Brain Tumor Segmentation Project 50.039

Ryan Ng, Ruan Yang, Marcus Ng

April 13, 2024

Abstract

This paper introduces a deep learning-based system designed to identify brain tumors from Magnetic Resonance Imaging (MRI) scans. The project demonstrates the potential of deep learning in enhancing the medical field. The developed system is packaged in a graphical interface, allowing for practical usability. Our findings show promising results for the application of deep learning in enhancing tumor recognition and segmentation.

1 Introduction

Early detection of tumours is an important step in prolonging the survival time which is the start of treatment to the point of death. Magnetic Resonance Imaging (MRI) is often used in helping to obtain image scans of the brain. However, though MRI is capable of scanning for brain tumours, it requires a doctor's expertise to check manually if there are any present brain tumours in the MRI scans. This process can be very inefficient as doctors will have to go through each and every MRI scan. Furthermore, the diagnosis process in clinical settings often involves a subjective element because the doctors are constrained by the way they perceive the MRI scans and are incapable of directly extracting detailed feature information from manual segmentation results to quantify diagnostic outcomes accurately.

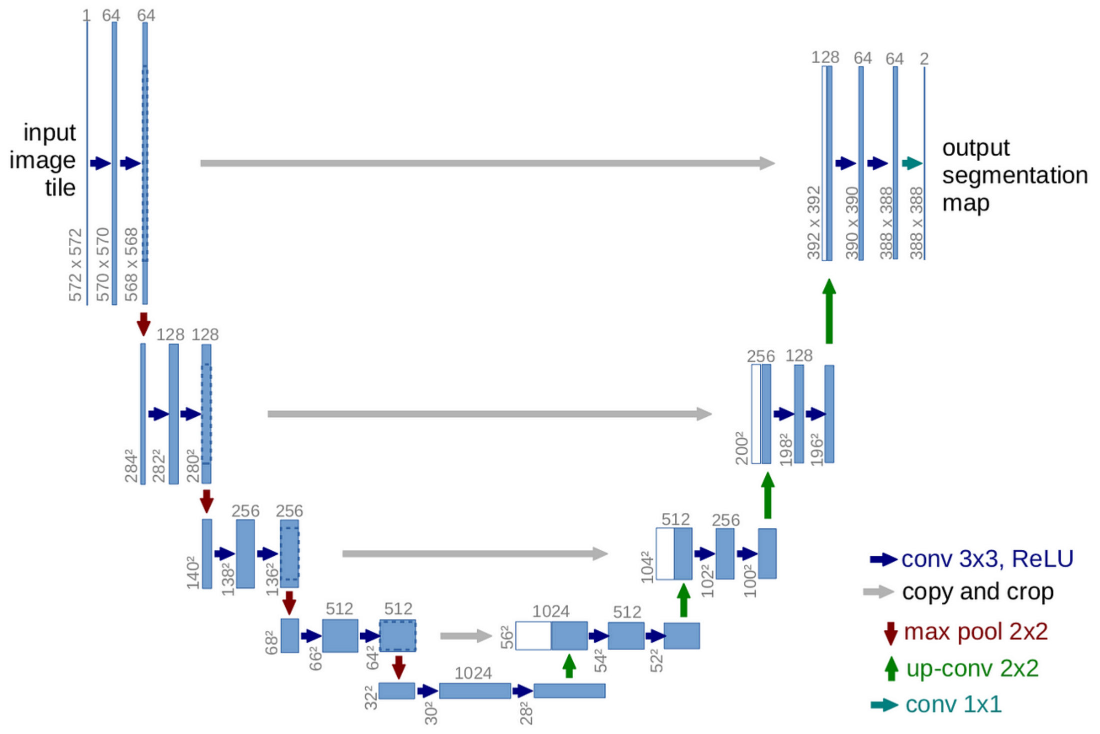
1.1 Objective

The primary aim of this project is to develop a model capable of using input MRI scans and perform segmentation on tumor locations.

