Deep Learning-Based Brain Tumor Segmentation Images

Nikhil Raj, Evan Leendertse, Jose Martinez, and Amrut Kulkarni

Abstract. Accurate and efficient segmentation of brain tumors in MRI scans is critical for diagnosis, treatment planning, and clinical decision-making. In this study, we present a comparative analysis of four deep learning-based segmentation models—YOLOv8, DeepLabV3+, Mask R-CNN, and a custom-built U-Net—on a dataset comprising MRI images across four distinct tumor classes. The U-Net architecture, designed specifically for biomedical image segmentation, achieved the highest performance with an average Dice coefficient of approximately 70%. In contrast, YOLOv8 and DeepLabV3+ attained lower optimal Dice scores of 63% and 50%, respectively, which we attribute to the lack of pre-training on medical imaging datasets and limitations in model tuning.

1. Introduction

Image segmentation is a fundamental task in computer vision, aiming to assign a label to every pixel in an image such that pixels with the same label share certain characteristics. In biomedical imaging, especially brain tumor segmentation from MRI scans, precise pixel-level classification is critical for diagnosis and treatment planning.

In the current state of cancer care, oncologists are carefully examining radiographic images (ex. CT, MRI, PET/CT) to assess tumor sizes, location, and classification. As cancer cases continue to rise across United States, there is not enough time to determine each classification of cancer with the allotted time that pathologists have on a day-to-day basis. As a result, patients endure extended delays before going through the treatment phase with their oncologist.

As noted by Antonio Martinez, Lead Dosimetrist at the University of Texas–Rio Grande Valley, whom we consulted during this study, “the onboarding process of patients is increasing at a rapid pace, and there’s not enough time to carefully review the increasing number of scans for patients that require medical attention as soon as possible.” This observation highlights the growing need for automated and accurate segmentation tools in clinical workflows to support timely and reliable decision-making.

Presently, cancer diagnosis is a multistep process. When a patient starts experiencing symptoms, their doctor requests tests, normally in the form of imaging. These images will be examined by specialists for any masses that could be cancerous. If potentially cancerous tissue is detected, it will be biopsied to confirm. This process, especially the time between imaging and confirmation by biopsy, takes a long time and is at risk of human error. If a specialist does not find anything they consider suspect when cancer is in fact present, then the patient must spend more time without treatment and liable to spend more money if they seek subsequent evaluations.

There’s an opportunity to leverage deep learning, more particularly object segmentation, to learn various brain T1/T2 MRI images, the mask of the identified tumor location, and tumor classification, to assist oncology teams with their diagnoses and rapidly create therapy plans to quickly begin treating their patients.

Among various deep learning-based segmentation models, U-Net has emerged as one of the most influential architectures. Introduced by Ronneberger et al. in 2015 for biomedical image analysis, U-Net employs a symmetrical encoder-decoder framework complemented by

skip connections. These connections merge high-resolution features from the contracting path with upsampled decoder outputs, preserving spatial details that might otherwise diminish during down sampling. This mechanism enhances the model's precision in identifying fine structures [1].

A key advantage of U-Net is its effectiveness even with limited annotated datasets—a frequent challenge in medical imaging. Over time, researchers have developed multiple adaptations to further refine its performance, incorporating techniques such as attention modules [2], residual blocks [3], and densely connected skip pathways [4].

In the context of brain MRI segmentation, U-Net and its variants have consistently achieved leading results. For instance, Myronenko (2018) augmented the architecture with a variational autoencoder to better handle uncertainty in tumor segmentation [5]. Such innovations highlight U-Net's enduring versatility and effectiveness in evolving medical imaging applications.

In this study, we evaluated the performance of four segmentation models—YOLOv8, DeepLabV3+, Mask R-CNN, and a custom-built U-Net—on brain MRI images encompassing four distinct tumor classes. Among these, the U-Net architecture developed from scratch achieved the highest average Dice coefficient of approximately 70%. In comparison, YOLOv8 and DeepLabV3+ attained peak performances of 63% and 57%, respectively. Unfortunately, we did not get any meaningful results from Mask R-CNN because it could not be developed due to time and resource limitations.

2. Approach

2.1 U-Net

As discussed in the introduction section, our baseline model is a custom-made U-Net model. Figure 1 shows the overall architecture of the model. The model is inspired by *Ronneberger et al.* seminal work with few key changes as discussed in the later sections [1].

