
import os
train_and_val_directories = [f.path for f in os.scandir(TRAIN_DATASET_PATH) if f.is_dir()]
```

```
def pathListIntoIds(dirList):
    x = []
    for i in range(0,len(dirList)):
        x.append(dirList[i][dirList[i].rfind('/')+1:])
    return x

train_and_test_ids = pathListIntoIds(train_and_val_directories);


train_test_ids, val_ids = train_test_split(train_and_test_ids,test_size=0.2)
train_ids, test_ids = train_test_split(train_test_ids,test_size=0.15)
```

```
train_and_test_ids[0]
```

```
'BraTS2021_00000'
```

```
train_ids[0]
```

```
'BraTS2021_01344'
```

This step involves splitting the training dataset into a train (0.65), test (0.15), and valid (0.20) for further training of a model and evaluating the scores based on the test generator and the valid generator.

5.3.4 Custom Data generator.

Step 8: Data Generating.

This step involves creating the custom data generator by cropping the image sizes to 128 and reshaping the images. Handling the large datasets for feeding it to the training of the model without causing memory space issues. And for this project, as we are dealing with the 3D U-net model, the images should be reshaped well so that they should not cause a memory error. So, in this project, the images are reshaped into (1, 128, 128, 128, 2). Defining the epochs, batch size, and n channels (feature channels) for data generator using Keras. And generating the data of each class image by doing the volumetric segmentation by slicing the images into 128 and loading all the images in the batches and the batch size =1 is chosen for training the model and generating the masks for it.

```
keras = tf.compat.v1.keras
Sequence = keras.utils.Sequence

class DataGenerator(Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, dim=(IMG_SIZE,IMG_SIZE), batch_size = 1, n_channels = 2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        # Find list of IDs
        Batch_ids = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(Batch_ids)

        return X, y

    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)

    def __data_generation(self, Batch_ids):
        'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
        # Initialization
        X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channels))
        y = np.zeros((self.batch_size*VOLUME_SLICES, 128, 128))
        Y = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, 4))


        # Generate data
        for c, i in enumerate(Batch_ids):
            case_path = os.path.join(TRAIN_DATASET_PATH, i)

            data_path = os.path.join(case_path, f'{i}_flair.nii.gz');
            flair = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_t1ce.nii.gz');
            ce = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_seg.nii.gz');
            seg = nib.load(data_path).get_fdata()

            for j in range(VOLUME_SLICES):
             X[j+(VOLUME_SLICES*c),:,:,0] = cv2.resize(flair[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))

             X[j+(VOLUME_SLICES*c),:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))


             y[j +VOLUME_SLICES*c,:,:] = cv2.resize(seg[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))

        X = X.reshape(1,128,128,128,2)
        y = y.reshape(1,128,128,128)
        # Generate masks
        y[y==4] = 3;
        y = tf.one_hot(y, 4);
        #Y = tf.image.resize(mask, (IMG_SIZE, IMG_SIZE));
        #Y = np.array(Y).reshape(1,128,128,128)
        return X/np.max(X), y

training_generator = DataGenerator(train_ids)
valid_generator = DataGenerator(val_ids)
test_generator = DataGenerator(test_ids)
```

```
print(len(train_ids))
print(len(val_ids))
print(len(test_ids))
```

```
850
251
150
```

```
def showDataLayout():
    plt.bar(["Train","Valid","Test"],
    [len(train_ids), len(val_ids), len(test_ids)], align='center',color=[ 'green','red', 'blue'])
    plt.legend()

    plt.ylabel('Number of images')
    plt.title('Data distribution')
    plt.savefig('data2021.png')
    plt.show()

showDataLayout()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is cal
led with no argument.
```

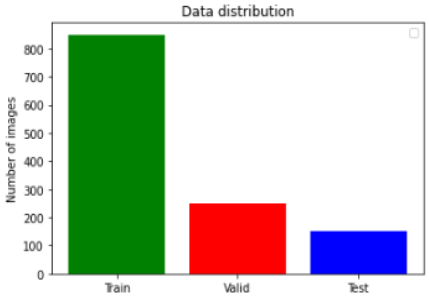


Figure 5.9: Distribution of data in the dataset.

From the above figure, the training dataset contains 1251 data, out of which the data is distributed into train generator containing 850 data, test generator containing 150 data, and valid generator containing 251 data.

5.3.5 Callback Functions.

Step 9: Adding callbacks.

```
csv_logger = CSVLogger('C:/Users/Computing/Desktop/archive/training.log', separator=',', append=False)


callbacks = [keras.callbacks.EarlyStopping(monitor='loss', min_delta=0,
                                patience=8, verbose=1, mode='auto'),
      keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                                patience=8, min_lr=0.000001, verbose=1),
      keras.callbacks.ModelCheckpoint(filepath = 'C:/Users/Computing/Desktop/archive/3D-UNet-2021model_.{epoch:02d}-{val_loss:.6f}.m5',
                               verbose=1, save_best_only=True, save_weights_only = True),
      csv_logger
    ]
```

Figure 5.10: Adding Callbacks

This step involves adding the callbacks for training the model to prevent overfitting, debugging the code, saving the model weights, checkpoints, generating the logs. These functions are executed during the training of the model by performing certain actions at the start or end of the epoch in the stages of the training. And used to write the TensorBoard logs after training the model for monitoring the metrics. In this project, the reduced learning rate is monitored on the validation loss and saves the best model weights and the patience given for this is 8 and early stopping is used on monitoring the loss for setting the patience at 8 for an overall 100 epochs of training.

5.3.6 Compiling the model.

```
kernel_initializer =  'he_uniform'
```

```
input_layer = Input((128,IMG_SIZE, IMG_SIZE, 2))

model1 = Unet_3d(input_layer)
model1.compile(loss="categorical_crossentropy", optimizer=Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
                                                                                                  dice_coef, precision, s
                                                                                                  dice_coef_edema ,dice_c


model1.summary()
```

Figure 5.11: Compiling the model

This step involves defining the loss function, choosing the metrics, and adding the optimizer. For this research, the he-uniform kernel initialization is chosen, the categorical crossentropy is chosen for loss using the Adam optimizer with a learning rate of 0.001and all the evaluation methods defined are chosen for the metrics. It is the final step in creating the model.

5.3.7 Training the model.