1.2 Scope

This project focuses on segmenting tumors given MRI inputs of the following modes: Flair, T1, T1CE & T2.
The subsequent sections of this report detail the background and related work in tumor segmentation, describe the dataset and preprocessing methods employed, outline the model architecture and experimental setup, and present the results and findings of the project. Through this comprehensive exploration, the project contributes to the ongoing efforts to harness AI technologies for social good, particularly in enhancing the medical sector.

2 Model Design



2.1 Architecture

The model architecture used for the project is UNET. The UNET architecture is a transformer based model. There are equal numbers of encoder and decoder layers. The encoder (class DownConv) is defined as having a double convolution (class DoubleConv), which will be concatenated with its decoder (class UpConv) counterpart, the result of the double convolution will also be pooled before proceeding with the next double convolution. After the fourth pooling and double convolution, a convolution transpose is done before each decoder is used. Finally, the output of the last decoder layer is the segmentation map. The entire architecture is implemented in the UNET class, making use of the UpConv, DownConv and DoubleConv classes.

The inputs for the model would be 4 different MRI scans in the above stated modes, and each will be an individual channel for the model. The actual output is defined as a mask, which is an image showing where the tumor is in relation to the input MRI scans, hence the output of the model is the predicted mask.

3 Dataset Description and Pre-processing

3.1 Dataset Source

The following dataset was chosen for this project [AWS20], it consists of 369 different datapoints. Each datapoint is made up of 4 MRI scans, one of each mode, and one mask file, all 5 files are of .nii format.

3.2 Dataset Pre-processing

To further prepare the dataset, we augment the it using techniques such rotating the MRI scan. For each datapoint, that are 240x240x155 3D images, each image is split into 16 layers rather than the full 155. It is then saved, and rotated randomly 4 times, saving after each rotation. This new datapoint is then saved into a numpy file for easier loading. Once loaded, the dataset is finally seperated such that each layer is an individual datapoint, hence each datapoint is 5 seperate 240x240 images, representing the 4 modes and mask.

We split the dataset with a ratio of 80:10:10 to create training, validation and test sets.

4 Model Tuning and Testing

4.1 Loss

We tested the use of 2 losses both of which are useful for segmentation tasks.

4.1.1 Dice Loss

Dice Loss is derived from the dice cooefficient and is largely used in segmentation tasks relating to medical images.

4.1.2 Binary Cross Entropy With Logits Loss

Binary Cross Entropy with Logits Loss is used as we do not run our output through an activation function, hence it is more appropriate than using pytorch's nn.BCELoss().

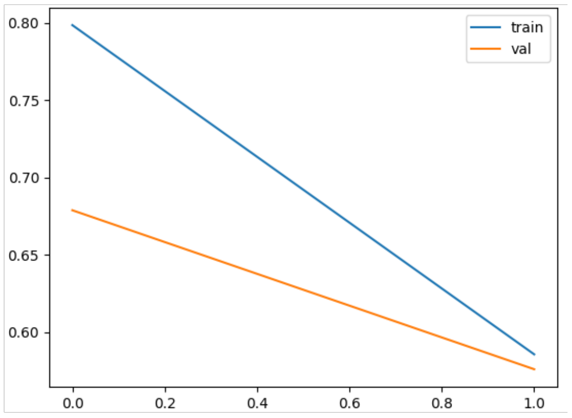
4.2 Model Variations

Below shows the different variations of the UNET model we have implemented, we used single channel models to decide on a suitable loss function, then proceeded with learning rate tuning. Graphs shown have a vertical axis representing loss and horizontal axis representing epoch number. The training losses are an average of the losses of all batches during each epoch.