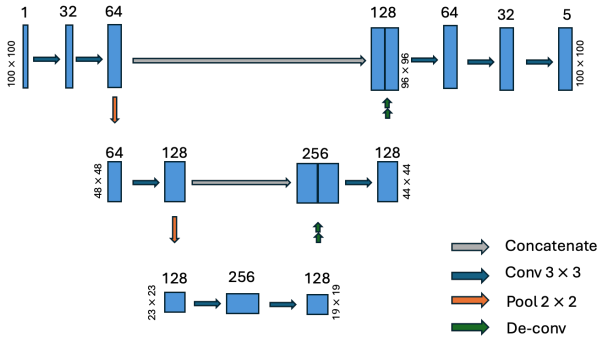

Figure 1. U-net Architecture applied for image segmentation task. The architecture is a smaller version of the model introduced by Ronneberger et al. with some key differences as discussed later.

2.1.1 Input image preparation

The input images were resized to 100×100 pixels while preserving their original aspect ratio to avoid geometric distortion. This was achieved by applying appropriate padding to make each image square prior to resizing. Additionally, the images were converted to grayscale, as MRI scans typically contain minimal color information. This conversion helped reduce the input dimensionality, enabling faster and more efficient training.

2.1.2 Convolution layers

The U-Net architecture comprises multiple convolutional layers, as illustrated in Figure 1. Each convolutional layer uses a 3×3 kernel and is followed by either a ReLU or Leaky ReLU activation function, depending on the experimental configuration. As a general design pattern, the contracting path of the U-Net employs convolutional layers that double the number of input channels, while the expanding path uses layers that halve the number of input channels—except for the final layer, which deviates from this pattern (see Figure 1)

2.1.3 Pooling layers

Both max pooling layers, shown in Figure 1, use a 2×2 kernel with a stride of 2. Pooling was introduced to reduce the spatial dimensions of the feature maps, enabling faster computation and promoting the summarization of local features. While pooling offers several advantages—such

as computational efficiency and better abstraction of local patterns—excessive down sampling can lead to the loss of fine-grained spatial information, which is critical for image segmentation. To mitigate this, pooling was used sparingly. Additionally, the input to each pooling layer was saved for use in skip connections (see Figure 1), which further aids in preserving local feature details during reconstruction.

2.1.4 Upsampling

Deconvolution layers, also known as transposed convolutions, were used for upsampling features in the expanding path of the U-Net architecture. This marks a significant departure from the original U-Net design. In the original model, encoder features were cropped to match the spatial dimensions of the decoder features before applying skip connections. We believe this cropping may lead to information loss, particularly at the image boundaries. To avoid this, we employed two deconvolution operations with carefully chosen kernel size, padding, and stride values to ensure that the encoder and decoder features align exactly in size. The parameters used for the first and second deconvolution layers are detailed below:

1st deconvolution layer: kernel size = 3, stride = 2, output padding = 0, padding = 0
2nd deconvolution layer: kernel size= 2, stride = 2, output padding = 1, padding = 0

These parameters were calculated using the following deconvolution size formulae:

$$O = (I - 1) \times S - 2P + K + O_p$$

Where I is input size, S is stride, P is padding, K is kernel size and O_p is output padding.

2.1.5 Skip connections

Two skip connections were established between the encoder and decoder pathways. These were implemented by directly concatenating the output of convolutional layers from the encoder with the corresponding output of the decoder's deconvolution layers (see Figure 1). Skip connections serve two important purposes: (1) they preserve high-resolution feature information from the encoder and pass it to the decoder, and (2) they facilitate improved gradient flow, helping to mitigate vanishing gradient issues. While vanishing gradients are less of a concern in our relatively shallow model, skip connections remain essential for preserving local spatial information critical to accurate segmentation.

2.1.6 Output layer

The final output layer is same size as the input image in width and height and contains five channels $(5 \times 100 \times 100)$, each channel representing the scores of four different tumors plus the normal tissue for each pixel.

2.2 YOLOv8 for Object Segmentation

In this section we describe YOLO model for image segmentation. YOLOv8-seg ("You Only Look Once") is a one-stage object detection, classification, and segmentation model. The notion of image segmentation was introduced in version 8 by featuring the prediction of pixel-level segmentation masks, thus consolidating all 3 object actions. Its one-stage architecture provides high computational efficiency, reduction in inference time, and high compatibility across a diverse range of custom datasets. Given these advantages, it's a versatile model that can be applicable for various scenarios, including our brain MRI image segmentation.