```
history =  model1.fit(training_generator,
                   epochs=100,
                   steps_per_epoch=len(train_ids),
                   callbacks= callbacks,
                   validation_data = valid_generator
                   )
model1.save("C:/Users/Computing/Desktop/archive/model_3dUnet_2mod.h5")
```

Figure 5.12: Training the model

This step involves the final part of training the model which consists of callbacks, number of epochs trained in this project is 100 for all the modified model architectures. It gives the weights and saves the model weights if there is an improvement in the performance of the model training.

5.3.8 Model Architecture layers

1) Convolution layers – It is the main building block of the Convolutional neural network (CNN) which converts all the pixels of the receptive field into a single value containing the set of parameters that helps in training the model. Each encoder and decoder path has this layer consisting of feature channels, 3*3*3 convolutions or 2*2*2 convolutions with the Activation ReLU function and he-uniform kernel initialization.
2) Convolution 3D Transpose – This layer is also called the Deconvolution layer which helps in reversing the operation of the convolution layer for getting the original output. Each decoder path has this layer consisting of feature channels, 3*3*3 convolutions or 2*2*2 convolutions with a stride of 2.
3) Max Pooling layers – It is a pooling process that selects the largest element from the area of the feature map which is covered by a filter. This layer consists of 2*2*2 pooling operations. The output contains prominent features after the max pooling operations are done on the previous feature map.
4) Activation function –it helps in the activation of neurons and uses simple mathematical operations for the prediction. Each convolution layer consists of a linear activation function ReLU and the final convolution layer consists of a Softmax activation function with the number of classes having 1*1*1 convolutions. It helps in preventing linearity.
5) Kernel Initialization – he-uniform initialization is used. It initializes the kernel data structure and from the hardware, the memory size is determined through the setup of BIOS.

6) Strides – All the modified architectures are trained with a stride of 2 in the decoder path of Convolution 3D Transpose which helps in modifying the movement over images.
7) Dropout –After each convolution layer the dropout layer is provided to prevent the overfitting of the model. It normally drops the units in the training phase.

5.3.8.1 Proposed Model 3D U-Net and Modified Model Architectures.

For 4 number of filters with Kernel size 3*3*3

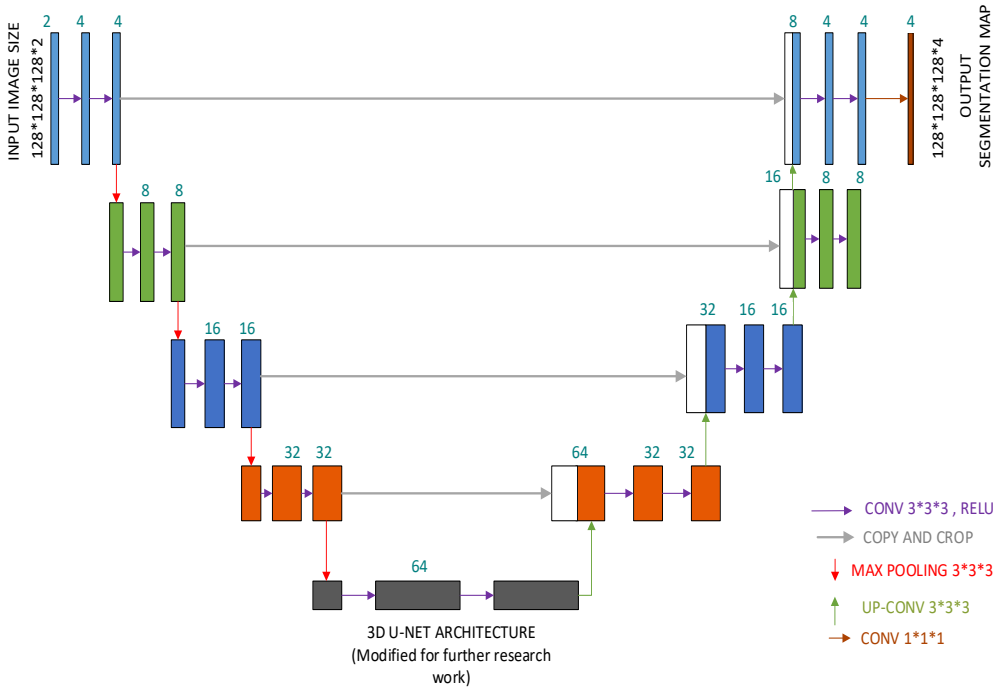


*Figure 5.13: 3D U-Net modified architectures started with 4 number of filters with Kernel size of 3*3*3 convolution.*

From the above figure, -The input image size of 128*128*128*2 is taken. The 3D U-Net model is further modified by tuning the hyperparameters by setting the number of filters to 4 in the first encoder block. In each encoder block, the number of filters is doubled and in each decoder block, the number of filters is halved. The convolution of 3*3*3 is used in each encoder and decoder block with the ReLU activation functions and the max pooling operation of 2*2*2 is used at the end of each encoding block. Dropout layers are used in each path to prevent overfitting and the dropouts of 0.1, 0.2, and 0.3 is given. The Convolution 3D Transpose consists of a stride 2. The final layer consists of 1*1*1 convolution with the Softmax activation function.

Similarly for other modifications, the number of filters has been changed by giving 4, 8, 16, and 32. The convolution of 3*3*3 and 2*2*2 has been given in each convolution layer, each encoder and decoder block is trained with the ReLU activation functions. In decoder path, each decoder block is concatenated with the encoder block. The dropouts of 0.1, 0.2, and 0.3 are used for encoder and decoder blocks. The 2*2*2 Max pooling operation is done after each encoding block, final layer consists of 1*1*1 convolution and number of classes with the Softmax activation function. Every decoder block consists of the Convolution 3D Transpose containing a stride of 2. Each modified 3D U-Net Architecture layer code work is justified in the Appendices section.

3D U-Net modification (started with 4 number of filters and (Kernel sizes- 3*3*3))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN, and Dropout (0.1)	1	4×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	4×128×128×128
	Encoder block 2	Conv3D, ReLU, BN, and Dropout (0.1)	1	8×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	8×64×64×64
	Encoder block 3	Conv3D, ReLU, BN, and Dropout (0.2)	1	16×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×32×32×32
	Encoder block 4	Conv3D, ReLU, BN, and Dropout (0.2)	1	32×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, and Dropout (0.3)	1	64×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	32×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	32×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	16×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	16×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	8×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	8×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	4×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	4×128×128×128

	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 2: The 3D U-Net network structure started with 4 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 3×3×3.

For 8 number of filters with Kernel size 3*3*3

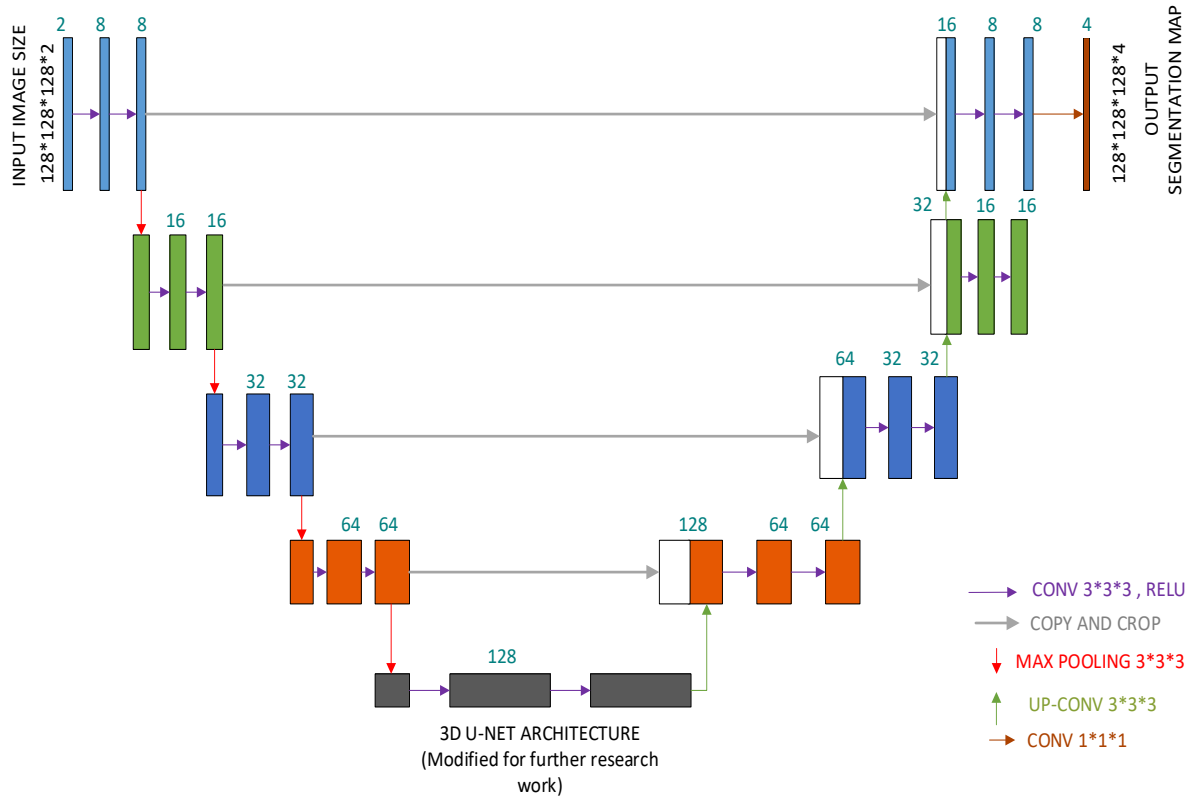


*Figure 5.14: 3D U-Net modified architectures started with 8 number of filters with Kernel size of 3*3*3 convolution.*

3D U-Net modification (started with 8 number of filters and (Kernel sizes- 3*3*3))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN, and Dropout (0.1)	1	8×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	8×128×128×128
	Encoder block 2	Conv3D, ReLU, BN, and Dropout (0.1)	1	16×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×64×64×64

	Encoder block 3	Conv3D, ReLU, BN, and Dropout (0.2)	1	32×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×32×32×32
	Encoder block 4	Conv3D, ReLU, BN, and Dropout (0.2)	1	64×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	64×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, and Dropout (0.3)	1	128×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	64×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	64×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	32×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	32×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	16×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	16×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	8×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	8×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 3: The 3D U-Net network structure started with 8 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 3×3×3.

For 16 number of filters with Kernel size 3*3*3

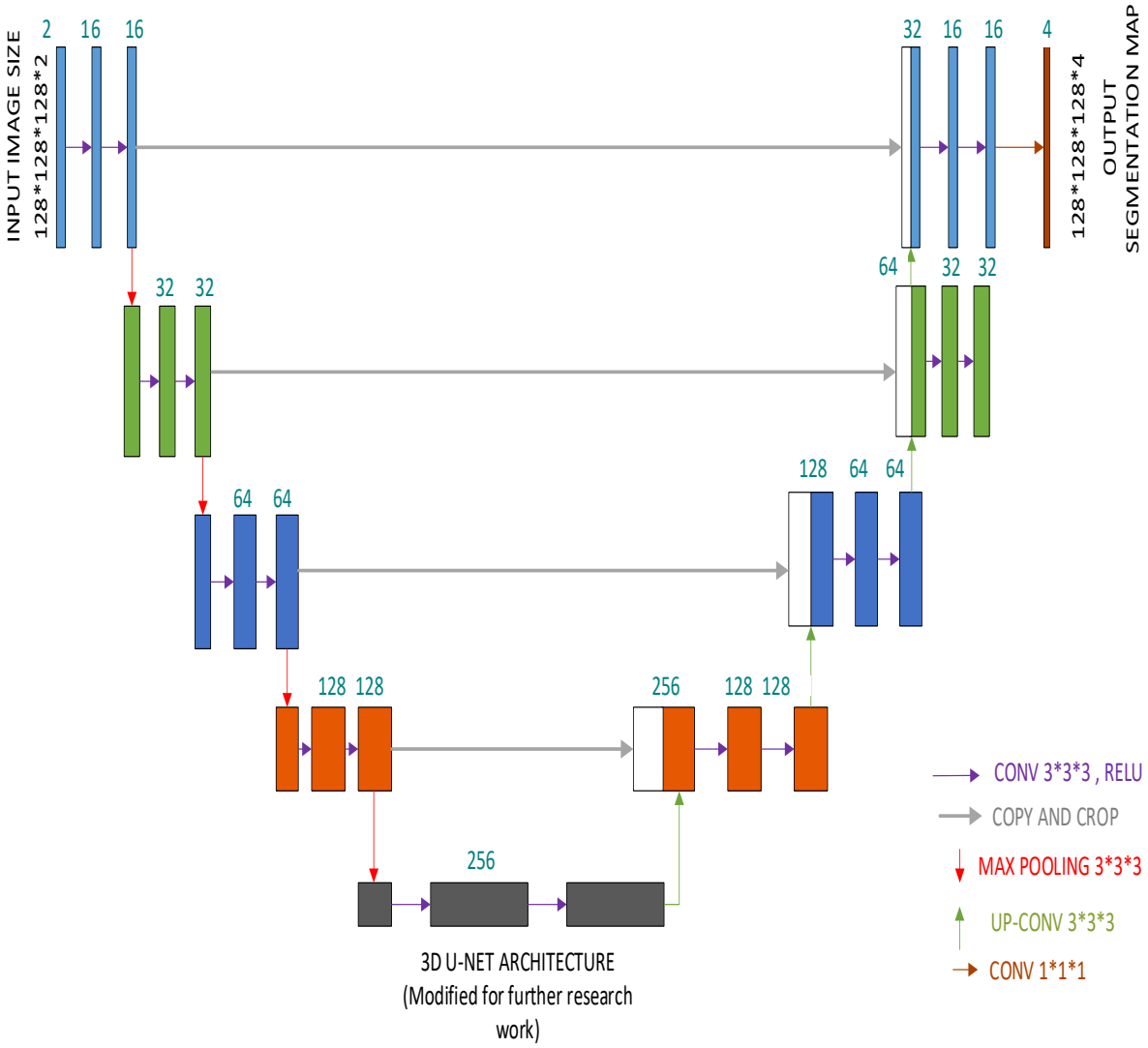


*Figure 5.15: 3D U-Net modified architectures started with 16 number of filters with Kernel size of 3*3*3 convolution.*

3D U-Net modification (started with 16 number of filters and (Kernel sizes- 3*3*3))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN, and Dropout (0.1)	1	16×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×128×128×128
	Encoder block 2	Conv3D, ReLU, BN, and Dropout (0.1)	1	32×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×64×64×64

	Encoder block 3	Conv3D, ReLU, BN, and Dropout (0.2)	1	64×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	64×32×32×32
	Encoder block 4	Conv3D, ReLU, BN, and Dropout (0.2)	1	128×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	128×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, and Dropout (0.3)	1	256×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	128×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	128×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	64×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D, and Dropout (0.2)	1	64×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	32×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	32×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	16×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D, and Dropout (0.1)	1	16×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 4: The 3D U-Net network structure started with 16 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 3×3×3.

For 32 number of filters with Kernel size 3*3*3

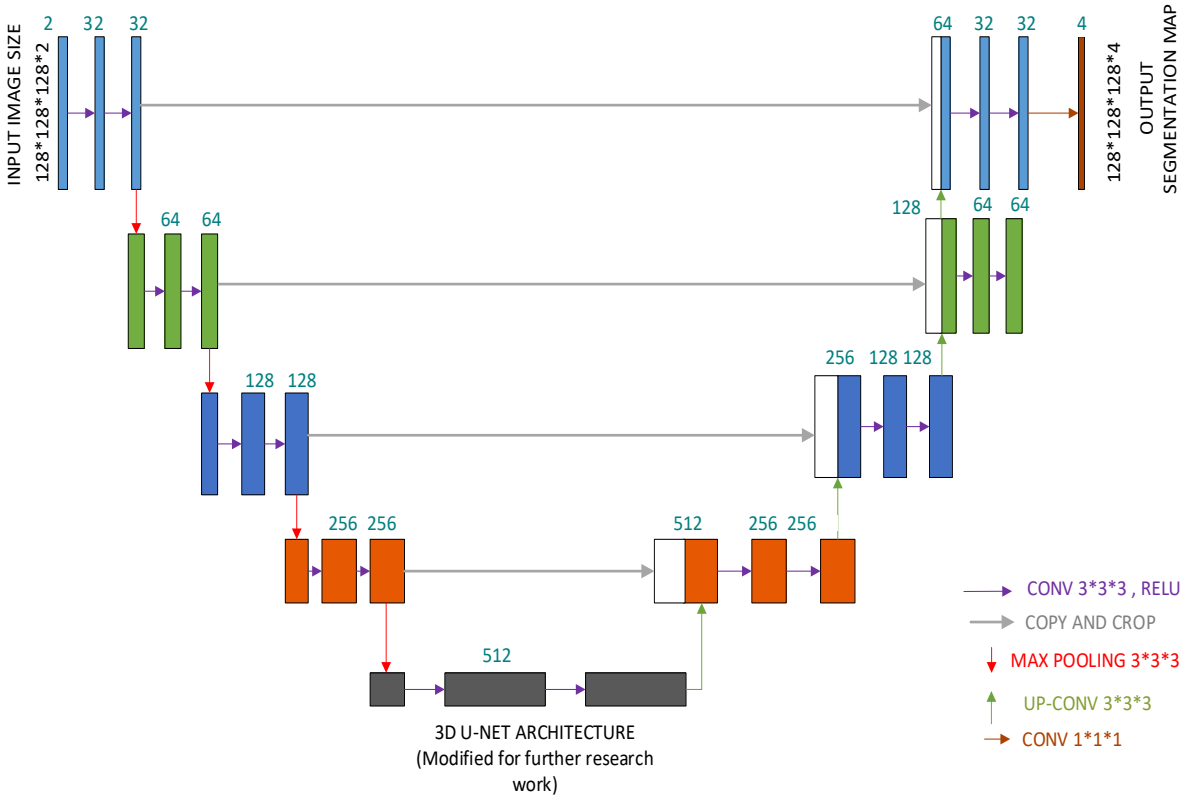


*Figure 5.16: 3D U-Net modified architectures started with 32 number of filters with Kernel size of 3*3*3 convolution.*

3D U-Net modification (started with 32 number of filters and (Kernel sizes- 3*3*3))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN	1	32×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×128×128×128
	Encoder block 2	Conv3D, ReLU, BN	1	64×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	64×64×64×64
	Encoder block 3	Conv3D, ReLU, BN	1	128×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	128×32×32×32
	Encoder block 4	Conv3D, ReLU, BN	1	256×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	256×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, Dropout (0.3)	1	512×8×8×8

Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	256×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D	1	256×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	128×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D	1	128×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	64×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D	1	64×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	32×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D	1	32×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 5: The 3D U-Net network structure started with 32 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 3×3×3.

For 4 number of filters with Kernel size 2*2*2

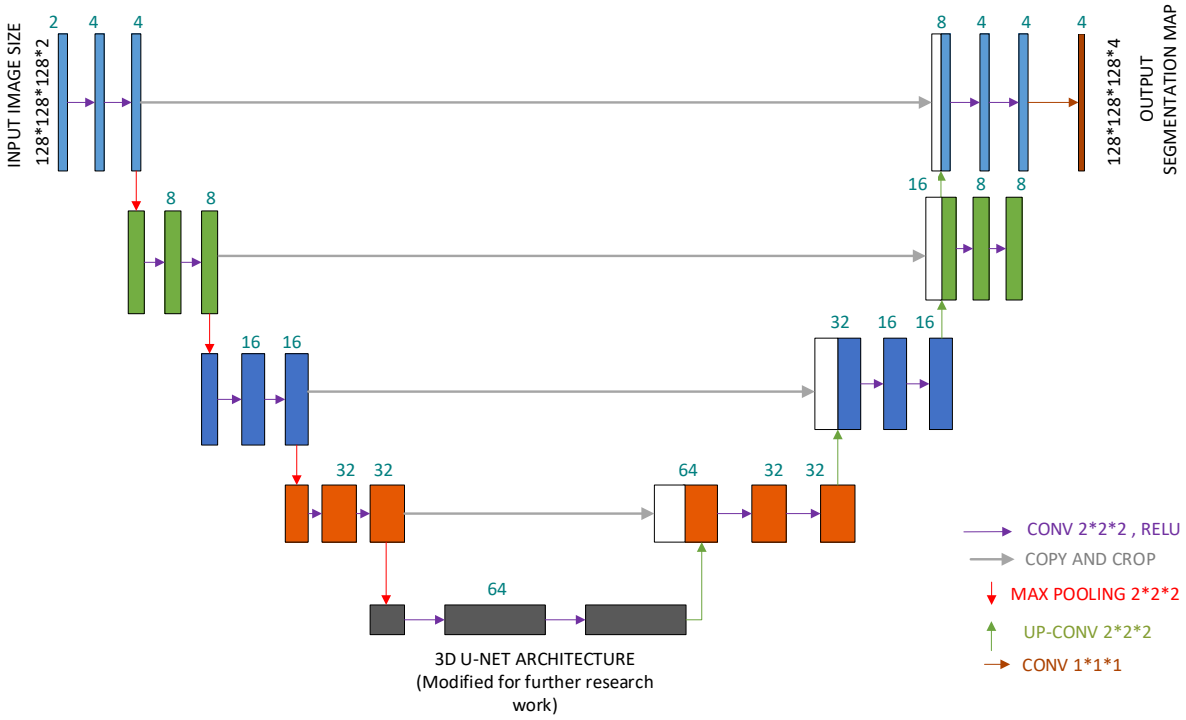


*Figure 5.17: 3D U-Net modified architectures started with 4 number of filters with Kernel size of 2*2*2 convolution.*

3D U-Net modification (started with 4 number of filters and (Kernel sizes- 2*2*2))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN	1	4×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	4×128×128×128
	Encoder block 2	Conv3D, ReLU, BN	1	8×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	8×64×64×64
	Encoder block 3	Conv3D, ReLU, BN	1	16×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×32×32×32
	Encoder block 4	Conv3D, ReLU, BN	1	32×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, Dropout (0.3)	1	64×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	32×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D	1	32×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	16×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D	1	16×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	8×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D	1	8×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	4×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D	1	4×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 6: The 3D U-Net network structure started with 4 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 2×2×2.

For 8 number of filters with Kernel size 2*2*2

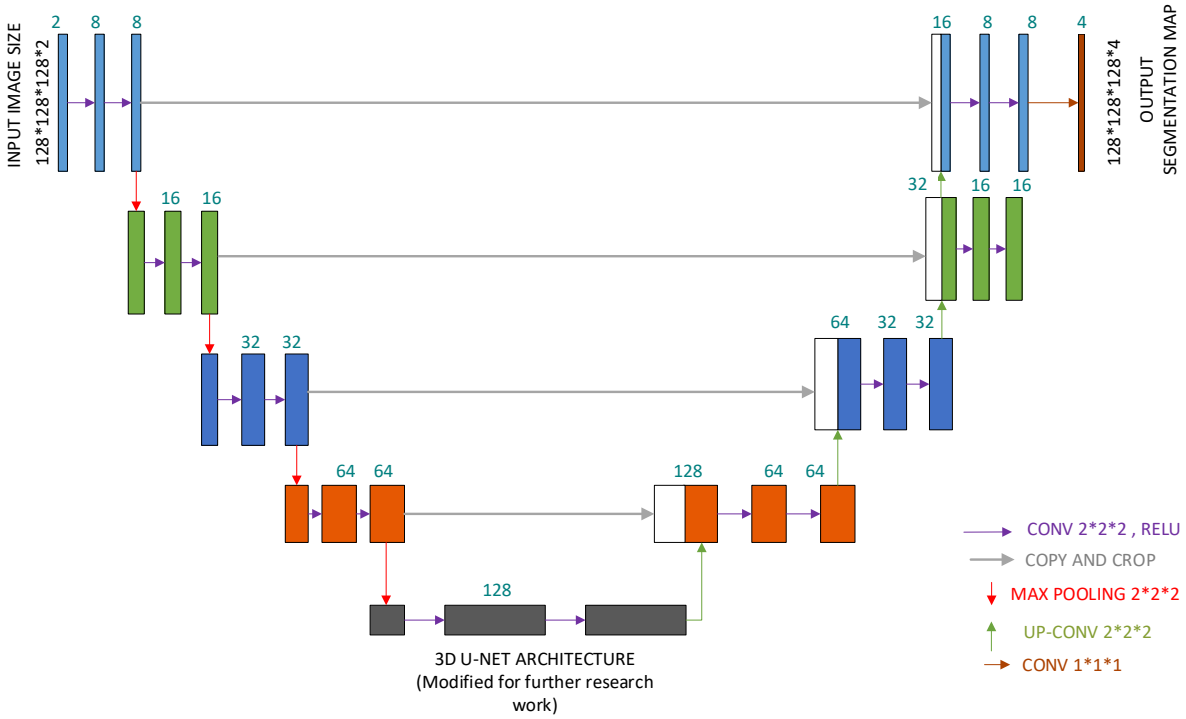


*Figure 5.18: 3D U-Net modified architectures started with 8 number of filters with Kernel size of 2*2*2 convolution.*

3D U-Net modification (started with 8 number of filters and (Kernel sizes- 2*2*2))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN	1	8×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	8×128×128×128
	Encoder block 2	Conv3D, ReLU, BN	1	16×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×64×64×64
	Encoder block 3	Conv3D, ReLU, BN	1	32×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×32×32×32
	Encoder block 4	Conv3D, ReLU, BN	1	64×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	64×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, Dropout (0.3)	1	128×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	64×16×16×16

	Decoder block 4	Conv3D, ReLU, BN, Conv3D	1	64×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	32×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D	1	32×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	16×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D	1	16×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	8×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D	1	8×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 7: The 3D U-Net network structure started with 8 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 2×2×2.

For 16 number of filters with Kernel size 2*2*2

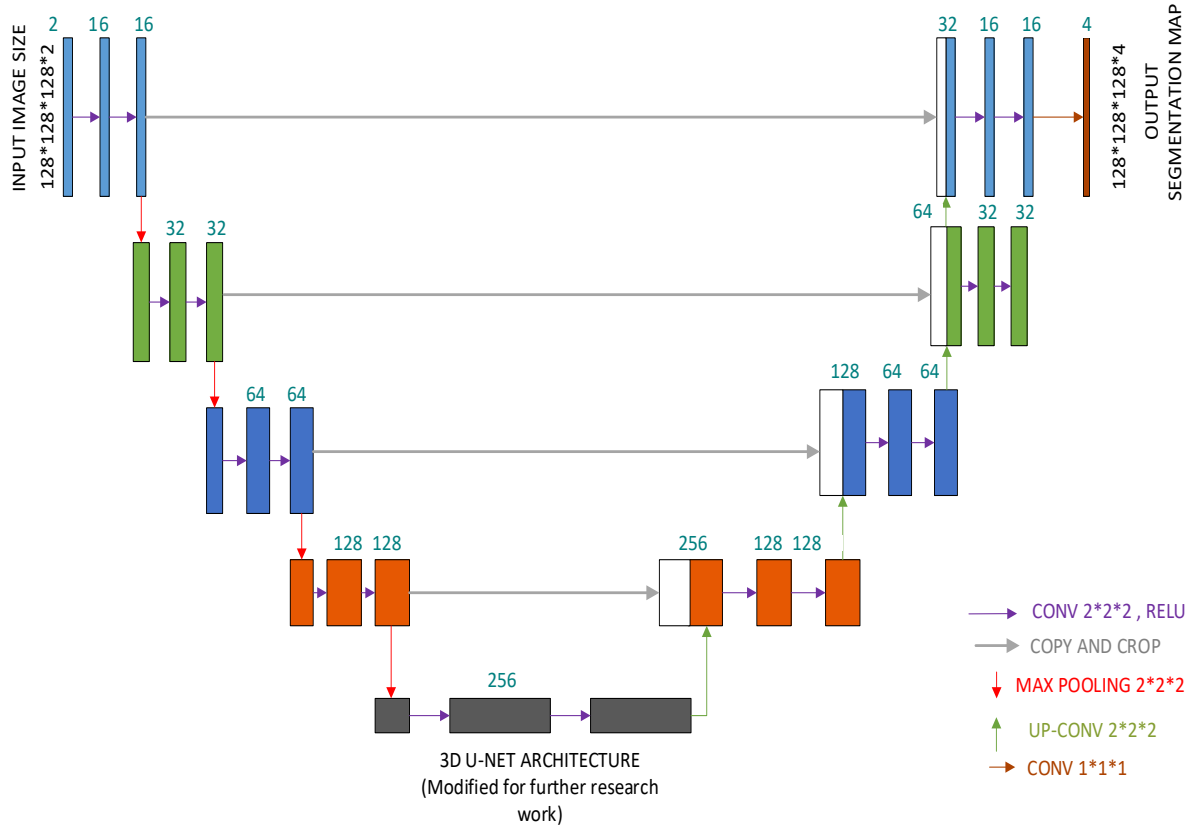


*Figure 5.19: 3D U-Net modified architectures started with 16 number of filters with Kernel size of 2*2*2 convolution.*

3D U-Net modification (started with 16 number of filters and (Kernel sizes- 2*2*2))				
Encoder part i.e. Contraction path.	Name	Details	Repeat	Size
	Input			2×128×128×128
	Encoder block 1	Conv3D, ReLU, BN	1	16×128×128×128
	Encoder Down Sampling 1	Conv3D, ReLU, BN, and MaxPooling 3D	1	16×128×128×128
	Encoder block 2	Conv3D, ReLU, BN	1	32×64×64×64
	Encoder Down Sampling 2	Conv3D, ReLU, BN, and MaxPooling 3D	1	32×64×64×64
	Encoder block 3	Conv3D, ReLU, BN	1	64×32×32×32
	Encoder Down Sampling 3	Conv3D, ReLU, BN, and MaxPooling 3D	1	64×32×32×32
	Encoder block 4	Conv3D, ReLU, BN	1	128×16×16×16
	Encoder Down Sampling 4	Conv3D, ReLU, BN, and MaxPooling 3D	1	128×16×16×16
	Encoder block 5	Conv3D, ReLU, BN, Dropout (0.3)	1	256×8×8×8
Decoder part i.e. Expansion path.	Decoder Up Sampling 4	Conv3DTranspose, strides, Encoder block 4.	1	128×16×16×16
	Decoder block 4	Conv3D, ReLU, BN, Conv3D	1	128×16×16×16
	Decoder Up Sampling 3	Conv3DTranspose, strides, Encoder block 3.	1	64×32×32×32
	Decoder block 3	Conv3D, ReLU, BN, Conv3D	1	64×32×32×32
	Decoder Up Sampling 2	Conv3DTranspose, strides, Encoder block 2.	1	32×64×64×64
	Decoder block 2	Conv3D, ReLU, BN, Conv3D	1	32×64×64×64
	Decoder Up Sampling 1	Conv3DTranspose, strides, Encoder block 1.	1	16×128×128×128
	Decoder block 1	Conv3D, ReLU, BN, Conv3D	1	16×128×128×128
	End Conv	Conv1D	1	4×128×128×128
	Softmax	Softmax	1	4×128×128×128

Table 8: The 3D U-Net network structure started with 16 number of filters in the first convolution layers of Conv3D - 3×3×3 convolution, Conv1D- 1×1×1, BN - Batch normalization, strides- 2×2×2, Conv3DTranspose – deconvolution with a kernel size of 2×2×2.

6. Experiments and Evaluation

This section briefs about the experiments carried out in training the model with different modified 3D U-Net architectures and evaluating the results with the evaluation methods chosen for evaluating the Brain tumor segmentation of the BraTS dataset for the research project. And an analysis of the results will be done based on the final generated scores of the evaluation methods of modified architectures of the 3D U-Net model.

The evaluation methods chosen for the research work are Dice Similarity co-efficient, Dice co-efficient of edema region, Dice co-efficient of the necrotic region, Dice co-efficient of enhancing tumor region, Specificity, Precision, and Sensitivity are considered.

6.1 Results of experiment.