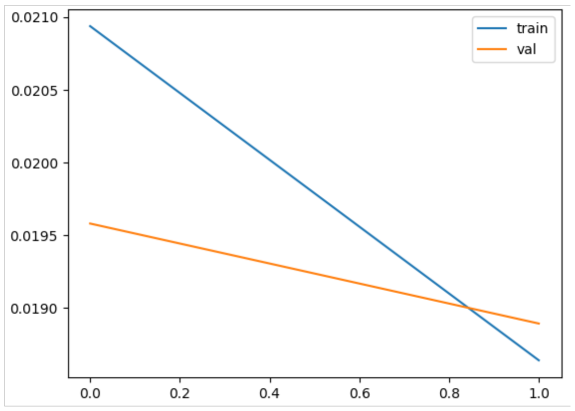
4.3 First Stage Testing

We test the following models with 2 epochs for basic testing, using only a single channel.

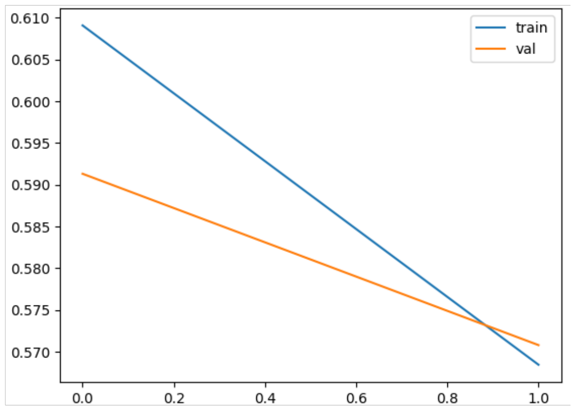
4.3.1 Model 1: Single Channel using Dice Loss



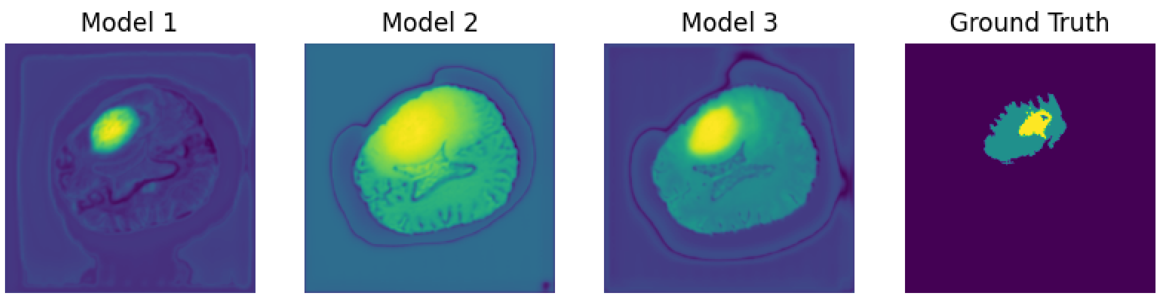
4.3.2 Model 2: Single Channel using BCEwithLogitsLoss

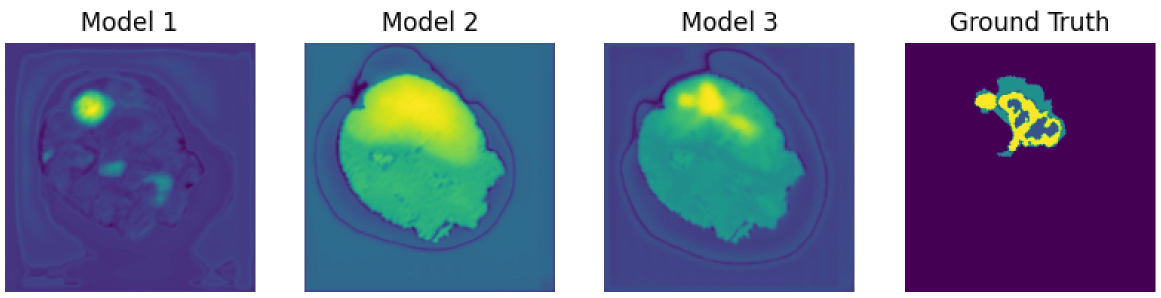


4.3.3 Model 3: Single Channel with Combined Loss



Although the test and validation losses for BCEwithLogitsLoss is the lowest, the use of Combined Loss allowed the model output mask of the model to be closest to the actual mask when tested manually. Below shows the comparisons of the models when using them to predict certain slices of the training set.