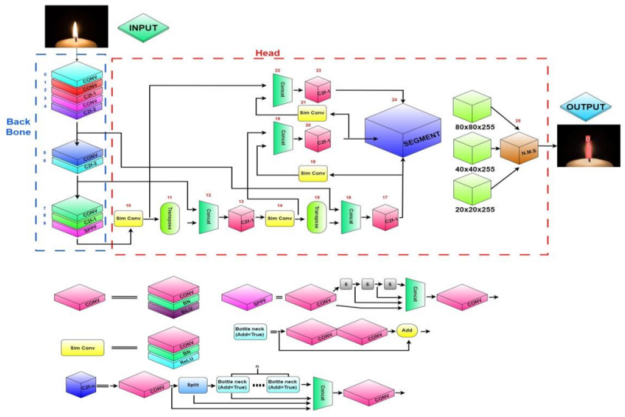

Figure 2 YOLOv8 segmentation architecture, adapted from ResearchGate

For data processing employed for this section please see Supplementary Information in Section 7.

2.3 DeepLabV3+

As shown in Figure 3, the DeepLabV3+ methodology uses a series of Atrous convolutions and upsampling with a set sequence of sizes. Additionally, as DeepLabV3+ extracts features from ResNet-50, a model trained on RGB images, we needed to duplicate the original image twice, to produce an image that appears as RGB format to the model. Thus, this model required input data of dimensions 256 x 256 x 3. These images were resized to preserve their original aspect ratio through padding. The information about model training, experimental setup, and discussion on results are provided in Supplementary Section (Section 7.4)

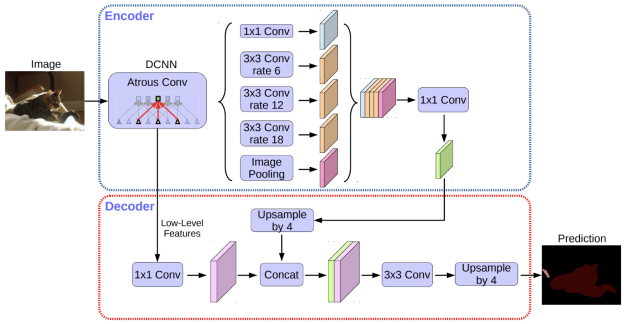

Figure 3. DeepLabV3+ model architecture for image segmentation

3. Experimental Setup

Experimental setup and configurations for U-Net are discussed in this section. For experimental configuration details for other models, please section supplementary information (Section 7)

3.1 Class balanced focal loss used for U-Net training

MRI tumor images naturally contain a high proportion of healthy tissue, with only small regions affected by tumor growth. This results in a significant class imbalance, where pixels corresponding to normal tissue vastly outnumber those representing tumor tissue. An analysis of pixel distribution revealed that normal tissue pixels were nearly 100 times more prevalent than tumor pixels. Additionally, the four tumor classes had approximately equal representation. To address this imbalance, we employed a custom loss function that combined focal loss with class-specific weighting. The loss formulation is shown below.

$$\boldsymbol{L} = \frac{\mathbf{1}}{\boldsymbol{BWH}} \sum_{x \epsilon \Omega} \sum_{i} \boldsymbol{w_i}\, \boldsymbol{y_i} (\mathbf{1} - \boldsymbol{p_{ix}})^{\gamma} \boldsymbol{log}(\boldsymbol{p_{ix}})$$

Where,
$B, W, and\ H$ are batch size, image width, and image height respectively
y_i is the one hot encoded true label for class i
w_i is weight for class i
γ is the focusing parameter
p_{ix} is the probability of pixel x and class i

3.2 Initialization and Optimization of U-Net model

Following Ronneberger et al., we applied Kaiming initialization to ensure stable training. This initialization samples weights from a Gaussian distribution:

$$x \sim N(0, \frac{2}{N})$$

Where N is the fan-in number of input units to the layer. The Kaiming initialization is widely used because it has been empirically shown to produce consistent gradient statistics.

For optimization, we use Adam optimizer with $\beta_1 = 0.9$ allowing faster adaptation to gradient updates given our relatively small batch size of 64. The learning rate was tuned as part of the hyperparameter search described later in Section 3.1.3.