```
# hist=history.history

acc=hist['accuracy']
val_acc=hist['val_accuracy']

epoch=range(len(acc))

loss=hist['loss']
val_loss=hist['val_loss']

train_dice=hist['dice_coef']
val_dice=hist['val_dice_coef']

f,ax=plt.subplots(1,4,figsize=(16,8))

ax[0].plot(epoch,acc,'b',label='Training Accuracy')
ax[0].plot(epoch,val_acc,'r',label='Validation Accuracy')
ax[0].legend()

ax[1].plot(epoch,loss,'b',label='Training Loss')
ax[1].plot(epoch,val_loss,'r',label='Validation Loss')
ax[1].legend()

ax[2].plot(epoch,train_dice,'b',label='Training dice coef')
ax[2].plot(epoch,val_dice,'r',label='Validation dice coef')
ax[2].legend()

ax[3].plot(epoch,hist['mean_io_u_5'],'b',label='Training mean IOU')
ax[3].plot(epoch,hist['val_mean_io_u_5'],'r',label='Validation mean IOU')
ax[3].legend()


plt.savefig('training_result_2021.png')
plt.show()
```

Figure 6.1: plotting the graphs for training and validation of Accuracy, Loss, Dice Similarity of co-efficient, and MeanIoU.

Results and Evaluation of Brain tumor segmentation for BraTS 2021 Dataset.

By loading the saved model.hdf5 (or) mod.h5 file and evaluating the results.

1) For 3D U-Net model having <u>16 number of filters</u> with <u>kernel size of (3, 3, 3).</u>

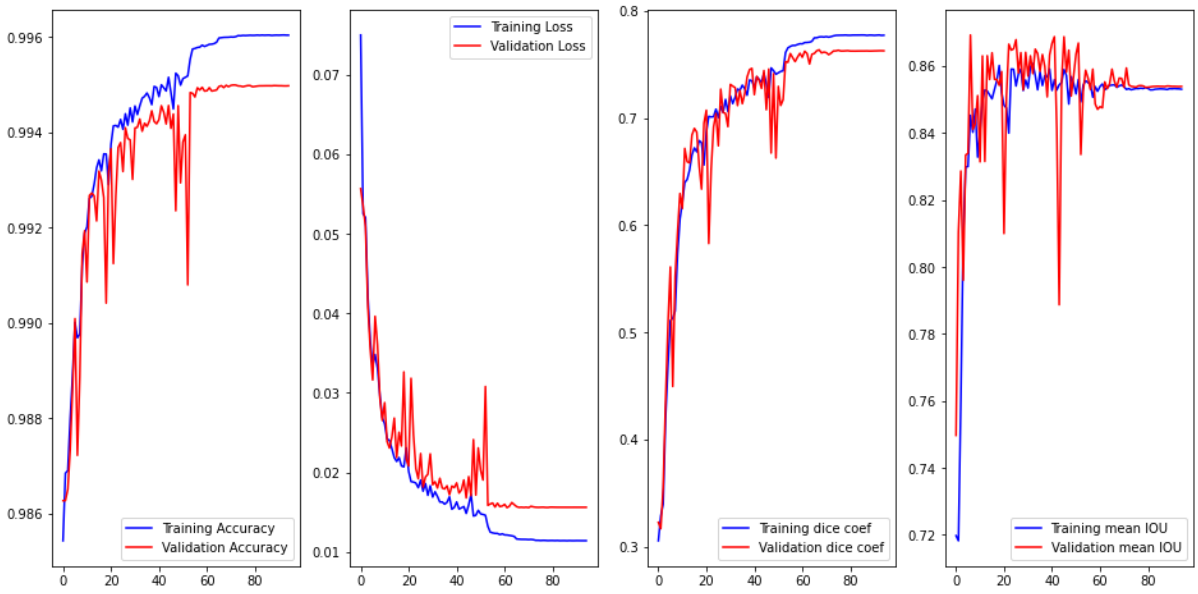


Figure 6.2. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 16 number of filters with kernel size of (3, 3, 3).

When the batch_size = 1

```
In [66]:
print("Evaluate on test data")
results = model1.evaluate(test_generator, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 41s 271ms/step - loss: 0.0138 - accuracy: 0.9955 - mean_io_u_5: 0.8558 - dice_coef: 0.7611 - precision: 0.99
65 - sensitivity: 0.9947 - specificity: 0.9989 - dice_coef_necrotic: 0.5071 - dice_coef_edema: 0.7794 - dice_coef_enhancing: 0.7592
test loss, test acc: [0.013767251744866371, 0.9955047369003296, 0.8557811975479126, 0.7611231803894043, 0.9965457916259766, 0.9946977496147156, 0.99885
15973091125, 0.5070666670799255, 0.7793629169464111, 0.7592341303825378]

In [67]:
print("Evaluate on test data")
results = model1.evaluate(valid_generator, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
251/251 [==============================] - 61s 243ms/step - loss: 0.0156 - accuracy: 0.9950 - mean_io_u_5: 0.8539 - dice_coef: 0.7626 - precision: 0.99
61 - sensitivity: 0.9941 - specificity: 0.9987 - dice_coef_necrotic: 0.5124 - dice_coef_edema: 0.7422 - dice_coef_enhancing: 0.7971
test loss, test acc: [0.015625623986124992, 0.9949766397476196, 0.8539320230484009, 0.7626184821128845, 0.9961360692977905, 0.9941028952598572, 0.99871
56987190247, 0.5123515725135803, 0.7422458529472351, 0.7971230149269104]
```

When the batch_size = 100

```
In [164…
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 44s 239ms/step - loss: 0.0115 - accuracy: 0.9961 - mean_io_u_5: 0.8538 - dice_coef: 0.7683 - precision: 0.99
71 - sensitivity: 0.9953 - specificity: 0.9990 - dice_coef_necrotic: 0.5288 - dice_coef_edema: 0.7520 - dice_coef_enhancing: 0.7932
test loss, test acc: [0.011514394544064999, 0.996078610420227, 0.853848397731781, 0.7682787179946899, 0.9971142411231995, 0.9952689409255981, 0.9990405
440330505, 0.5287842750549316, 0.7520086765289307, 0.7932301759719849]

In [166…
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
# Evaluate the model on the test data using `evaluate`
print("Evaluate on val data")
results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 60s 234ms/step - loss: 0.0120 - accuracy: 0.9960 - mean_io_u_7: 0.8543 - dice_coef: 0.7807 - precision: 0.99
70 - sensitivity: 0.9952 - specificity: 0.9990 - dice_coef_necrotic: 0.5365 - dice_coef_edema: 0.7690 - dice_coef_enhancing: 0.8184
val loss, val acc: [0.011950459331274033, 0.9960130453109741, 0.8542948961257935, 0.7807278633117676, 0.9970437288284302, 0.9952036142349243, 0.9990173
578262329, 0.5364580154418945, 0.7690145373344421, 0.8184013366699219]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0750	0.9854	0.0557	0.9863
10	0.0269	0.9919	0.0265	0.9919
20	0.0231	0.9929	0.0216	0.9932
30	0.0183	0.9942	0.0223	0.9930
40	0.0156	0.9949	0.0181	0.9942
50	0.0152	0.9950	0.0231	0.9929
60	0.0123	0.9958	0.0158	0.9949
70	0.0116	0.9960	0.0156	0.9950
80	0.0114	0.9960	0.0156	0.9950
90	0.0114	0.9960	0.0156	0.9950
Early Stopping = 95	0.0114	0.9960	0.0156	0.9950

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.7198	0.3055	0.0577	0.1280	0.0461	0.9834	0.9785	0.9958
10	0.8328	0.6056	0.3174	0.5024	0.6052	0.9938	0.9903	0.9979
20	0.8521	0.6560	0.3883	0.5855	0.6524	0.9947	0.9914	0.9982
30	0.8558	0.7060	0.4480	0.6598	0.7178	0.9957	0.9930	0.9986
40	0.8592	0.7347	0.4884	0.6979	0.7536	0.9963	0.9939	0.9988
50	0.8548	0.7407	0.4942	0.7069	0.7632	0.9963	0.9939	0.9988
60	0.8525	0.7692	0.5341	0.7538	0.7900	0.9969	0.9949	0.9990
70	0.8544	0.7755	0.5433	0.7649	0.7946	0.9970	0.9952	0.9990
80	0.8536	0.7771	0.5457	0.7678	0.7959	0.9971	0.9952	0.9990
90	0.8533	0.7772	0.5452	0.7679	0.7967	0.9971	0.9952	0.9990
Early Stopping = 95	0.8531	0.7772	0.5456	0.7678	0.7961	0.9971	0.9952	0.9990

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch [Early stopping=95]	0.0138	0.9955	0.0137	0.9955	0.8558	0.7611
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.5071	0.7794	0.7592	0.9965	0.9947	0.9989	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early stopping=95]	0.0115	0.9961	0.0115	0.9960	0.8538	0.7683
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.5288	0.7520	0.7932	0.9971	0.9953	0.9990	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early stopping=95]	0.0156	0.9950	0.0156	0.9949	0.8539	0.7626
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.5124	0.7422	0.7971	0.9961	0.9941	0.9987	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early stopping=95]	0.0120	0.9960	0.0119	0.99601	0.8543	0.7807
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.5365	0.7690	0.8184	0.9970	0.9952	0.9990	

2) For 3D U-Net model having <u>8 number of filters</u> with <u>kernel size of (3, 3, 3).</u>

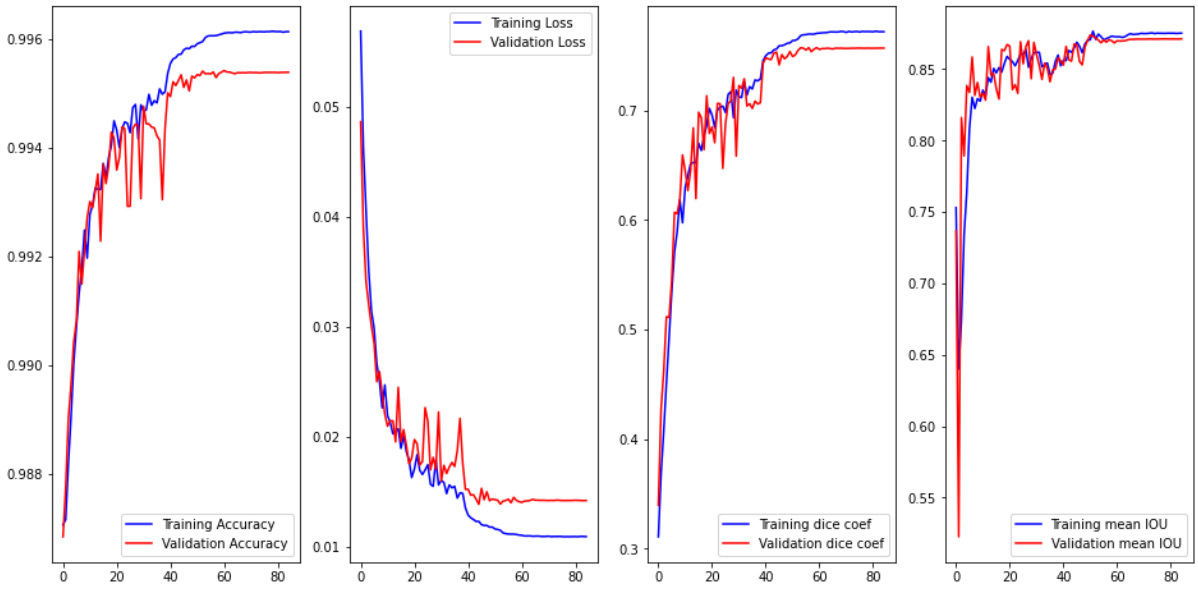


Figure 6.3. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 8 number of filters with kernel size of (3, 3, 3).

When the batch_size = 1

```
In [104…
print("Evaluate on test data")
results = model1.evaluate(test_generator, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 37s 246ms/step - loss: 0.0160 - accuracy: 0.9948 - mean_io_u_5: 0.8694 - dice_coef: 0.7315 - precision: 0.9
960 - sensitivity: 0.9940 - specificity: 0.9987 - dice_coef_necrotic: 0.4813 - dice_coef_edema: 0.7333 - dice_coef_enhancing: 0.7124
test loss, test acc: [0.015970751643180847, 0.9948136210441589, 0.8694289922714233, 0.7314507365226746, 0.9959919452667236, 0.99396151304245, 0.998668
1342124939, 0.48131272196769714, 0.7333204746246338, 0.7123534083366394]

In [98]:
print("Evaluate on val data")
results = model1.evaluate(valid_generator, callbacks= callbacks)
print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 62s 247ms/step - loss: 0.0142 - accuracy: 0.9954 - mean_io_u_5: 0.8713 - dice_coef: 0.7569 - precision: 0.9
965 - sensitivity: 0.9946 - specificity: 0.9988 - dice_coef_necrotic: 0.5035 - dice_coef_edema: 0.7607 - dice_coef_enhancing: 0.7645
val loss, val acc: [0.01416836865246296, 0.9953837394714355, 0.8712506294250488, 0.7569038271903992, 0.9964852333068848, 0.9945586323738098, 0.9988313
317298889, 0.5034840106964111, 0.760688841342926, 0.7644568681716919]
```

<u>Evaluation scores generated during the training:</u>

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0569	0.9871	0.0486	0.9868
10	0.0247	0.9920	0.0220	0.9927
20	0.0163	0.9945	0.0180	0.9942
30	0.0156	0.9948	0.0222	0.9931
40	0.0135	0.9954	0.0152	0.9950
50	0.0118	0.9959	0.0143	0.9953
60	0.0110	0.9961	0.0141	0.9954
70	0.0109	0.9961	0.0142	0.9954
80	0.0109	0.9961	0.0142	0.9954
Early stopping = 85	0.0109	0.9961	0.0142	0.9954

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.7531	0.3108	0.0671	0.1370	0.0452	0.9871	0.9870	0.9957
10	0.8276	0.5974	0.2915	0.5372	0.5631	0.9942	0.9902	0.9981
20	0.8590	0.7019	0.4195	0.6932	0.6962	0.9960	0.9933	0.9987
30	0.8615	0.7186	0.4502	0.7138	0.7116	0.9963	0.9936	0.9988
40	0.8524	0.7454	0.4866	0.7486	0.7475	0.9967	0.9943	0.9989
50	0.8702	0.7617	0.5124	0.7766	0.7587	0.9971	0.9949	0.9990
60	0.8728	0.7705	0.5254	0.7921	0.7651	0.9972	0.9952	0.9991
70	0.8751	0.7720	0.5275	0.7939	0.7674	0.9973	0.9952	0.9991
80	0.8753	0.7720	0.5276	0.7940	0.7673	0.9973	0.9952	0.9991
Early stopping = 85	0.8753	0.7719	0.5276	0.7939	0.7669	0.9973	0.9952	0.9991

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch [Early stopping=85]	0.0160	0.9948	0.0159	0.9948	0.8694	0.7315
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4813	0.7333	0.7124	0.9960	0.9940	0.9987	

Overall evaluated scores generated on the **Validation data.**

When Batch size = 1

Epoch	**Loss**	**Accuracy**	**Validation loss**	**Validation Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch [Early stopping=85]	0.0142	0.9954	0.0141	0.9953	0.8713	0.7569
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.5035	0.7607	0.7645	0.9965	0.9946	0.9988	

3) For 3D U-Net model having 4 number of filters with kernel size of (3, 3, 3).

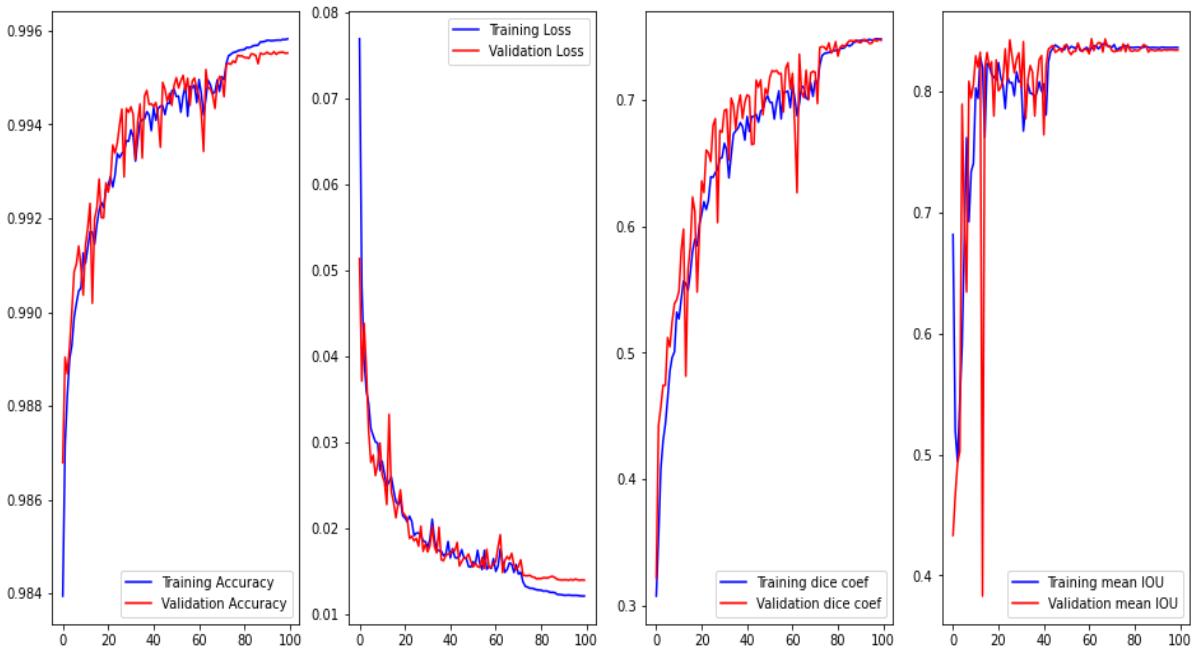


Figure 6.4. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 4 number of filters with kernel size of (3, 3, 3).

When the batch_size = 1

```
In [64]: print("Evaluate on test data")
         results = model1.evaluate(test_generator, callbacks= callbacks)
         print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 40s 266ms/step - loss: 0.0158 - accuracy: 0.9952 - mean_io_u_2: 0.8357 - dice_coef: 0.7175 - precision: 0.9
963 - sensitivity: 0.9944 - specificity: 0.9988 - dice_coef_necrotic: 0.3893 - dice_coef_edema: 0.7502 - dice_coef_enhancing: 0.7316
test loss, test acc: [0.015774782747030258, 0.9951964020729065, 0.8357417583465576, 0.7174946069717407, 0.9962745308876038, 0.9943706393241882, 0.9987
611770629883, 0.38926824927330017, 0.7502455115318298, 0.7316102385520935]

In [65]: print("Evaluate on val data")
         results = model1.evaluate(valid_generator, callbacks= callbacks)
         print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 62s 244ms/step - loss: 0.0139 - accuracy: 0.9955 - mean_io_u_2: 0.8344 - dice_coef: 0.7468 - precision: 0.9
966 - sensitivity: 0.9947 - specificity: 0.9989 - dice_coef_necrotic: 0.4723 - dice_coef_edema: 0.7623 - dice_coef_enhancing: 0.7535
val loss, val acc: [0.013919276185333729, 0.9955261945724487, 0.8343731164932251, 0.7467852830886841, 0.996623694896698, 0.9946653246879578, 0.9988774
657249451, 0.4722719192504883, 0.7623094320297241, 0.7535340785980225]
```