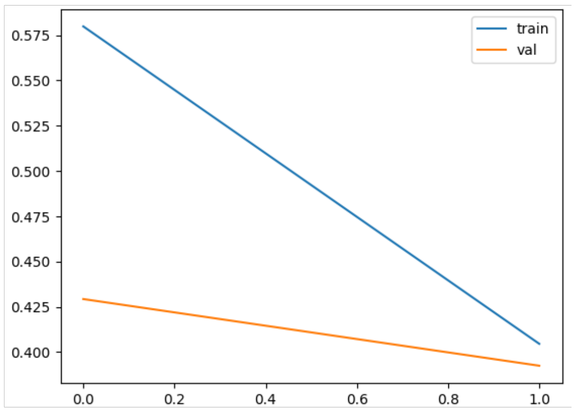




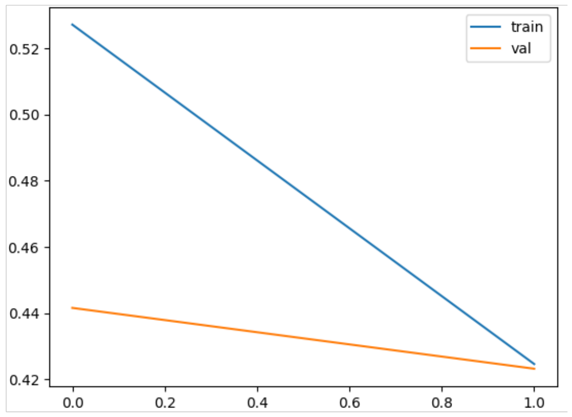
From observation, Model 1 is able to accurately predict the location of the tumors with less noise. Model 2 is able to better predict the size of the tumor but with more noise. Model 3 has the advantages of both in which it can capture a shape similar to the tumor as compared to Model 1 while having lesser noise compared to Model 2. Hence from this we decided to use Model 3 as the basis, using the Combined Loss as our loss functions for the proceeding models.

4.4 Second Stage Testing

4.4.1 Model 4: 4 Channel with Combined Loss (Learning Rate = 0.0001)



4.4.2 Model 5: 4 Channel with Combined Loss (Learning Rate = 0.001)

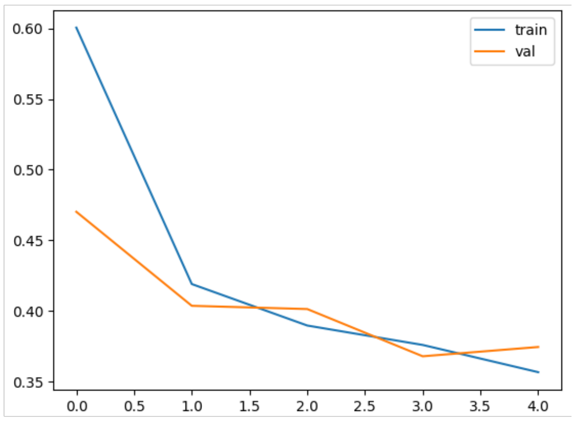


After evaluation of the losses, we found that a learning rate of 0.0001 was able to obtain lower losses as compared to a learning rate of 0.001. Hence we decided to use a model configuration of: 4 channels with combined loss & learning rate of 0.0001.

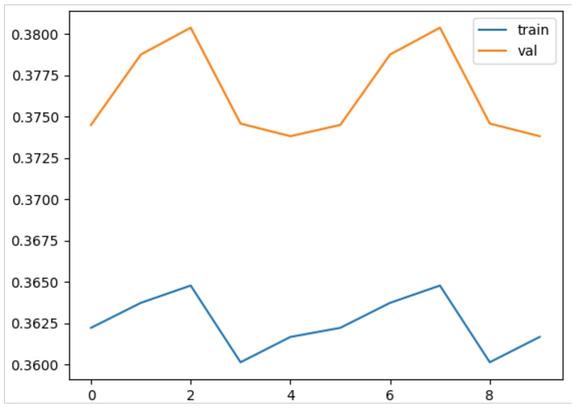
5 Final Model Training, Testing & Metrics

To finalise the project, we proceed with an initial 5 epochs of training followed by an additional 5 epochs of further training. Below depicts the graphs of loss.