3.3 Hyper-parameter tuning of U-Net model

To reduce computational overhead, we opted not to use an exhaustive grid search for hyperparameter tuning. Instead, we conducted a design of experiments study using a fractional factorial design. This approach allowed us to evaluate the effects of multiple hyperparameters on model performance while significantly reducing the number of required experiments. We selected four hyperparameters, each at three levels, as outlined in Table 2 in supplementary information (Section 7).

As discussed in Section 3.1.1, class weights were applied to address the class imbalance present in the dataset. Since the number of pixels corresponding to each tumor class was approximately equal, their weights were uniformly set to 1. However, determining the appropriate weight for the normal tissue class—which was significantly overrepresented—was less straightforward. Therefore, the weight for the normal class was treated as a hyperparameter and tuned experimentally. Three different values were tested, as shown in the first row of Table 2. All model variants were trained for 85 epochs.

4. Results and Discussion

4.1 U-Net Results

4.1.1 Hyper-parameter tuning

Following the recommendations of Ronneberger et al., ReLU activations were applied after each convolution and deconvolution layer. The model was then optimized using DoE approach described in Section 3.1.3. The Dice indices for the resulting experiments are presented in Table 3 (see Supplementary Section).

To assess the sensitivity of model performance to different hyperparameters, we conducted a simple linear regression analysis. Table 4 (Supplementary Section) reports the corresponding p-values for each hyperparameter. Among the parameters tested, only the weight assigned to the normal tissue class had a statistically significant effect on model performance (Dice index). The optimal hyperparameter combination is highlighted in Table 3, and using this configuration, we achieved a Dice index of approximately 65%.

Figure 4 displays the training and validation learning curves, as well as the Dice index progression for the optimal configuration. These plots indicate stable learning behavior, with convergence occurring at epoch 85.

4.1.2 Leaky ReLU activations

Next, we investigated the effect of different activation functions. As discussed in Section 2.1.2, in addition to ReLU, we also experimented with Leaky ReLU as an alternative activation function. This was a purely empirical exploration aimed at further improving model performance. Since the optimal hyperparameters had already been determined for the ReLU-based model, we retained the same settings and simply replaced the ReLU activations with Leaky ReLU.

Interestingly, this change led to a further improvement in performance. The model achieved a Dice index of approximately 70%, representing a ~5% increase over the ReLU-based model (see Figure 7 in Supplementary section).

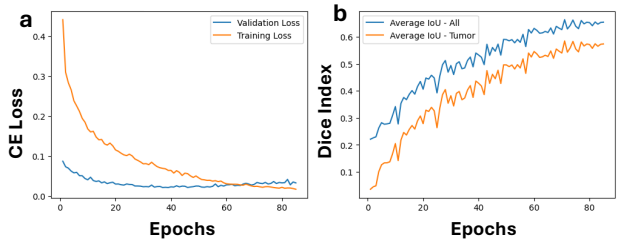

Figure 4. (a) Learning curves and (b) Dice index for U-Net with ReLU activation function after each convolution operation. The hyper-parameter used for generating these curves are highlighted in Table 3.

Figure 7 (a) shows the training and validation learning curves, along with the Dice index progression for the Leaky ReLU model. As shown in Figure 7 (b), the Dice index saturated at epoch 85. Notably, the validation loss exhibited an upward trend, even though the model's performance—as indicated by the Dice index—continued to improve. This divergence is likely due to the lower-class weight assigned to normal tissue. A possible explanation is that many normal tissue pixels, which were previously classified with high confidence, are now being classified with lower confidence due to the revised weight distribution. As a result, even though the pixel classifications remain correct (as reflected in the Dice index), the overall loss increases due to the way the loss function penalizes lower confidence predictions.

Table 1. Key performance metrics for each class of tumor as well as the normal tissues for the optimal hyper-parameters with Leaky ReLU activation. An average dice index of ~70% was achieved with the optimal model.

Tissue type	Dice Index	F1-score
Normal Tissue	0.983	0.991
Astrocytoma	0.459	0.459
Glioblastoma	0.506	0.672
Gliomas	0.709	0.709
Meningioma	0.799	0.888

4.1.3 Visual analysis of optimal model

Figure 5 below presents several examples comparing predicted and ground truth masks for all four tumor classes. Based on both visual inspection and the Dice index results discussed in the previous section, the model performs reasonably well—particularly considering the limited size of the training dataset.