When the batch_size = 100

```
In [71]: model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
         # Evaluate the model on the test data using `evaluate`
         print("Evaluate on test data")
         results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
         print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 38s 242ms/step - loss: 0.0158 - accuracy: 0.9952 - mean_io_u_4: 0.8357 - dice_coef: 0.7175 - precision: 0.9
963 - sensitivity: 0.9944 - specificity: 0.9988 - dice_coef_necrotic: 0.3893 - dice_coef_edema: 0.7502 - dice_coef_enhancing: 0.7316
test loss, test acc: [0.015774786472320557, 0.9951962232589722, 0.8357417583465576, 0.717494547367096, 0.9962742328643799, 0.9943709373474121, 0.99876
15942955017, 0.3892682194709778, 0.7502455115318298, 0.731610119342804]

In [72]: model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
         # Evaluate the model on the test data using `evaluate`
         print("Evaluate on val data")
         results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
         print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 63s 246ms/step - loss: 0.0139 - accuracy: 0.9955 - mean_io_u_5: 0.8344 - dice_coef: 0.7468 - precision: 0.9
966 - sensitivity: 0.9947 - specificity: 0.9989 - dice_coef_necrotic: 0.4723 - dice_coef_edema: 0.7623 - dice_coef_enhancing: 0.7535
val loss, val acc: [0.01391927432268858, 0.9955262541770935, 0.8343730568885803, 0.7467852234840393, 0.9966233372688293, 0.9946650862693787, 0.9988773
465156555, 0.4722718894481659, 0.7623094916343689, 0.7535343766212463]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0769	0.9839	0.0513	0.9868
10	0.0267	0.9913	0.0299	0.9904
20	0.0214	0.9926	0.0218	0.9928
30	0.0184	0.9936	0.0181	0.9943
40	0.0184	0.9939	0.0169	0.9944
50	0.0154	0.9948	0.0161	0.9948
60	0.0164	0.9945	0.0158	0.9948
70	0.0157	0.9947	0.0151	0.9950
80	0.0128	0.9956	0.0141	0.9955
90	0.0122	0.9958	0.0139	0.9955
100	0.0121	0.9958	0.0139	0.9955

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.6818	0.3079	0.0725	0.1120	0.0569	0.9835	0.9783	0.9955
10	0.7406	0.5320	0.2361	0.5132	0.3810	0.9941	0.9891	0.9980
20	0.8096	0.6015	0.2780	0.5963	0.5333	0.9950	0.9909	0.9983
30	0.8080	0.6542	0.3423	0.6463	0.6295	0.9955	0.9922	0.9985
40	0.8003	0.6680	0.3653	0.6591	0.6489	0.9956	0.9925	0.9985
50	0.8346	0.7030	0.4129	0.7050	0.6954	0.9962	0.9936	0.9987
60	0.8364	0.6936	0.4042	0.6882	0.6834	0.9960	0.9933	0.9987
70	0.8373	0.7025	0.4153	0.7063	0.6896	0.9962	0.9935	0.9987
80	0.8364	0.7387	0.4655	0.7511	0.7393	0.9968	0.9946	0.9989
90	0.8364	0.7464	0.4766	0.7613	0.7485	0.9969	0.9949	0.9990
100	0.8364	0.7477	0.4781	0.7630	0.7506	0.9970	0.9949	0.9990

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch	0.0158	0.9952	0.0157	0.9951	0.8357	0.7175
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.3893	0.7502	0.7316	0.9963	0.9944	0.9988	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0158	0.9952	0.0157	0.9951	0.8357	0.7175
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.3893	0.7502	0.7316	0.9963	0.9944	0.9988	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0139	0.9955	0.0139	0.9955	0.8344	0.7468
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.472	0.7623	0.7535	0.9966	0.9947	0.9989	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0139	0.9955	0.0139	0.9955	0.8344	0.7468
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4723	0.7623	0.7535	0.9966	0.9947	0.9989	

4) For 3D U-Net model having 32 number of filters with kernel size of (3, 3, 3).

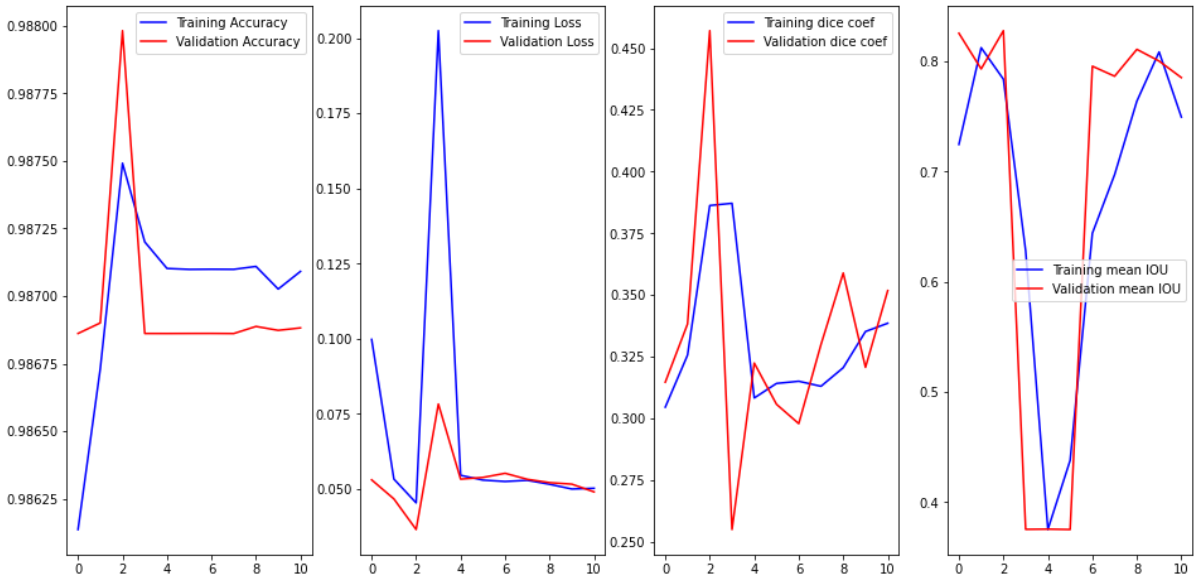


Figure 6.5. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 32 number of filters with kernel size of (3, 3, 3).

When the batch_size = 1

```
In [36]: print("Evaluate on test data")
         results = model1.evaluate(test_generator, callbacks= callbacks)
         print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 39s 257ms/step - loss: 0.0507 - accuracy: 0.9864 - mean_io_u: 0.7848 - dice_coef: 0.3503 - precision: 0.9871
- sensitivity: 0.9864 - specificity: 0.9957 - dice_coef_necrotic: 0.0656 - dice_coef_edema: 0.2046 - dice_coef_enhancing: 0.1364
test loss, test acc: [0.05070583522319794, 0.9864477515220642, 0.7847846746444702, 0.35029444098472595, 0.9871368408203125, 0.9864104986190796, 0.99571
48432731628, 0.06563331931829453, 0.20455127954483032, 0.1364289751361847]

In [37]: print("Evaluate on val data")
         results = model1.evaluate(valid_generator, callbacks= callbacks)
         print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 62s 248ms/step - loss: 0.0491 - accuracy: 0.9869 - mean_io_u: 0.7851 - dice_coef: 0.3518 - precision: 0.9873
- sensitivity: 0.9868 - specificity: 0.9958 - dice_coef_necrotic: 0.0657 - dice_coef_edema: 0.2116 - dice_coef_enhancing: 0.1350
val loss, val acc: [0.04908156022429466, 0.9868813157081604, 0.7851289510726929, 0.3517545163631439, 0.9873476624488831, 0.9868322014808655, 0.99578464
03121948, 0.06566378474235535, 0.21161355078220367, 0.1349981725215912]
```

When the batch_size = 100

```
In [43]: model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
         # Evaluate the model on the test data using `evaluate`
         print("Evaluate on test data")
         results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
         print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 38s 244ms/step - loss: 0.0507 - accuracy: 0.9864 - mean_io_u_2: 0.7848 - dice_coef: 0.3503 - precision: 0.98
71 - sensitivity: 0.9864 - specificity: 0.9957 - dice_coef_necrotic: 0.0656 - dice_coef_edema: 0.2046 - dice_coef_enhancing: 0.1364
test loss, test acc: [0.05070582777261734, 0.9864476323127747, 0.7847846746444702, 0.3502945005893707, 0.9871370196342468, 0.9864106178283691, 0.995714
9028778076, 0.06563333421945572, 0.20455121994018555, 0.13642887771129608]

In [44]: model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(nu
         # Evaluate the model on the test data using `evaluate`
         print("Evaluate on val data")
         results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
         print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 65s 257ms/step - loss: 0.0491 - accuracy: 0.9869 - mean_io_u_3: 0.7851 - dice_coef: 0.3518 - precision: 0.98
73 - sensitivity: 0.9868 - specificity: 0.9958 - dice_coef_necrotic: 0.0657 - dice_coef_edema: 0.2116 - dice_coef_enhancing: 0.1350
val loss, val acc: [0.04908156394958496, 0.9868820905685425, 0.7851290106773376, 0.3517542779445648, 0.9873482584953308, 0.9868326783180237, 0.99578475
95214844, 0.06566379964351654, 0.21161338686943054, 0.1349981278181076]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0998	0.9861	0.0531	0.9869
2	0.0534	0.9867	0.0468	0.9869
3	0.0454	0.9875	0.0366	0.9880
4	0.2025	0.9872	0.0783	0.9869
5	0.0546	0.9871	0.0533	0.9869
6	0.0530	0.9871	0.0539	0.9869
7	0.0526	0.9871	0.0553	0.9869
8	0.0529	0.9871	0.0533	0.9869
9	0.0516	0.9871	0.0522	0.9869
10	0.0500	0.9870	0.0517	0.9869
Early stopping = 11	0.0503	0.9871	0.0491	0.9869

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.7245	0.3045	0.0514	0.1363	0.0458	0.9839	0.9650	0.9957
2	0.8123	0.3256	0.0671	0.1610	0.0800	0.9872	0.9861	0.9957
3	0.7837	0.3862	0.1174	0.2361	0.1960	0.9902	0.9854	0.9967
4	0.6282	0.3871	0.1164	0.2258	0.2119	0.9894	0.9855	0.9965
5	0.3753	0.3082	0.0555	0.1222	0.0609	0.9873	0.9869	0.9958
6	0.4380	0.3141	0.0602	0.1310	0.0709	0.9876	0.9868	0.9959
7	0.6440	0.3150	0.0608	0.1327	0.0721	0.9876	0.9866	0.9959
8	0.6972	0.3130	0.0587	0.1320	0.0668	0.9876	0.9868	0.9959
9	0.7638	0.3205	0.0589	0.1447	0.0839	0.9876	0.9868	0.9959
10	0.8084	0.3352	0.0625	0.1646	0.1190	0.9877	0.9867	0.9959
Early stopping = 11	0.7494	0.3385	0.0631	0.1632	0.1332	0.9876	0.9865	0.9959

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch [Early Stopping = 11]	0.0507	0.9864	0.0507	0.9864	0.7848	0.3503
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.0656	0.2046	0.1364	0.9871	0.9864	0.9957	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early Stopping = 11]	0.0507	0.9864	0.0507	0.9864	0.7848	0.3503
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.0656	0.2046	0.1364	0.9871	0.9864	0.9957	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early Stopping = 11]	0.0491	0.9869	0.0490	0.9868	0.7851	0.3518
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.0657	0.2116	0.1350	0.9873	0.9868	0.9958	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch [Early Stopping = 11]	0.0491	0.9869	0.0490	0.9868	0.7851	0.3518
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.0657	0.2116	0.1350	0.9873	0.9868	0.9958	

5) For 3D U-Net model having <u>16 number of filters</u> with <u>kernel size of (2, 2, 2).</u>

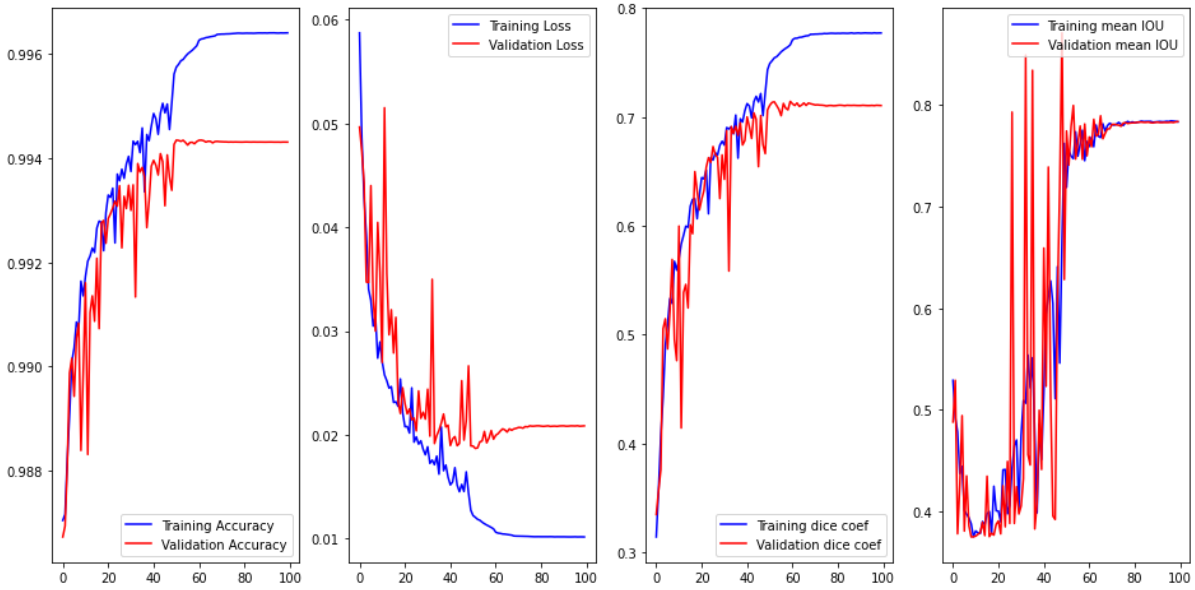


Figure 6.6. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 16 number of filters with kernel size of (2, 2, 2).

When the batch_size = 1

```
In [316]:
print("Evaluate on test data")
results = model1.evaluate(test_generator, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 39s 257ms/step - loss: 0.0182 - accuracy: 0.9946 - mean_io_u_17: 0.7828 - dice_coef: 0.7134 - precision: 0.9
957 - sensitivity: 0.9939 - specificity: 0.9986 - dice_coef_necrotic: 0.4518 - dice_coef_edema: 0.6950 - dice_coef_enhancing: 0.7080
test loss, test acc: [0.018159989267587662, 0.9946404099464417, 0.7827781438827515, 0.713378369808197, 0.995714008808136, 0.9938835501670837, 0.9985752
105712891, 0.4518028199672699, 0.6949742436408997, 0.7079852819442749]

In [317]:
print("Evaluate on val data")
results = model1.evaluate(valid_generator, callbacks= callbacks)
print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 64s 252ms/step - loss: 0.0209 - accuracy: 0.9943 - mean_io_u_17: 0.7837 - dice_coef: 0.7105 - precision: 0.9
954 - sensitivity: 0.9935 - specificity: 0.9985 - dice_coef_necrotic: 0.4365 - dice_coef_edema: 0.6851 - dice_coef_enhancing: 0.7219
val loss, val acc: [0.020855408161878586, 0.9943078756332397, 0.7837415337562561, 0.7105005979537964, 0.9953800439834595, 0.9935299754142761, 0.9984638
690948486, 0.43653178215026855, 0.6850897669792175, 0.7218950986862183]
```

When the batch_size = 100

```
In [322]:
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
print("test loss, test acc:", results)

Evaluate on test data
150/150 [==============================] - 39s 252ms/step - loss: 0.0182 - accuracy: 0.9946 - mean_io_u_19: 0.7828 - dice_coef: 0.7134 - precision: 0.9
957 - sensitivity: 0.9939 - specificity: 0.9986 - dice_coef_necrotic: 0.4518 - dice_coef_edema: 0.6950 - dice_coef_enhancing: 0.7080
test loss, test acc: [0.018159985542297363, 0.9946408271789551, 0.7827781438827515, 0.7133783102035522, 0.9957138299942017, 0.9938830733299255, 0.99857
51509666443, 0.45180273056030273, 0.6949741244316101, 0.7079851031303406]

In [323]:
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on val data")
results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
print("val loss, val acc:", results)

Evaluate on val data
251/251 [==============================] - 64s 252ms/step - loss: 0.0209 - accuracy: 0.9943 - mean_io_u_20: 0.7837 - dice_coef: 0.7105 - precision: 0.9
954 - sensitivity: 0.9935 - specificity: 0.9985 - dice_coef_necrotic: 0.4365 - dice_coef_edema: 0.6851 - dice_coef_enhancing: 0.7219
val loss, val acc: [0.020855406299233437, 0.994307816028595, 0.7837415337562561, 0.7105004787445068, 0.9953799843788147, 0.9935302138328552, 0.99846363
06762695, 0.43653178215026855, 0.6850895881652832, 0.7218949794769287]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0587	0.9871	0.0497	0.9867
10	0.0289	0.9914	0.0357	0.9899
20	0.0221	0.9930	0.0245	0.9924
30	0.0181	0.9940	0.0215	0.9935
40	0.0159	0.9946	0.0209	0.9938
50	0.0127	0.9956	0.0189	0.9943
60	0.0109	0.9962	0.0196	0.9943
70	0.0102	0.9964	0.0206	0.9943
80	0.0102	0.9964	0.0209	0.9943
90	0.0102	0.9964	0.0209	0.9943
100	0.0101	0.9964	0.0209	0.9943

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.5292	0.3142	0.0626	0.1386	0.0614	0.9873	0.9867	0.9958
10	0.3754	0.5593	0.2283	0.4970	0.5143	0.9935	0.9897	0.9978
20	0.4008	0.6282	0.3095	0.5928	0.6123	0.9948	0.9915	0.9983
30	0.4016	0.6777	0.3786	0.6599	0.6738	0.9956	0.9928	0.9985
40	0.4514	0.7057	0.4199	0.6963	0.7077	0.9961	0.9935	0.9987
50	0.7623	0.7434	0.4773	0.7530	0.7442	0.9968	0.9947	0.9989
60	0.7641	0.7663	0.5099	0.7899	0.7659	0.9973	0.9953	0.9991
70	0.7821	0.7759	0.5268	0.8034	0.7741	0.9974	0.9956	0.9991
80	0.7835	0.7767	0.5276	0.8043	0.7756	0.9974	0.9956	0.9991
90	0.7834	0.7772	0.5285	0.8047	0.7762	0.9974	0.9956	0.9992
100	0.7834	0.7771	0.5285	0.8048	0.7760	0.9974	0.9956	0.9992

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch	0.0182	0.9946	0.0181	0.9946	0.7828	0.7134
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4518	0.6950	0.7080	0.9957	0.9939	0.9986	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0182	0.9946	0.0181	0.9946	0.7828	0.7134
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4518	0.6950	0.7080	0.9957	0.9939	0.9986	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0209	0.9943	0.0208	0.9940	0.7837	0.7105
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4365	0.6851	0.7219	0.9954	0.9935	0.9985	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0209	0.9943	0.0208	0.9943	0.7837	0.7105
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4365	0.6851	0.7219	0.9954	0.9935	0.9985	

6) For 3D U-Net model having 8 number of filters with kernel size of (2, 2, 2).

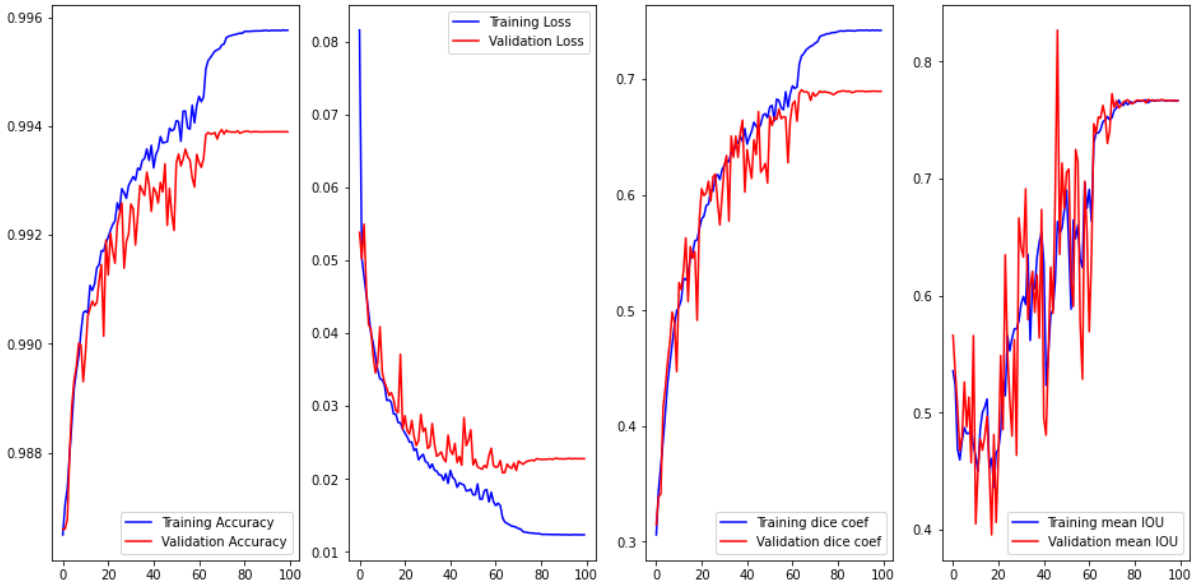


Figure 6.7. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 8 number of filters with kernel size of (2, 2, 2).

When the batch_size = 1

```
print("Evaluate on test data")
results = model1.evaluate(test_generator, callbacks= callbacks)
print("test loss, test acc:", results)
```

```
Evaluate on test data
150/150 [==============================] - 38s 251ms/step - loss: 0.0201 - accuracy: 0.9943 - mean_io_u: 0.7647 - dice_coef: 0.6831 - precision: 0.9954
- sensitivity: 0.9934 - specificity: 0.9985 - dice_coef_necrotic: 0.4123 - dice_coef_edema: 0.6551 - dice_coef_enhancing: 0.6665
test loss, test acc: [0.020077688619494438, 0.9942690134048462, 0.7646974325180054, 0.6831079125404358, 0.9953976273536682, 0.9934458136558533, 0.99846
98295593262, 0.4122549295425415, 0.6551347970962524, 0.6665340662002563]
```