5.1 Model 6: Final Training



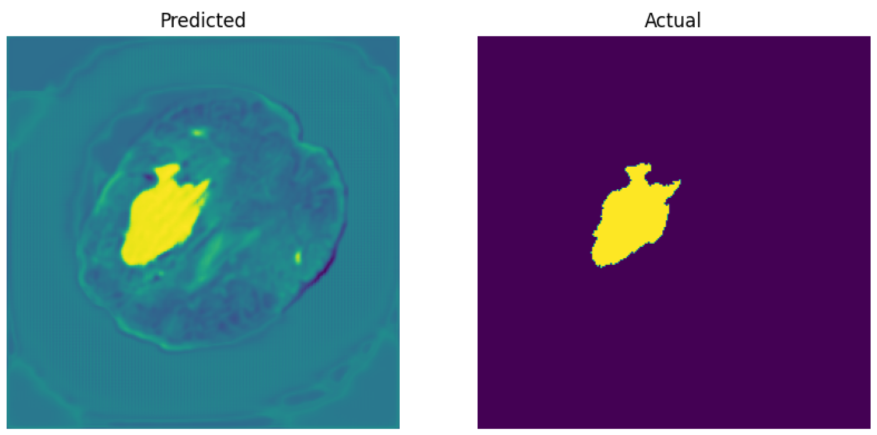
5.2 Model 7: Further Training

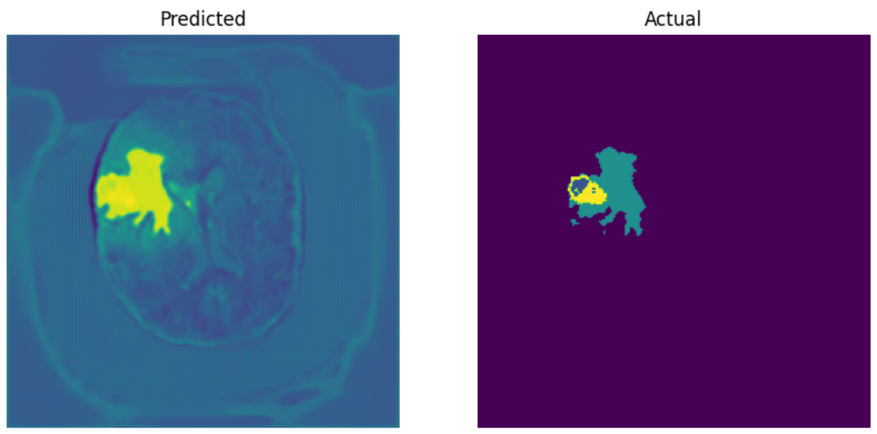


From the above graphs, we observe that further training after 5 epochs yields little to no improvements in loss.