```
print("Evaluate on val data")
results = model1.evaluate(valid_generator, callbacks= callbacks)
print("val loss, val acc:", results)
```

```
Evaluate on val data
251/251 [==============================] - 64s 252ms/step - loss: 0.0228 - accuracy: 0.9939 - mean_io_u: 0.7667 - dice_coef: 0.6893 - precision: 0.9951
- sensitivity: 0.9930 - specificity: 0.9984 - dice_coef_necrotic: 0.4075 - dice_coef_edema: 0.6575 - dice_coef_enhancing: 0.6940
val loss, val acc: [0.022776633501052856, 0.9938965439796448, 0.7667243480682373, 0.689336359500885, 0.9950680136680603, 0.9930272102355957, 0.99836015
70129395, 0.4074677526950836, 0.657456636428833, 0.6940410137176514]
```

When the batch_size = 100

```
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
print("test loss, test acc:", results)
```

```
Evaluate on test data
150/150 [==============================] - 38s 251ms/step - loss: 0.0201 - accuracy: 0.9943 - mean_io_u_3: 0.7647 - dice_coef: 0.6831 - precision: 0.99
54 - sensitivity: 0.9934 - specificity: 0.9985 - dice_coef_necrotic: 0.4123 - dice_coef_edema: 0.6551 - dice_coef_enhancing: 0.6665
test loss, test acc: [0.020077699795365334, 0.9942691326141357, 0.7646974325180054, 0.6831079721450806, 0.99539715051651, 0.9934466481208801, 0.9984697
699546814, 0.4122548997402191, 0.6551348567008972, 0.6665341854095459]
```

```
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on val data")
results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
print("val loss, val acc:", results)
```

```
Evaluate on val data
251/251 [==============================] - 63s 248ms/step - loss: 0.0228 - accuracy: 0.9939 - mean_io_u_4: 0.7667 - dice_coef: 0.6893 - precision: 0.99
51 - sensitivity: 0.9930 - specificity: 0.9984 - dice_coef_necrotic: 0.4075 - dice_coef_edema: 0.6575 - dice_coef_enhancing: 0.6940
val loss, val acc: [0.022776614874601364, 0.9938969016075134, 0.7667244076728821, 0.6893362998962402, 0.9950679540634155, 0.9930274486541748, 0.9983600
378036499, 0.4074676036834717, 0.657456636428833, 0.6940407752990723]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0816	0.9865	0.0538	0.9866
10	0.0337	0.9906	0.0409	0.9893
20	0.0268	0.9919	0.0268	0.9919
30	0.0224	0.9929	0.0270	0.9920
40	0.0194	0.9936	0.0260	0.9924
50	0.0186	0.9939	0.0267	0.9921
60	0.0169	0.9944	0.0218	0.9935
70	0.0134	0.9954	0.0212	0.9939
80	0.0125	0.9957	0.0227	0.9939
90	0.0123	0.9958	0.0228	0.9939
100	0.0123	0.9958	0.0228	0.9939

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.5358	0.3060	0.0608	0.1148	0.0595	0.9859	0.9734	0.9958
10	0.4719	0.4996	0.1686	0.4274	0.4056	0.9925	0.9891	0.9975
20	0.4659	0.5706	0.2335	0.5141	0.5370	0.9938	0.9904	0.9979
30	0.5784	0.6217	0.2973	0.5809	0.6103	0.9946	0.9916	0.9982
40	0.6540	0.6571	0.3465	0.6299	0.6536	0.9952	0.9925	0.9984
50	0.6728	0.6666	0.3631	0.6429	0.6619	0.9954	0.9928	0.9985
60	0.6749	0.6874	0.3891	0.6777	0.6840	0.9958	0.9933	0.9986
70	0.7502	0.7292	0.4464	0.7432	0.7280	0.9966	0.9945	0.9989
80	0.7648	0.7402	0.4640	0.7603	0.7372	0.9968	0.9948	0.9990
90	0.7664	0.7421	0.4663	0.7633	0.7398	0.9969	0.9949	0.9990
100	0.7666	0.7421	0.4666	0.7634	0.7391	0.9969	0.9949	0.9990

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch	0.0201	0.9943	0.0200	0.9942	0.7647	0.6831
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4123	0.6551	0.6665	0.9954	0.9934	0.9985	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0201	0.9943	0.0200	0.9942	0.7647	0.6831
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4123	0.6551	0.6665	0.9954	0.9934	0.9985	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0228	0.9939	0.0227	0.9938	0.7667	0.6893
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4075	0.6575	0.6940	0.9951	0.9930	0.9984	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0228	0.9939	0.0227	0.9938	0.7667	0.6893
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.4075	0.6575	0.6940	0.9951	0.9930	0.9984	

7) For 3D U-Net model having 4 number of filters with kernel size of (2, 2, 2).

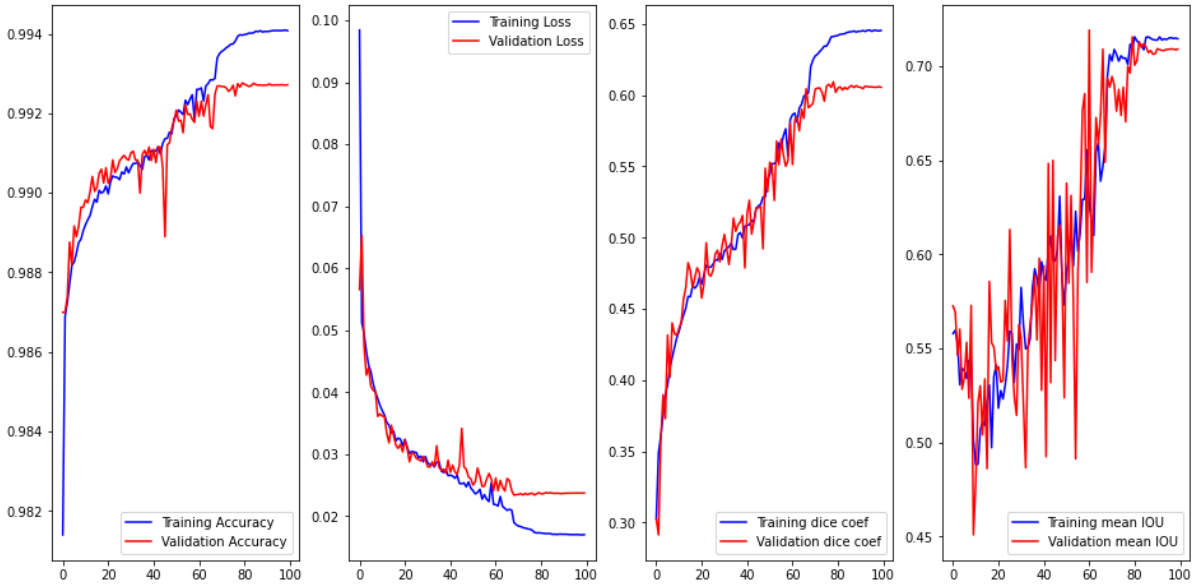


Figure 6.8. Training and validation curves for Accuracy, Loss, dice co-efficient and Mean IOU scores for 3D U-Net model having 4 number of filters with kernel size of (2, 2, 2).

When the batch_size = 1

```
print("Evaluate on test data")
results = model1.evaluate(test_generator, callbacks= callbacks)
print("test loss, test acc:", results)
```

```
Evaluate on test data
150/150 [==============================] - 38s 254ms/step - loss: 0.0220 - accuracy: 0.9931 - mean_io_u: 0.7160 - dice_coef: 0.6099 - precision: 0.9951
- sensitivity: 0.9916 - specificity: 0.9984 - dice_coef_necrotic: 0.2676 - dice_coef_edema: 0.6217 - dice_coef_enhancing: 0.5520
test loss, test acc: [0.022028518840670586, 0.9931288361549377, 0.7159587144851685, 0.6099130511283875, 0.9950692653656006, 0.991640031337738, 0.998364
1505241394, 0.267586886882782, 0.6216508746147156, 0.5519809722900391]
```

```
print("Evaluate on val data")
results = model1.evaluate(valid_generator, callbacks= callbacks)
print("val loss, val acc:", results)
```

```
Evaluate on val data
251/251 [==============================] - 60s 237ms/step - loss: 0.0237 - accuracy: 0.9927 - mean_io_u: 0.7092 - dice_coef: 0.6056 - precision: 0.9947
- sensitivity: 0.9912 - specificity: 0.9982 - dice_coef_necrotic: 0.2720 - dice_coef_edema: 0.6075 - dice_coef_enhancing: 0.5445
val loss, val acc: [0.023715598508715 63, 0.99271559715271, 0.7091755867004395, 0.6055861115455627, 0.9946848154067993, 0.9912353754043579, 0.9982356429
100037, 0.2719831466674804 7, 0.6075194478034973, 0.5445340275764465]
```

When the batch_size = 100

```
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_generator, batch_size=100, callbacks= callbacks)
print("test loss, test acc:", results)
```

```
Evaluate on test data
150/150 [==============================] - 36s 236ms/step - loss: 0.0220 - accuracy: 0.9931 - mean_io_u_2: 0.7160 - dice_coef: 0.6099 - precision: 0.99
51 - sensitivity: 0.9916 - specificity: 0.9984 - dice_coef_necrotic: 0.2676 - dice_coef_edema: 0.6217 - dice_coef_enhancing: 0.5520
test loss, test acc: [0.022028518840670586, 0.9931290745735168, 0.7159584760665894, 0.6099130511283875, 0.9950689673423767, 0.9916393160820007, 0.99836
3733291626, 0.26758676767349243, 0.6216508746147156, 0.5519809722900391]
```

```
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
# Evaluate the model on the test data using `evaluate`
print("Evaluate on val data")
results = model.evaluate(valid_generator, batch_size=100, callbacks= callbacks)
print("val loss, val acc:", results)
```

```
Evaluate on val data
251/251 [==============================] - 60s 235ms/step - loss: 0.0237 - accuracy: 0.9927 - mean_io_u_3: 0.7092 - dice_coef: 0.6056 - precision: 0.99
47 - sensitivity: 0.9912 - specificity: 0.9982 - dice_coef_necrotic: 0.2720 - dice_coef_edema: 0.6075 - dice_coef_enhancing: 0.5445
val loss, val acc: [0.023715605959296227, 0.9927155375480652, 0.7091755867004395, 0.6055859923362732, 0.9946843385696411, 0.991235613822937, 0.99823623
89564514, 0.2719830572605133, 0.6075195074081421, 0.5445339679718018]
```

Evaluation scores generated during the training:

Epoch (100)	Loss	Accuracy	Validation loss	Validation Accuracy
1	0.0984	0.9814	0.0566	0.9870
10	0.0379	0.9890	0.0365	0.9896
20	0.0313	0.9902	0.0303	0.9906
30	0.0296	0.9905	0.0290	0.9908
40	0.0265	0.9911	0.0290	0.9908
50	0.0245	0.9919	0.0260	0.9918
60	0.0219	0.9926	0.0241	0.9923
70	0.0186	0.9935	0.0235	0.9927
80	0.0173	0.9940	0.0238	0.9928
90	0.0171	0.9941	0.0236	0.9927
100	0.0170	0.9941	0.0237	0.9927

Epoch	MeanIoU_5	Dice_coefficient loss	dice_co-efficient_necrotic	dice_coefficient_edema	dice_coefficient_enhancing	precision	sensitivity	specificity
1	0.5579	0.3029	0.0492	0.1149	0.0653	0.9727	0.9605	0.9956
10	0.5015	0.4296	0.1390	0.3720	0.2105	0.9932	0.9855	0.9978
20	0.5411	0.4721	0.1792	0.4588	0.2527	0.9935	0.9875	0.9979
30	0.5493	0.4847	0.1948	0.4756	0.2706	0.9936	0.9881	0.9979
40	0.5962	0.5079	0.2213	0.5170	0.2955	0.9939	0.9889	0.9980
50	0.5731	0.5406	0.2252	0.5571	0.3820	0.9949	0.9896	0.9983
60	0.6556	0.5829	0.2447	0.5944	0.4939	0.9949	0.9909	0.9983
70	0.7063	0.6247	0.2857	0.6446	0.5698	0.9956	0.9918	0.9985
80	0.7091	0.6416	0.3055	0.6680	0.5941	0.9959	0.9923	0.9987
90	0.7141	0.6449	0.3074	0.6730	0.6002	0.9960	0.9924	0.9987
100	0.7147	0.6455	0.3090	0.6757	0.5985	0.9960	0.9924	0.9987

Overall evaluated scores generated on the **test data.**

Batch size = 1

Epoch	**Loss**	**Accuracy**	**Test loss**	**Test Accuracy**	**MeanIoU**	**Dice_coefficient loss**
100 epoch	0.0220	0.9931	0.0220	0.9931	0.7160	0.6099
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.2676	0.6217	0.5520	0.9951	0.9916	0.9984	

Batch size = 100

Epoch	Loss	Accuracy	Test loss	Test Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0220	0.9931	0.0220	0.9931	0.7160	0.6099
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.2676	0.6217	0.5520	0.9951	0.9916	0.9984	

Overall evaluated scores generated on the **Validation data.**

Batch size = 1

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0237	0.9927	0.0237	0.9927	0.7092	0.6056
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.2720	0.6075	0.5445	0.9947	0.9912	0.9982	

Batch size = 100

Epoch	Loss	Accuracy	Validation loss	Validation Accuracy	MeanIoU	Dice_coefficient loss
100 epoch	0.0237	0.9927	0.0237	0.9927	0.7092	0.6056
Dice_co-efficient_necrotic	**Dice_coefficient_edema**	**Dice_coefficient_enhancing**	**Precision**	**Sensitivity**	**Specificity**	
0.2720	0.6075	0.5445	0.9947	0.9912	0.9982	

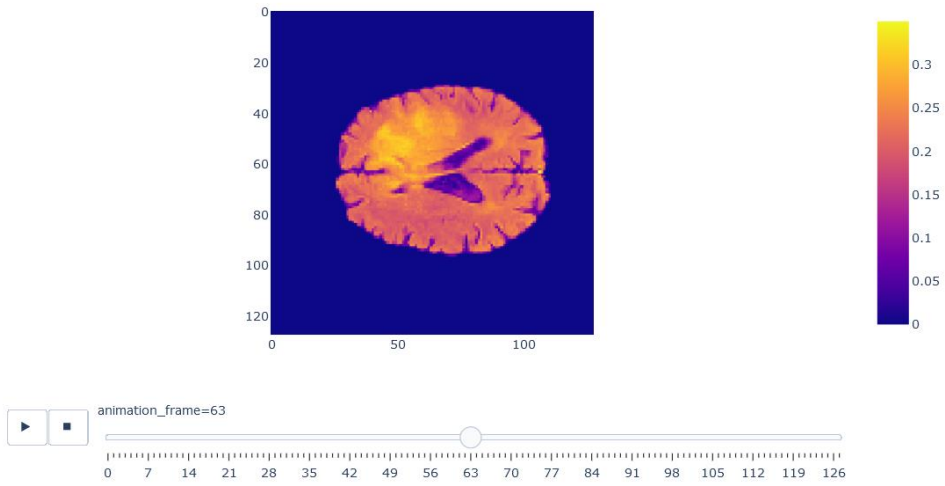
Prediction of BraTS 2021 3D Dataset based on the tumor sub-regions for different slices.

Tumor Sub regions –

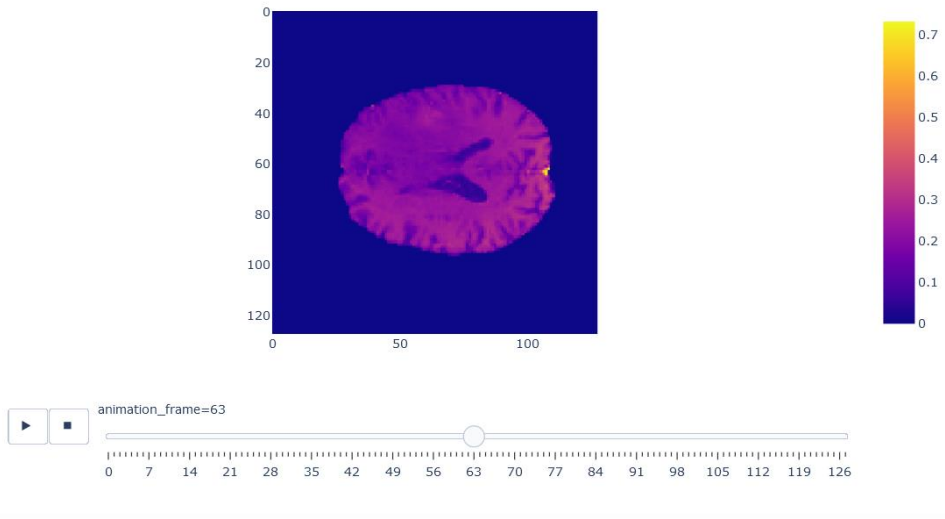
0- Normal - means no tumor
1- Necrotic/core region.
2- Edema region.
3- Enhancing tumor region.

1) Predicting the prediction for images in the test_generator based on the tumor sub-regions.

```
In [15]: import plotly.express as px
         import matplotlib.pyplot as plt

         fig = px.imshow(test_generator[0][0][0][:, :, :, 0], animation_frame=0, binary_string=False)
         fig.show()
```




```
In [16]: fig = px.imshow(test_generator[0][0][0][:, :, :, 1], animation_frame=0, binary_string=False)
         fig.show()
```

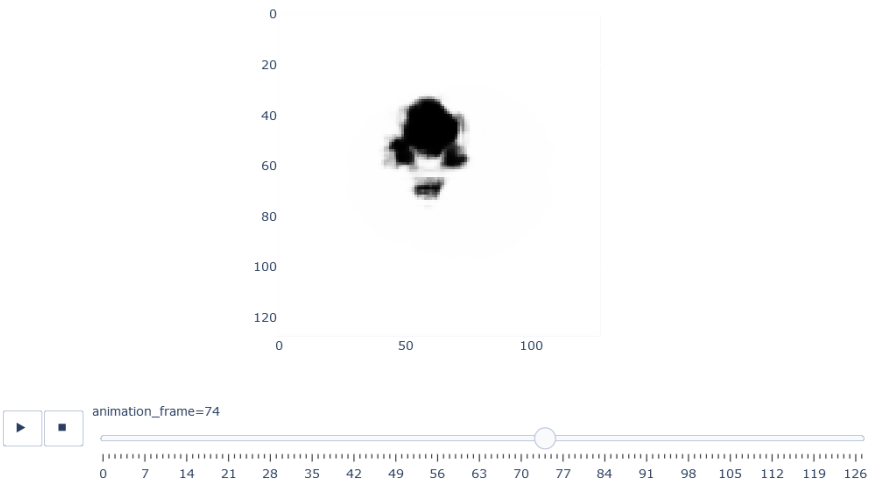


Detection of tumor sub-regions of an images in the test_generator.

```
In [17]: i = test_generator[0]

In [18]: i[0].shape
Out[18]: (1, 128, 128, 128, 2)

In [19]: z = model(i[0])

In [20]: z.shape
Out[20]: TensorShape([1, 128, 128, 128, 4])
```

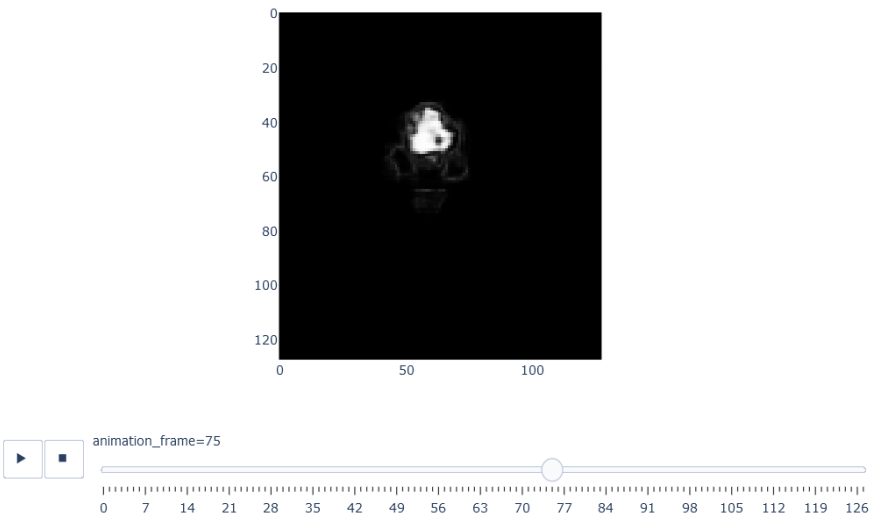
0- Normal - means no tumor

```
In [21]: fig = px.imshow(z[0][:, :, :, 0], animation_frame=0, binary_string=True)
         fig.show()
```



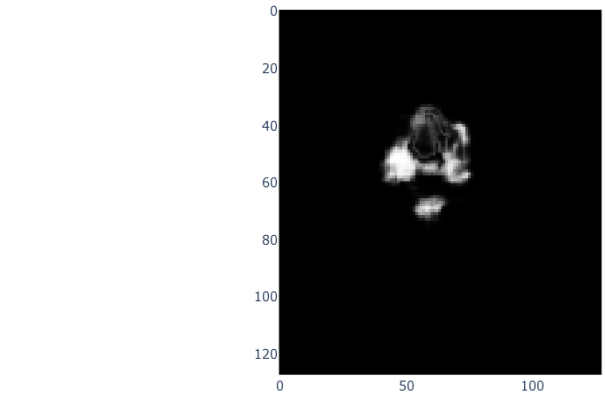
1- Necrotic/core region of an image.

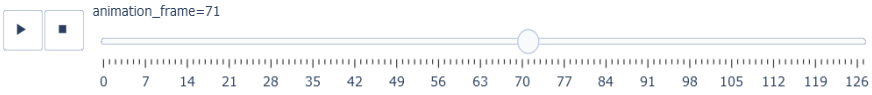
```
In [22]: fig = px.imshow(z[0][:, :, :, 1], animation_frame=0, binary_string=True)
         fig.show()
```



2- Edema region of an image.

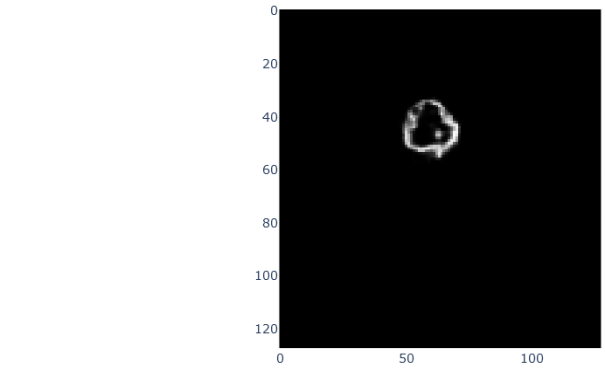
```
In [23]: fig = px.imshow(z[0][:, :, :, 2], animation_frame=0, binary_string=True)
         fig.show()
```

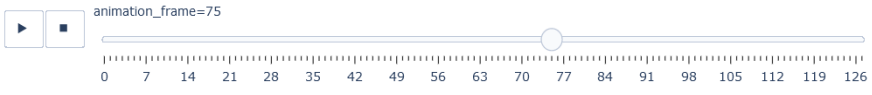




3- Enhancing tumor region of an image.

```
In [24]: fig = px.imshow(z[0][:, :, :, 3], animation_frame=0, binary_string=True)
         fig.show()
```

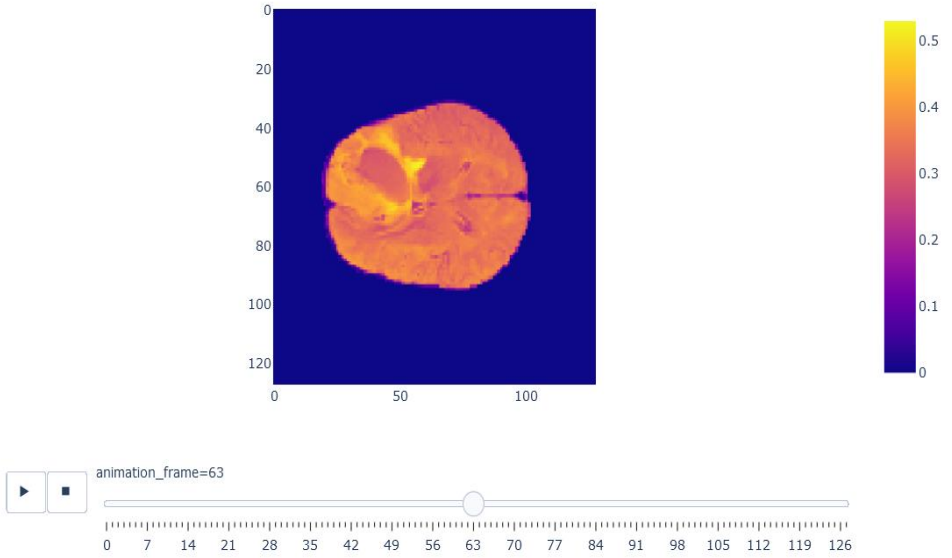




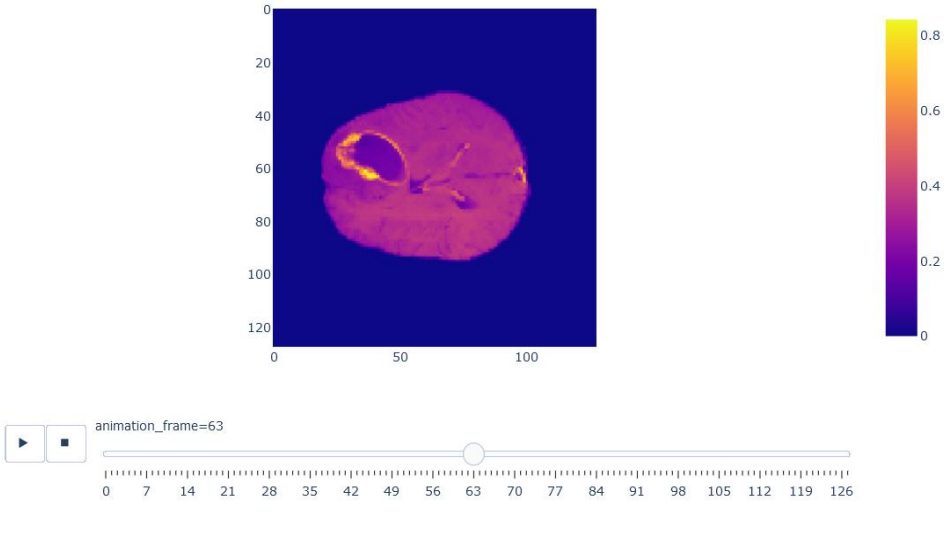
2) Predicting the prediction for images in the valid_generator based on the tumor sub-regions.

Valid generator

In [25]:
```
fig = px.imshow(valid_generator[0][0][0][:, :, :, 0], animation_frame=0, binary_string=False)
fig.show()
```



In [26]:
```
fig = px.imshow(valid_generator[0][0][0][:, :, :, 1], animation_frame=0, binary_string=False)
fig.show()
```

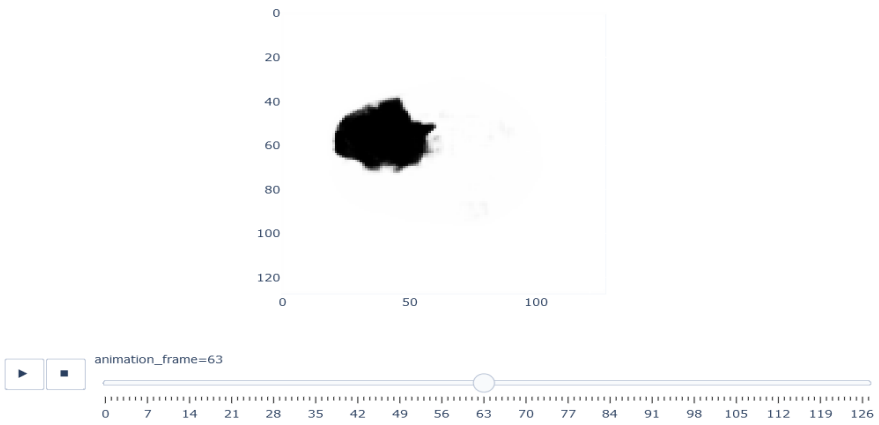


Detection of tumor sub-regions of an images in the validation generator.

```
In [33]: i1 = valid_generator[0]

In [34]: i1[0].shape
Out[34]: (1, 128, 128, 128, 2)

In [35]: z1 = model(i1[0])

In [36]: z1.shape
Out[36]: TensorShape([1, 128, 128, 128, 4])

In [37]: fig = px.imshow(z1[0][:, :, :, 0], animation_frame=0, binary_string=True)
         fig.show()
```

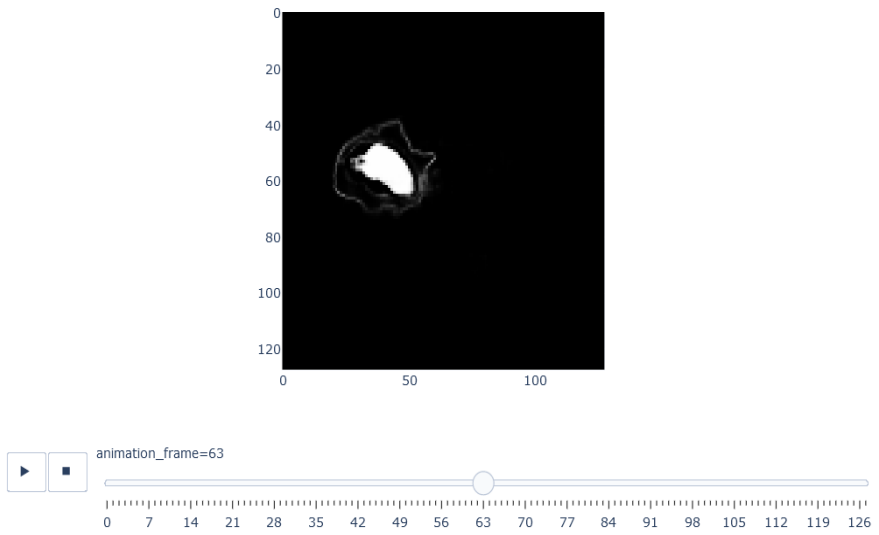
0- Normal - means no tumor

```
In [37]: fig = px.imshow(z1[0][:, :, :, 0], animation_frame=0, binary_string=True)
         fig.show()
```



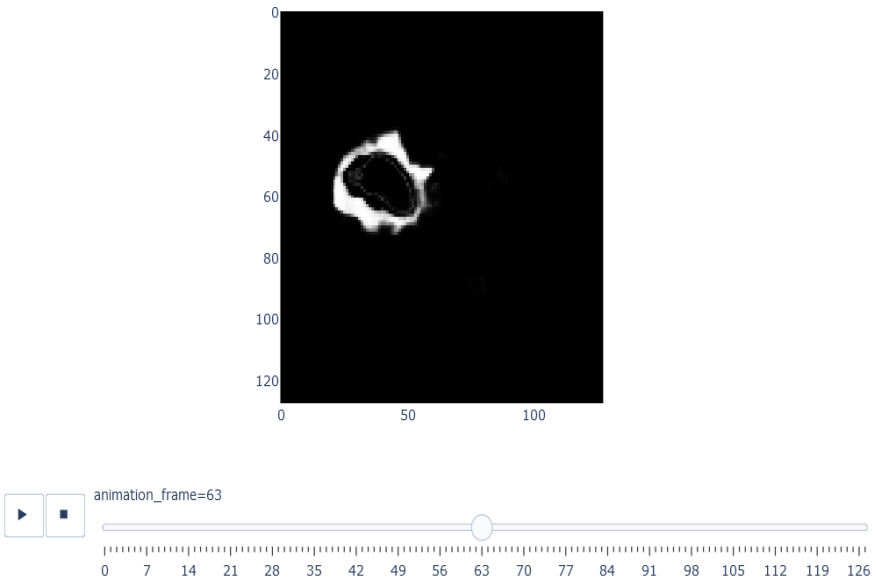
1- Necrotic/core region of an image.

```
In [38]: fig = px.imshow(z1[0][:, :, :, 1], animation_frame=0, binary_string=True)
         fig.show()
```



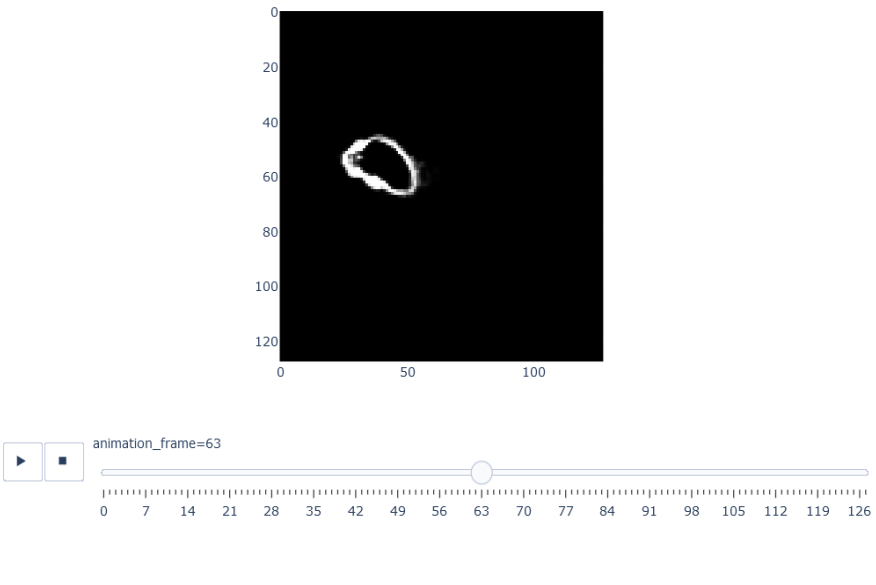
2- Edema region of an image.

```
In [39]: fig = px.imshow(z1[0][:, :, :, 2], animation_frame=0, binary_string=True)
         fig.show()
```



3- Enhancing tumor region of an image.

```
In [40]: fig = px.imshow(z1[0][:, :, :, 3], animation_frame=0, binary_string=True)
         fig.show()
```



6.2 Analysis of results.

EVALUATION SCORES GENERATED ON THE TESTING SET/ DATA FOR 100 EPOCHS WITH BATCH SIZE OF 1										
								DICE CO-EFFICIENT OF		
MODIFIED 3D U-Net MODELS	LOSS	ACCURACY	PRECISION	SENSITIVITY	SPECIFICITY	MEANIOU SCORE	DICE SIMILARITY OF CO-EFFICIENT	NECROTIC REGION	EDEMA REGION	ENHANCING TUMOR REGION
4 number of filter with 3*3*3 convolution	0.0158	0.9952	0.9963	0.9944	0.9988	0.8357	0.7175	0.3893	0.7502	0.7316
8 number of filter with 3*3*3 convolution	0.016	0.9948	0.996	0.994	0.9987	0.8694	0.7315	0.4813	0.7333	0.7124
16 number of filter with 3*3*3 convolution	0.0138	0.9955	0.9965	0.9947	0.9989	0.8558	0.7611	0.5071	0.7794	0.7592
4 number of filter with 2*2*2 convolution	0.022	0.9931	0.9951	0.9916	0.9984	0.716	0.6099	0.2676	0.6217	0.552
8 number of filter with 2*2*2 convolution	0.0201	0.9943	0.9954	0.9934	0.9985	0.7647	0.6831	0.4123	0.6551	0.6665
16 number of filter with 2*2*2 convolution	0.0182	0.9946	0.9957	0.9939	0.9986	0.7828	0.7134	0.4518	0.695	0.708

EVALUATION SCORES GENERATED ON THE VALIDATION SET/ DATA FOR 100 EPOCHS WITH BATCH SIZE OF 1										
								DICE CO-EFFICIENT OF		
MODIFIED 3D U-Net MODELS	LOSS	ACCURACY	PRECISION	SENSITIVITY	SPECIFICITY	MEANIOU SCORE	DICE SIMILARITY OF CO-EFFICIENT	NECROTIC REGION	EDEMA REGION	ENHANCING TUMOR REGION
4 number of filter with 3*3*3 convolution	0.0139	0.9955	0.9966	0.9947	0.9989	0.8344	0.7468	0.472	0.7623	0.753
8 number of filter with 3*3*3 convolution	0.0142	0.9954	0.9965	0.9946	0.9888	0.8713	0.7569	0.5035	0.7607	0.7645
16 number of filter with 3*3*3 convolution	0.0156	0.995	0.9961	0.9941	0.9987	0.8539	0.7626	0.5124	0.7422	0.7971
4 number of filter with 2*2*2 convolution	0.0237	0.9927	0.9947	0.9912	0.9982	0.7092	0.6056	0.272	0.6075	0.5445
8 number of filter with 2*2*2 convolution	0.0228	0.9939	0.9951	0.993	0.9984	0.7667	0.6893	0.4075	0.6575	0.694
16 number of filter with 2*2*2 convolution	0.0209	0.9943	0.9954	0.9935	0.9985	0.7837	0.7105	0.4365	0.6851	0.7219

Figure 6.9: Evaluation scores generated for the Testing set and the Validation set of Brain Tumor segmentation of the BraTS 2021 dataset.

The research project is demonstrated with the 3D U-Net architecture for segmentation of brain tumor segmentation on the BraTS 2021 dataset containing the large-scale data of having quality pixel-level annotations of 1251 data in the training dataset, each data is provided with five different annotation images (Chen et al., 2018). The overall evaluated methods results are tabulated for all the modified 3D U-Net architectures for this refer to section (6.1 results of experiments). In our study, we compared all the seven modified 3D U-Net architectures scores and also compared our work with the other segmentation models performed on the BraTS dataset.

The above figures in the Analysis of results show the evidence of our evaluation scores generated for the testing set and the validation set for which several modifications are done for the 3D U-Net model design work by tuning the hyper-parameters of the model like changing the convolution size of the layers, increasing the number of filters, doing 2*2*2 max pooling operations after the convolution layers, adding the strides in the convolution 3D transpose layer in the decoder path. Thus, tuning of hyper-parameters additionally helped the model to improve the performance of the dice scores of the tumor sub-regions.

On comparing all the seven modified 3D U-Net architectures results with the test generator or test sets, the 3D U-Net modified model started with 16 number of filters with a kernel size of 3*3*3 convolution produced less loss i.e., 0.0138 for the model, also produced high accuracy of 0.9955 with the precision of 0.9965, the sensitivity of 0.9947, and specificity of 0.9989 scores. The dice similarity of co-efficient

is 0.7611, the segmentation of tumor sub-region like dice co-efficient of Necrotic, dice co-efficient of Edema, and dice co-efficient of Enhancing tumor produced the highest scores of 0.5071, 0.7794, and 0.7592, among all the seven modified 3D U-Net architectures. But, the 3D U-Net modified model started with 8 number of filters with a kernel size of 3*3*3 convolution produced the highest MeanIoU score of 0.8694.