5.3 Model Testing

Final loss from test set = 0.364





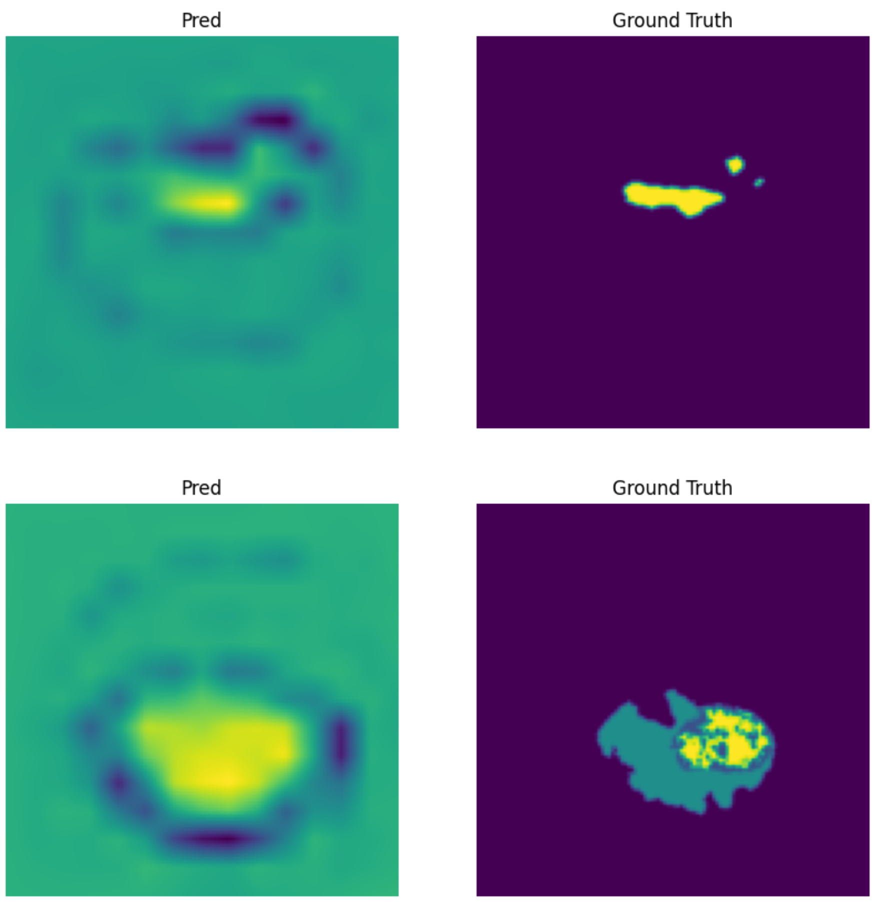
As observed from the above 2 example prediction slices, we observe that the model is able to accurately predict the location, shape and size of the tumor. Hence we can now conclude the model training and finish the project by packaging it in a usable GUI for users.

6 Comparison to State-of-the-Art Implementations

The following video shows a deep learning model using MatLab for brain tumor segmnentation [Sol23]. Observing the performance of the model, we can see that our model performs similarly, being able to predict the shape, location and size of the tumor.

7 Alternatives Explored

From our previous section, we got inspiration to try out state-of-the-art model. We explored the option of transfer learning by using pre-trained state-of-the-art model as it is too complex to train from scratch. One of the state-of-art models is DeepLab models. It have achieved state-of-the-art results on various semantic segmentation benchmarks like PASCAL VOC, Cityscapes, and COCO.
The pre-trained model that we used was DeepLab. The model was initiated with pre-trained weights from PyTorch and slight modifications were made to the model to suit our problem scenario. The changes include allowing the model to accept our dataset images and outputting the correct mask. The changes were necessary as the DeepLab model was trained on a different dataset that has RGB images.
We used the same pre-processing steps as previously. The image used in this model was shrank from 240 x 240 to 120 x 120 for time simplicity.
The results were generally decent because we can see the clear segmentation. It could be improved with higher resolution of images. Comparing it with our model, our model is still better in terms of segmentation with higher resolution.



8 How to run code

Download the code at the following Github Repo [Ng24a].

Please perform the following in order. Alternatively, refer to readme.md for instructions.

8.1 Setup

1. Download dataset from [AWS20]
2. Within the downloaded dataset, move the "BraTS2020_TrainingData" folder into the "Dataset" folder, we ignore the ValidationData folder as it does not contain the mask files as the dataset was meant for a competition. Next, ensure that the "BraTS2020_TrainingData" folder contains the "MICCAI_BraTS2020_TrainingData" folder. This then contains all 369 data points, each of which consists of the 4 MRI modes and mask file.
3. Download the saved models from the following drive [Ng24b]
4. Move the models folder into the project folder, ensuring the folder is named "saved_models"
5. Run code using the below steps