On comparing all the seven modified 3D U-Net architectures results with the valid generator or validation sets, the 3D U-Net modified model started with 4 number of filters with a kernel size of 3*3*3 convolution produced less loss i.e., 0.0139, also produced the highest accuracy of 0.9955 with the precision of 0.9966, the sensitivity of 0.9947, and specificity of 0.9989 scores with high dice co-efficient of Edema score of 0.7623. And the 3D U-Net modified model started with 16 number of filters with a kernel size of 3*3*3 convolution produced the highest dice similarity of co-efficient score of 0.7626 and for tumor sub-regions, this model produced the highest score for dice co-efficient of Necrotic and dice co-efficient of Enhancing tumor is 0.5124 and 0.7971. But, the 3D U-Net modified model started with 8 number of filters with a kernel size of 3*3*3 convolution produced the highest MeanIoU score of 0.8713.

Overall, considering all the models, the 3D U-Net modified model started with 16 number of filters with a kernel size of 3*3*3 convolution performed well on the testing set by producing good segmentation scores for all the evaluation methods. And when talking about the validation set, the 3D U-Net modified model started with 4 number of filters with a kernel size of 3*3*3 convolution performed well on producing the lowest loss, and yielded high scores on Accuracy, Precision, Sensitivity, and Specificity in evaluation methods. And on comparing the modified 3D U-Net model's performance and generated results are compared with the other segmentation algorithms performed in the (Dong et al., 2017) paper for the various BraTS datasets like 2013, 2015, and used BraTS 2019 dataset in (Jiang, Ding, Liu and Tao, 2019) paper, The 3D U-Net modified model started with 16 number of filters with a kernel size of 3*3*3 convolution achieved the good score of dice co-efficient for enhancing tumor is 0.7971 score for the BraTS 2021 dataset in the validation set and also achieved highest accuracy and MeanIoU score of 0.8713 for (8 number of filters with 3*3*3 convolutions) for this model.

7. Discussions and Reflective Analysis

In this research work, the project proposed the 3D U-Net model based on the original U-Net model, and remodified the 3D U-Net model architectures for training and evaluating the results. The proposed method helps in segmenting the tumors present in the tumor sub-regions like necrotic core, Edema, and Enhancing tumor core by generating the dice scores. And comparing the results among the modified architectures and the final comparison is done with the state-of-art-methods used in the previous datasets of Brain tumor segmentation. Our modified method yields good accuracy when it is compared with the other state-of-art methods. And one of the modified models started with 16 number of filters with a kernel size of 3*3*3 convolution performed well on the testing set by producing the good segmentation scores for all the tumor sub-regions. Nevertheless, one of our modified models achieved the highest MeanIoU score of 0.8713. Our results have been compared with the (Dong et al., 2017) paper and (Jiang, Ding, Liu and Tao, 2019) paper. And our method achieved superior results in the enhancing tumor core region and also yields good precision, Specificity, and Sensitivity. The proposed modified 3D U-net models performed better in the validation sets when it is compared with the testing set of the BraTS 2021 dataset. As the callbacks are used for training the model to prevent overfitting of the model. The 3D U-Net model architecture with 32 number of filters with a kernel size of 3*3*3 convolution has stopped at 12 epochs when the whole model is set to 100 epochs for training using the callbacks by setting the patience to 8. Further, this model could have been trained again for improving the evaluation scores.

The project uses the Lab systems provided by the University of Lincoln for cost-efficient and has 16 GB of memory with Nvidia 3060 GPUs graphics card provided for the systems. For training of each modified 3D U-Net model architecture, it took around 8 to 12 hours. For further improving the model architectures, when the number of filters is increased and fed for training the model, At the initial epoch, the model is used to run and in between the steps in the first epoch, it stops automatically and throws the root node error like graph execution, if we are reshaping the image sizes for further training the model it throws the memory error. Sometimes when the project increases the number of filters in the convolution layer for further research and fed to the training, during the training, the system gets crashed because it cannot withstand the maximum load. As the dataset size is huge and for withstanding the load the system needs high computing power for training the model and high GPUs are required to complete the training at a faster time.

8. Conclusion

From the research point of view, the segmentation of MRI images of brain tumors done manually is more challenging, difficult, and takes more time. To overcome the manual segmentation, further research has been carried out to save time for doing the Automatic segmentation of MRI images of brain tumors. The challenging part in doing the automatic segmentation is finding the exact size of the tumor and the location of gliomas present in the brain tumor. The various state-of-art methods are employed for the segmentation of MRI images in the brain tumor segmentation of the BraTS 2012 to 2021 dataset of the BraTS challenge. This project implements the 3D U-Net model and the modified 3D U-Net model architectures for the Brain tumor segmentation of the BraTS 2021 dataset. The evaluation methods chosen for Brain tumor Segmentations are Loss, Accuracy, MeanIoU, Precision, Dice Similarity of Co-efficient, Specificity, Sensitivity, The tumor sub-regions like Dice Co-efficient of Necrotic core region, Dice Co-efficient of Edema region, and Dice Co-efficient of Enhancing tumor region. The testing set and the validation set results show that the modified 3D U-Net models architecture can achieve excellent performance and each model trained are end-to-end fashioned.

The models are modified by tuning the hyperparameters like adding the 3*3*3 convolution or 2*2*2 convolution layers with a different number of filters and using the activation ReLU function in each convolution layer of the encoder and decoder path. The final layer consists of 1*1*1 convolution with a softmax activation function for improving the analyses of the evaluation scores and when we see the evaluation scores generated for the testing set and validation set in figures (6.2 section for analysis of results) the modified models yielded good evaluation scores for accuracy, MeanIoU score. The tuning of hyperparameters of the model architectures helped in producing good dice similarity scores of the tumor sub-regions for the Necrotic core region, Edema core region, and the enhancing tumor core region.

The results generated from the testing set and the validation set which are trained by different modified 3D U-Net model architectures are compared among themselves and further compared with the various state-of-art methods. The proposed 3D U-Net models generate good dice scores for all the tumor sub-regions, as one of the modified 3D U-Net models, achieved good dice co-efficient for enhancing the tumor score of 0.7971. The modified 3D U-net models achieved the highest accuracy, precision, sensitivity, Specificity and one of the modified 3D U-Net models achieved the highest MeanIoU score of 0.8713 in the validation set and 0.8694 in the Testing set.

9. References and Appendices

9.1 References

Baid, U., Ghodasara, S., Mohan, S., Bilello, M., Calabrese, E., Colak, E., Farahani, K., Kalpathy-Cramer, J., Kitamura, F.C., Pati, S., Prevedello, L.M., Rudie, J.D., Sako, C., Shinohara, R.T., Bergquist, T., Chai, R., Eddy, J., Elliott, J., Reade, W. and Schaffter, T. (2021). The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification. *arXiv:2107.02314 [cs]*. [online] Available at: https://arxiv.org/abs/2107.02314.

Bakas, S., Reyes, M., Jakab, A., Bauer, S., Rempfler, M., Crimi, A., Shinohara, R.T., Berger, C., Ha, S.M., Rozycki, M., Prastawa, M., Alberts, E., Lipkova, J., Freymann, J., Kirby, J., Bilello, M., Fathallah-Shaykh, H., Wiest, R., Kirschke, J. and Wiestler, B. (2019). Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge. *arXiv:1811.02629 [cs, stat]*. [online] Available at: https://arxiv.org/abs/1811.02629.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F. and Adam, H. (2018). *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. [online] openaccess.thecvf.com. Available at: http://openaccess.thecvf.com/content_ECCV_2018/html/Liang-Chieh_Chen_Encoder-Decoder_with_Atrous_ECCV_2018_paper.html [Accessed 21 Sep. 2022].

Dong, H., Yang, G., Liu, F., Mo, Y. and Guo, Y. (2017). Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks. *Communications in Computer and Information Science*, [online] pp.506–517. doi:10.1007/978-3-319-60964-5_44.

Işın, A., Direkoğlu, C. and Şah, M. (2016). Review of MRI-based Brain Tumor Image Segmentation Using Deep Learning Methods. *Procedia Computer Science*, [online] 102, pp.317–324. doi:10.1016/j.procs.2016.09.407.

Jiang, Z., Ding, C., Liu, M. and Tao, D. (2020). Two-Stage Cascaded U-Net: 1st Place Solution to BraTS Challenge 2019 Segmentation Task. *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pp.231–241. doi:10.1007/978-3-030-46640-4_22.

Lloyd, T. and Macrae, C. (2020). *Masaryk University Faculty of Economics and Administration Field of Study: Business Management IMPLEMENTING THE PRINCE2 PROJECT MANAGEMENT METHOD IN AN EXISTING PROJECT ENVIRONMENT*. [online] Available at: https://is.muni.cz/th/u8ju8/Thesis_Demianchuk.pdf.

Menze, B.H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., Lanczi, L., Gerstner, E., Weber, M.-A., Arbel, T., Avants, B.B., Ayache, N.,

Buendia, P., Collins, D.L., Cordier, N. and Corso, J.J. (2015). The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10), pp.1993–2024. doi:10.1109/tmi.2014.2377694.

Pereira, S., Pinto, A., Alves, V. and Silva, C.A. (2016). Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images. *IEEE Transactions on Medical Imaging*, 35(5), pp.1240–1251. doi:10.1109/tmi.2016.2538465.

Ronneberger, O., Fischer, P. and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science*, pp.234–241. doi:10.1007/978-3-319-24574-4_28.

Schwaber, K. (1997). SCRUM Development Process. *Business Object Design and Implementation*, pp.117–134. doi:10.1007/978-1-4471-0947-1_11.

Sundaresan, V., Griffanti, L. and Jenkinson, M. (2021). Brain Tumour Segmentation Using a Triplanar Ensemble of U-Nets on MR Images. *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pp.340–353. doi:10.1007/978-3-030-72084-1_31.

Zhao, H., Shi, J., Qi, X., Wang, X. and Jia, J. (2017). *Pyramid Scene Parsing Network*. [online] openaccess.thecvf.com. Available at: http://openaccess.thecvf.com/content_cvpr_2017/html/Zhao_Pyramid_Scene_Parsing_CVPR_2017_paper.html [Accessed 21 Sep. 2022].

9.2 Appendices

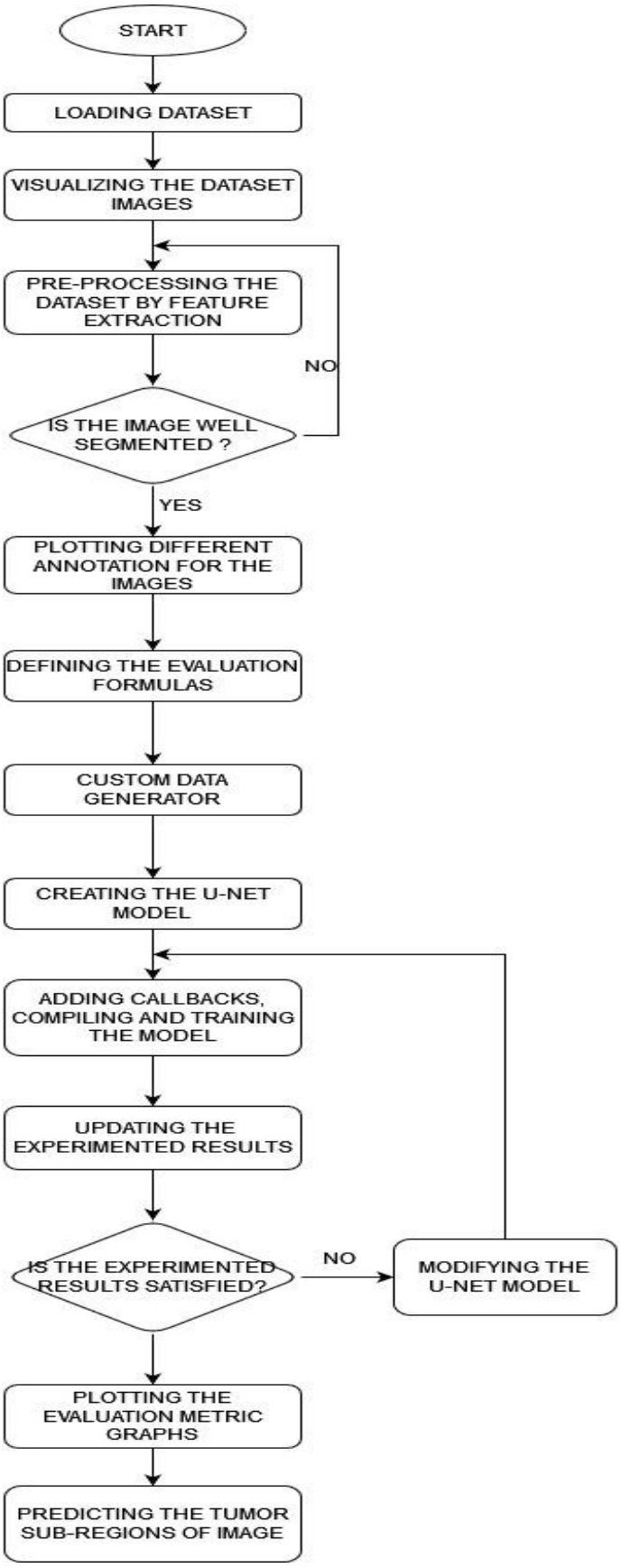


Figure 9.1: Flowchart for the source-code.

Code for Brain Tumor Segmentation of BraTS 2021 Dataset.

The given code is implemented with the help of the following links:
https://www.kaggle.com/code/shreyaspj/3d-mri-analysis and
https://www.kaggle.com/code/maksudaislamlima/3d-u-net-brats20
https://www.kaggle.com/code/maksudaislamlima/3d-unet-brats2020

The prediction for all the Brain tumor sub-regions of the brain tumor segmentation of the BraTS 2021 dataset code has been implemented from the following link:

https://www.kaggle.com/code/watermasterz/mri-brats-3d-conv

The 3D U-Net model architecture has been modified with the help of the given link:
https://github.com/bnsreenu/python_for_microscopists/blob/master/231_234_BraTa2020_Unet_segmentation/simple_3d_unet.py

The modifications done for 3D U-Net model architectures code is available in my GitHub account for Brain tumor Segmentation of BraTS 2021 Dataset using deep learning method:

https://github.com/saduabhi/MSc-Project--Brain-tumor-segmentation-BraTS-2021-

The Overview structure of the modifications done for the 3D U-Net model and trained for evaluating the evaluation method results.

Step 1: Installing the Dependencies.

Installing the dependencies

```
import numpy as np
import nibabel as nib
import itk
import itkwidgets
from ipywidgets import interact, interactive, IntSlider, ToggleButtons
import matplotlib.pyplot as plt
from skimage.util import montage
from skimage.transform import rotate

%matplotlib inline
import seaborn as sns
```

```
import keras
import keras.backend as K
from keras.callbacks import CSVLogger
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, TensorBoard
from tensorflow.keras.layers.experimental import preprocessing
import cv2
# Make numpy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
```

```
import nilearn as nl
import nibabel as nib
import nilearn.plotting as nlplt
```

Step 2: Loading the dataset.

```
image_path = r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/BraTS2021_00005/BraTS2021_00005_flair.nii.gz"
image_obj = nib.load(image_path)
type(image_obj)
```

```
nibabel.nifti1.Nifti1Image
```

```
image_data = image_obj.get_fdata()
print(type(image_data))
image_data.shape
```

```
<class 'numpy.ndarray'>
(240, 240, 155)
```

```
TRAIN_DATASET_PATH = r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/"
```

```
import os
nSample = os.listdir(TRAIN_DATASET_PATH + 'BraTS2021_00005')
nSample
```

```
['BraTS2021_00005_flair.nii.gz',
 'BraTS2021_00005_seg.nii.gz',
 'BraTS2021_00005_t1.nii.gz',
 'BraTS2021_00005_t1ce.nii.gz',
 'BraTS2021_00005_t2.nii.gz']
```

```
test_image_flair=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_flair.nii.gz').get_fdata()
test_image_t1=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t1.nii.gz').get_fdata()
test_image_t1ce=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t1ce.nii.gz').get_fdata()
test_image_t2=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_t2.nii.gz').get_fdata()
test_mask=nib.load(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_seg.nii.gz').get_fdata()
```

Step 3: visualizing the images in the dataset.

```
def visualize_3d(layer):
    plt.figure(figsize=(10, 5))
    plt.imshow(image_data[:, :, layer], cmap='gray');
    plt.axis('off')
    return layer

interact(visualize_3d, layer=(0, image_data.shape[2] - 1));
```

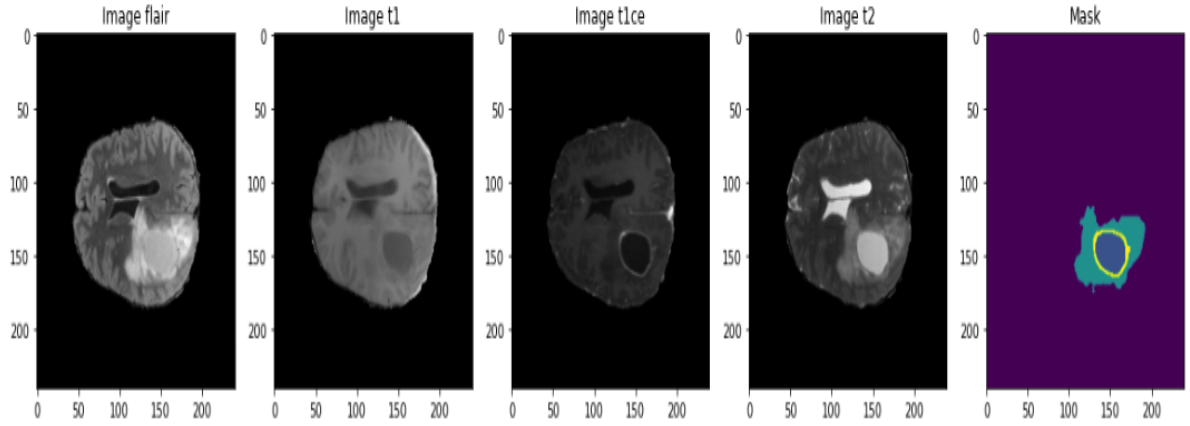
```
interactive(children=(IntSlider(value=77, description='layer', max=154), Output()), _dom_classes=('widget-inte…
```

```
classes_dict = {
    'Normal': 0.,
    'Edema': 1.,
    'Non-enhancing tumor': 2.,
    'Enhancing tumor': 3.
}

def visualize_3d_labels(layer):
    mask = nib.load(r"C:/Users/Computing/Desktop/archive/BraTS2021_Training_Data/BraTS2021_00005/BraTS2021_00005_seg.nii.gz").get_fdata()
    plt.imshow(mask[:,:,layer])
    plt.axis('off')
    plt.tight_layout()

interact(visualize_3d_labels, layer=(0, image_data.shape[2] - 1));
```

```
interactive(children=(IntSlider(value=77, description='layer', max=154), Output()), _dom_classes=('widget-inte…
```

```
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1,5, figsize = (20, 10))
slice_w = 25
ax1.imshow(test_image_flair[:,:,test_image_flair.shape[0]//2-slice_w], cmap = 'gray')
ax1.set_title('Image flair')
ax2.imshow(test_image_t1[:,:,test_image_t1.shape[0]//2-slice_w], cmap = 'gray')
ax2.set_title('Image t1')
ax3.imshow(test_image_t1ce[:,:,test_image_t1ce.shape[0]//2-slice_w], cmap = 'gray')
ax3.set_title('Image t1ce')
ax4.imshow(test_image_t2[:,:,test_image_t2.shape[0]//2-slice_w], cmap = 'gray')
ax4.set_title('Image t2')
ax5.imshow(test_mask[:,:,test_mask.shape[0]//2-slice_w])
ax5.set_title('Mask')
```

```
Text(0.5, 1.0, 'Mask')
```



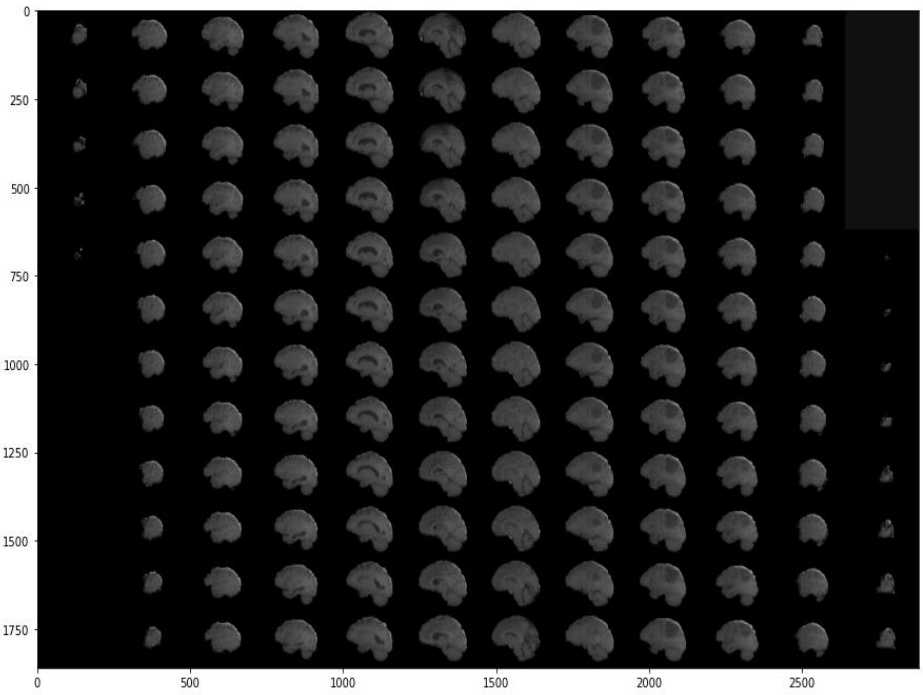
Step 4: Segmentation of images in the dataset.

```
# DEFINE seg-areas
SEGMENT_CLASSES = {
    0 : 'NOT tumor',
    1 : 'NECROTIC/CORE', # or NON-ENHANCING tumor CORE - RED
    2 : 'EDEMA',  # Green
    3 : 'ENHANCING' # original 4 -> converted into 3 later, Yellow
}

# there are 155 slices per volume
# to start at 5 and use 145 slices means we will skip the first 5 and last 5
VOLUME_SLICES = 128
VOLUME_START_AT = 22 # first slice of volume that we will include

IMG_SIZE=128
```

Step 5: Plotting the annotation images and masking them.

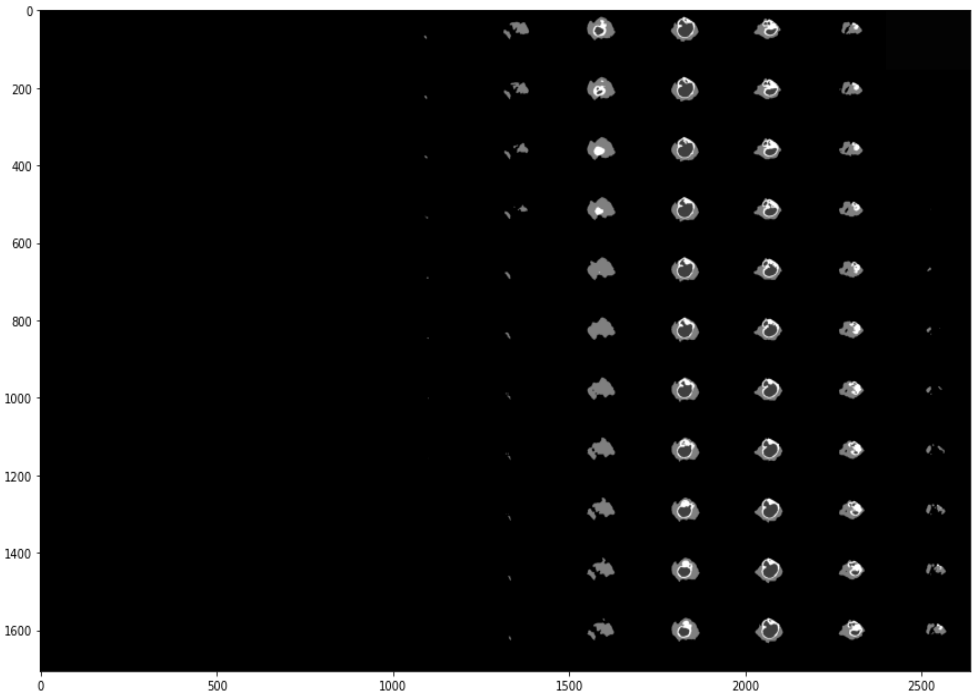
```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_image_t1[50:-50,:,:]), 90, resize=True), cmap ='gray')
```

```
<matplotlib.image.AxesImage at 0x19281621370>
```

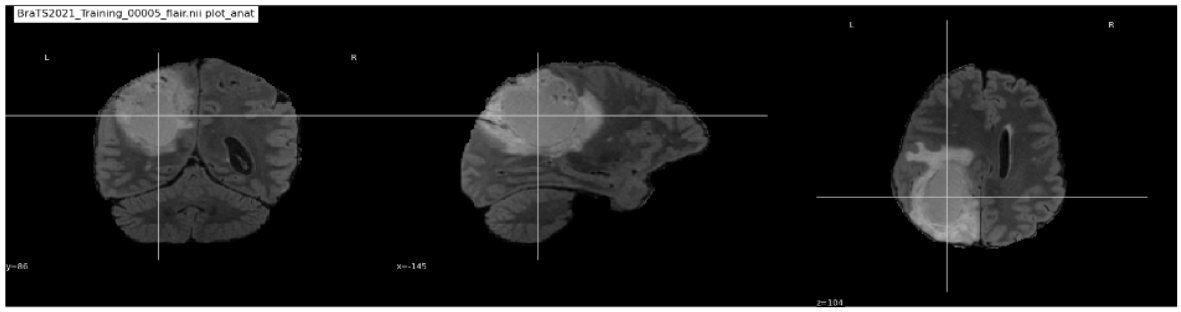


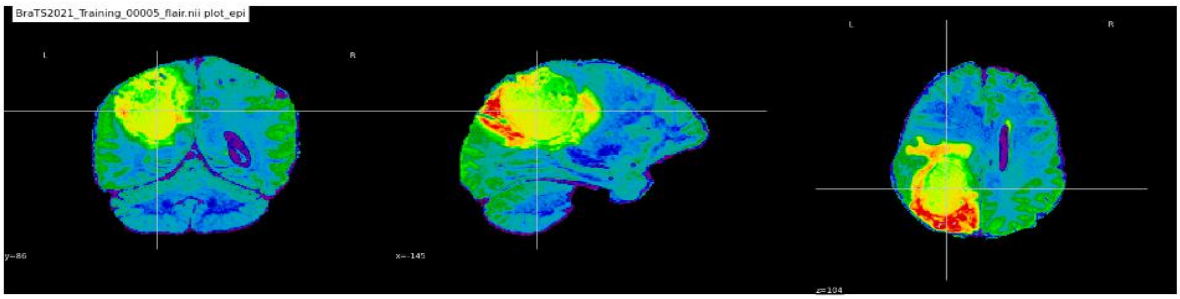

```
fig, ax1 = plt.subplots(1, 1, figsize = (15,15))
ax1.imshow(rotate(montage(test_mask[60:-60,:,:]), 90, resize=True), cmap ='gray')
```

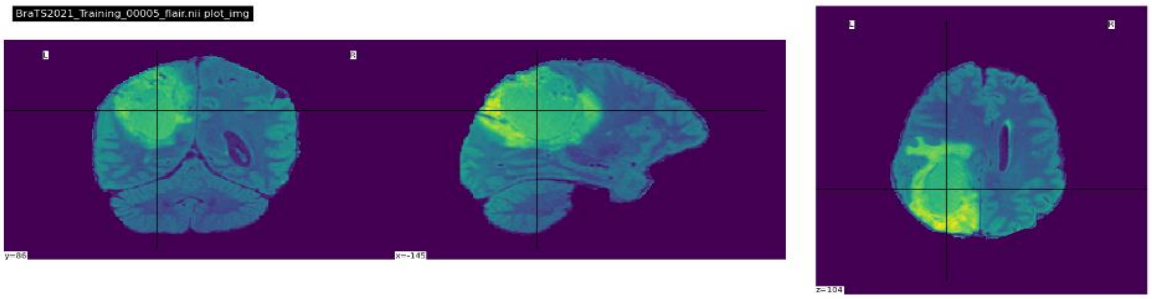
```
<matplotlib.image.AxesImage at 0x192802b7fd0>
```

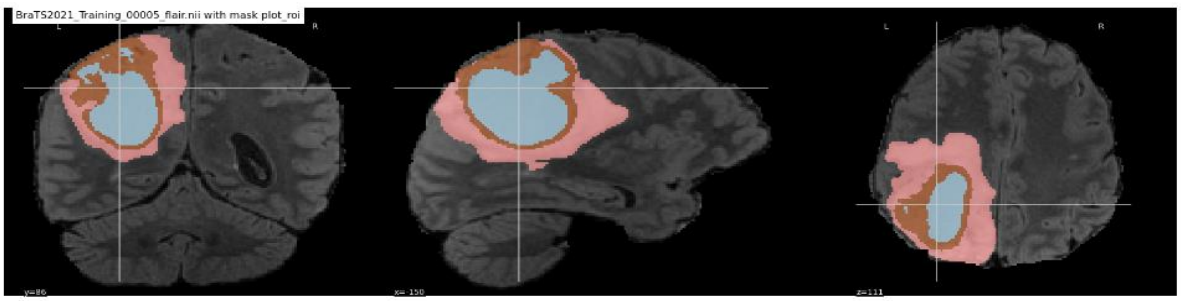


```
niimg = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_flair.nii.gz')
nimask = nl.image.load_img(TRAIN_DATASET_PATH + 'BraTS2021_00005/BraTS2021_00005_seg.nii.gz')

fig, axes = plt.subplots(nrows=4, figsize=(30, 40))


nlplt.plot_anat(niimg,
                title='BraTS2021_Training_00005_flair.nii plot_anat',
                axes=axes[0])

nlplt.plot_epi(niimg,
               title='BraTS2021_Training_00005_flair.nii plot_epi',
               axes=axes[1])

nlplt.plot_img(niimg,
               title='BraTS2021_Training_00005_flair.nii plot_img',
               axes=axes[2])

nlplt.plot_roi(nimask,
               title='BraTS2021_Training_00005_flair.nii with mask plot_roi',
               bg_img=niimg,
               axes=axes[3], cmap='Paired')

plt.show()
```









Step 6: Defining the Evaluation method formulas.

```
# dice loss as defined above for 4 classes
def dice_coef(y_true, y_pred, epsilon=0.00001):
    """
    Dice = (2*|X & Y|)/ (|X|+ |Y|)
         =  2*sum(|A*B|)/(sum(A^2)+sum(B^2))
    ref: https://arxiv.org/pdf/1606.04797v1.pdf

    """
    axis = (0,1,2,3)
    dice_numerator = 2. * K.sum(y_true * y_pred, axis=axis) + epsilon
    dice_denominator = K.sum(y_true*y_true, axis=axis) + K.sum(y_pred*y_pred, axis=axis) + epsilon
    return K.mean((dice_numerator)/(dice_denominator))


# define per class evaluation of dice coef
# inspired by https://github.com/keras-team/keras/issues/9395
def dice_coef_necrotic(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,1] * y_pred[0,:,:,:,1]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,1])) + K.sum(K.square(y_pred[0,:,:,:,1])) + epsilon)

def dice_coef_edema(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,2] * y_pred[0,:,:,:,2]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,2])) + K.sum(K.square(y_pred[0,:,:,:,2])) + epsilon)

def dice_coef_enhancing(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[0,:,:,:,3] * y_pred[0,:,:,:,3]))
    return (2. * intersection) / (K.sum(K.square(y_true[0,:,:,:,3])) + K.sum(K.square(y_pred[0,:,:,:,3])) + epsilon)

def Hausdorff_distance(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)

    print(type(y_pred_f))
    # first method
    d2_matrix = pairwise_distances(y_true_f,y_pred_f, metric ='euclidean')
    result = tf.math.reduce_mean(tf.math.minimum(d2_matrix, axis=0)) + tf.math.reduce_mean(tf.math.minimum(d2_matrix, axis=1))
    return result

# Computing Precision
def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision


# Computing Sensitivity
def sensitivity(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    return true_positives / (possible_positives + K.epsilon())


# Computing Specificity
def specificity(y_true, y_pred):
    true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
    possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
    return true_negatives / (possible_negatives + K.epsilon())
```

Step 7: Splitting the data.

```
import os
train_and_val_directories = [f.path for f in os.scandir(TRAIN_DATASET_PATH) if f.is_dir()]
```

```
def pathListIntoIds(dirList):
    x = []
    for i in range(0,len(dirList)):
        x.append(dirList[i][dirList[i].rfind('/')+1:])
    return x

train_and_test_ids = pathListIntoIds(train_and_val_directories);


train_test_ids, val_ids = train_test_split(train_and_test_ids,test_size=0.2)
train_ids, test_ids = train_test_split(train_test_ids,test_size=0.15)
```

```
train_and_test_ids[0]
```

```
'BraTS2021_00000'
```

```
train_ids[0]
```

```
'BraTS2021_01344'
```

Step 8: Data Generating.

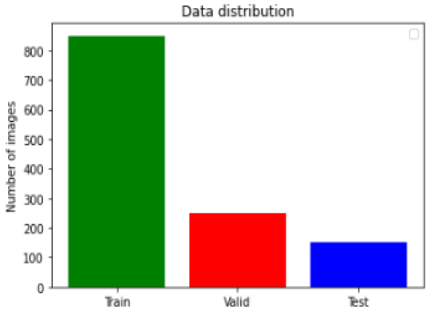
```
keras = tf.compat.v1.keras
Sequence = keras.utils.Sequence

class DataGenerator(Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, dim=(IMG_SIZE,IMG_SIZE), batch_size = 1, n_channels = 2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
        # Find list of IDs
        Batch_ids = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(Batch_ids)

        return X, y

    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)

    def __data_generation(self, Batch_ids):
        'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
        # Initialization
        X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channels))
        y = np.zeros((self.batch_size*VOLUME_SLICES, 128, 128))
        Y = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, 4))


        # Generate data
        for c, i in enumerate(Batch_ids):
            case_path = os.path.join(TRAIN_DATASET_PATH, i)

            data_path = os.path.join(case_path, f'{i}_flair.nii.gz');
            flair = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_t1ce.nii.gz');
            ce = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_seg.nii.gz');
            seg = nib.load(data_path).get_fdata()

            for j in range(VOLUME_SLICES):
             X[j+(VOLUME_SLICES*c),:,:,0] = cv2.resize(flair[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))

             X[j+(VOLUME_SLICES*c),:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))


             y[j +VOLUME_SLICES*c,:,:] = cv2.resize(seg[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE))

        X = X.reshape(1,128,128,128,2)
        y = y.reshape(1,128,128,128)
        # Generate masks
        y[y==4] = 3;
        y = tf.one_hot(y, 4);
        #Y = tf.image.resize(mask, (IMG_SIZE, IMG_SIZE));
        #Y = np.array(Y).reshape(1,128,128,128)
        return X/np.max(X), y

training_generator = DataGenerator(train_ids)
valid_generator = DataGenerator(val_ids)
test_generator = DataGenerator(test_ids)
```

```
print(len(train_ids))
print(len(val_ids))
print(len(test_ids))
```

```
850
251
150
```

```
def showDataLayout():
    plt.bar(["Train","Valid","Test"],
    [len(train_ids), len(val_ids), len(test_ids)], align='center',color=[ 'green','red', 'blue'])
    plt.legend()

    plt.ylabel('Number of images')
    plt.title('Data distribution')
    plt.savefig('data2021.png')
    plt.show()

showDataLayout()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is cal
led with no argument.
```



Step 9: Adding callbacks.

```
csv_logger = CSVLogger('C:/Users/Computing/Desktop/archive/training.log', separator=',', append=False)


callbacks = [keras.callbacks.EarlyStopping(monitor='loss', min_delta=0,
                              patience=8, verbose=1, mode='auto'),
      keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                             patience=8, min_lr=0.000001, verbose=1),
      keras.callbacks.ModelCheckpoint(filepath = 'C:/Users/Computing/Desktop/archive/3D-UNet-2021model_.{epoch:02d}-{val_loss:.6f}.m5',
                            verbose=1, save_best_only=True, save_weights_only = True),
      csv_logger
    ]
```

Step 10: Compiling the model.

```
kernel_initializer =  'he_uniform'
```

```
input_layer = Input((128,IMG_SIZE, IMG_SIZE, 2))

model1 = Unet_3d(input_layer)
model1.compile(loss="categorical_crossentropy", optimizer=Adam(learning_rate=0.001), metrics = ['accuracy',tf.keras.metrics.MeanIoU(num
                                                                                                dice_coef, precision, s
                                                                                                dice_coef_edema ,dice_c


model1.summary()
```

Step 11: Training the model.

```
history =  model1.fit(training_generator,
                      epochs=100,
                      steps_per_epoch=len(train_ids),
                      callbacks= callbacks,
                      validation_data = valid_generator
                      )
model1.save("C:/Users/Computing/Desktop/archive/model_3dUnet_2mod.h5")
```

Step 12: Modified 3D U-Net Model Architectures.

For 4 number of filters with Kernel size 3*3*3

```
def Unet_3d(input_img, n_filters = 4,  batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 8 number of filters with Kernel size 3*3*3

```
def Unet_3d(input_img, n_filters = 8, dropout = 0.2, batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 16 number of filters with Kernel size 3*3*3

```
def Unet_3d(input_img, n_filters = 16, dropout = 0.2, batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 32 number of filters with Kernel size 3*3*3

```
def Unet_3d(input_img, n_filters = 32,  batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    #c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    #c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    #c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    #c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    #c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    #c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    #c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    #c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (3, 3, 3), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 4 number of filters with Kernel size 2*2*2

```
def Unet_3d(input_img, n_filters = 4,  batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    #c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    #c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    #c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    #c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    #c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    #c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    #c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    #c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 8 number of filters with Kernel size 2*2*2

```
def Unet_3d(input_img, n_filters = 8,  batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    #c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    #c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    #c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    #c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    #c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    #c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    #c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    #c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```

For 16 number of filters with Kernel size 2*2*2

```
def Unet_3d(input_img, n_filters = 16,  batch_norm = True):

    #Contraction path
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(input_img)
    #c1 = Dropout(0.1)(c1)
    c1 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c1)
    p1 = MaxPooling3D((2, 2, 2))(c1)

    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p1)
    #c2 = Dropout(0.1)(c2)
    c2 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c2)
    p2 = MaxPooling3D((2, 2, 2))(c2)

    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p2)
    #c3 = Dropout(0.2)(c3)
    c3 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c3)
    p3 = MaxPooling3D((2, 2, 2))(c3)

    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p3)
    #c4 = Dropout(0.2)(c4)
    c4 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c4)
    p4 = MaxPooling3D(pool_size=(2, 2, 2))(c4)

    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv3D(n_filters*16, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c5)

    #Expansive path
    u6 = Conv3DTranspose(n_filters*8, (2, 2, 2), strides=(2, 2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u6)
    #c6 = Dropout(0.2)(c6)
    c6 = Conv3D(n_filters*8, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c6)

    u7 = Conv3DTranspose(n_filters*4, (2, 2, 2), strides=(2, 2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u7)
    #c7 = Dropout(0.2)(c7)
    c7 = Conv3D(n_filters*4, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c7)

    u8 = Conv3DTranspose(n_filters*2, (2, 2, 2), strides=(2, 2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u8)
    #c8 = Dropout(0.1)(c8)
    c8 = Conv3D(n_filters*2, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c8)

    u9 = Conv3DTranspose(n_filters, (2, 2, 2), strides=(2, 2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(u9)
    #c9 = Dropout(0.1)(c9)
    c9 = Conv3D(n_filters, (2, 2, 2), activation='relu', kernel_initializer=kernel_initializer, padding='same')(c9)

    outputs = Conv3D(4, (1, 1, 1), activation='softmax')(c9)

    model = Model(inputs=input_img, outputs=outputs)

    return model
```