8.1.1 Run in Terminal

Create virtual env
- python3.9 -m venv venv

Activate Virtual Env
- source venv/bin/activate (mac)
- venv\Scripts\activate (windows command prompt)

Install required packages
- pip install -r requirements.txt

8.1.2 Run training.ipynb Jupyter Notebook

1. Ensure that the dataset_path variable is defined correctly.
2. Uncomment the third cell to generate the numpy dataset.
3. Run the proceeding cells sequentially. The notebook will visualise some data slices.
4. The models as stated above are commented out in the training file. You may uncomment them to train the model yourself, or use the models in the downloaded "saved_models" folder. The cells after each model can be uncommented as well to display the loss against epochs graphs for each trained model, after its respective model has been trained. These were the graphs used in the report.
5. Other predicted masks and visualisation can be observed within the training file as well.

8.1.3 Run comparison.ipynb Jupyter Notebook

1. Ensure that the dataset_path variable is defined correctly.
2. Run the proceeding cells sequentially. The final outputs show the predicted masks using the DeepLab architecture.

8.1.4 Disclaimer

There may be slight differences when training again, even after seeding there were minor differences such as the loss being approx 0.36 but not exactly 0.364.

8.2 Metrics

The metric used for evaluation would be the combined loss. This is mainly due to the Dice Coefficient commonly being used for measuring performance of models in medical image segmentation and use of accuracy measures such as pixel accuracy can be misleading.

9 Application

9.1 Steps

1. Ensure that virtual environment is activated
2. Ensure the model_path variable is correct in the app.py file. In this case running the very last model saved as "./saved_models/multi_channel_further/model_5.pth"
3. Start the application with - python3 app.py
4. Access any datapoint within the dataset folder, and upload the corresponding MRI mode to the correct box. For example the T1 MRI scan should be uploaded using the button "T1".
5. Click the "Analyse" Button to use the model to create the predicted mask.
6. Once completed, the mask can be observed. You may also choose to upload the actual mask (tagged as the seg.nii file) within data point folder.
7. The sliders can be used to view the different layers of the model.

9.2 Demo Video

A preview of the application and steps to run it can be viewed on this video [Ng24c]

10 Conclusion

In conclusion, the project demonstrates the potential that Deep Learning can be leveraged to solve problems within the medical field. The application demonstrates the accuracy that can be attained from training, and can be highly useful in aiding doctors or medical personnel in identifying tumours more quickly.

10.1 Future Improvements

The model could be used to accommodate different channels, such that whether users have 1 or 4 available MRI scans, they will still be able to use the model to get a preliminary prediction of where a tumor might be. Additionally, the model can be further improved with further hyper-parameter tuning, different loss functions and a modified architecture such as changing the number of encoder decoder pairs within the model. For training, we could increase the training batch from 16 to full 155 depth, but we did not do that in this project for time simplicity.

References

[AWS20] AWSAF. Brats2020 dataset (training + validation). *https://www.kaggle.com/datasets/awsaf49/brats20-dataset-training-validation*, 2020.

[Ng24a] Ryan Ng. 50.039 tumor project. *https://github.com/DarthPenguinz/50.039-Tumor-Project*, 2024.

[Ng24b] Ryan Ng. 50.039 tumor project saved models. *https://tinyurl.com/mpwvnpy9*, 2024.

[Ng24c] Ryan Ng. Brain tumor segmentation application. *https://tinyurl.com/ycxkp4d8*, 2024.

[Sol23] Pantech Solutions. 3d brain tumor segmentation using deep learning matlab. *https://www.youtube.com/watch?v=IHyWcZRctHst=334s*, 2023.

11 Contributions

Ryan Ng (1005869): Model development, Model Fine-tuning, Data pre-processing, Proposal, Final Report, Research
Ruan Yang (1006365) : Proposal, Research, Final Report, Model Architecture Testing, Transfer Learning (DeepLab V3)
Marcus Ng (1006118) : Proposal, Research, Final Report, Data Pre-Processing