To further evaluate the model's generalization capability, we tested it on a synthetic image containing multiple tumor types within a single frame. Although such cases are highly unlikely in real-world clinical scenarios, this experiment was inspired by broader applications of U-Net, such as in cell segmentation, where multiple object categories may be present in a single image. The goal was to assess whether our model, although trained on single-tumor images, could generalize to multi-class segmentation tasks.

To conduct this experiment, we created an artificial image by concatenating four separate tumor images—each representing a different tumor type—and resizing the composite image to 100x100 pixels (see Figure 6(a)). Figure 6 (b) and Figure 6(c) show the corresponding ground truth and predicted masks. Remarkably, the model was able to correctly locate the positions of tumor types with some error, despite never having been trained on such multi-class, multi-skull images. For Astrocytoma and Glioblastoma, the model also got the general shape of the mask correct. This result suggests that the model has learned both local and global feature representations effectively, enabling it to generalize beyond the conditions seen during training.

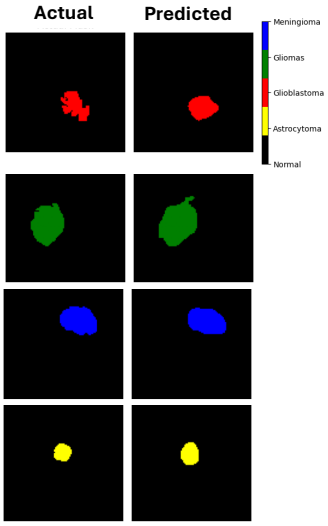

Figure 5. Examples of predicted and actual masks for different tumor types.

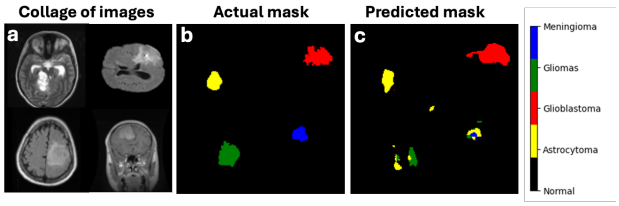

Figure 6. Generalization test on a synthetic multi-class tumor image: (a) A synthetic image constructed by assembling four separate MRI scans, each representing a different tumor class (Astrocytoma, Glioblastoma, Gliomas, and Meningioma). (b) The corresponding ground truth segmentation mask. (c) The predicted segmentation mask produced by the model trained exclusively on single-class tumor images. The color bar on the right shows the class labels used for visualization.

5. Experience/Challenges and Project Success

5.1 Experience and Challenges

The biggest challenge was the problem of class imbalance. When we tried the model without any weight, it will give zero accuracy for all tumor pixels and an accuracy of 1 for normal tissue. Although, it is not discussed in this paper, we also tried Class-Balanced Loss by Cui et al. with

various β values but none of the values were working. Finally, we had to painstakingly try different weights to come to the optimized solution. One thing we learnt was the importance of monitoring different metrices during training because our optimal model showed upward validation loss trend even though the mode performance was improving. This was extensively discussed earlier in Section 4.1.2.

5.2 Project Success

As expected, the U-Net model outperformed all other models evaluated on this dataset. We attribute this superior performance to the design of U-Net, which was specifically developed for medical image segmentation tasks. In contrast, the other models we tested—such as YOLOv8, DeepLabV3+, and Mask R-CNN—are primarily pre-trained on natural image datasets (e.g., MS COCO, ImageNet) and are optimized for object detection and segmentation in everyday scenes. Since brain tumors exhibit subtle and domain-specific features not commonly found in such datasets, these models struggle to generalize effectively without extensive domain adaptation.

6. References

1. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.

2. Oktay, O., et al. (2018). Attention U-Net: Learning Where to Look for the Pancreas.

3. Zhang, Z., Liu, Q., & Wang, Y. (2018). Road Extraction by Deep Residual U-Net. IEEE Geoscience and Remote Sensing Letters, 15(5), 749–753

4. Zhou, Z., Siddiquee, M.M.R., Tajbakhsh, N., & Liang, J. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation.

5. Myronenko, A. (2018). 3D MRI Brain Tumor Segmentation Using Autoencoder Regularization.

GitHub Repository for Code:

U-Net:
https://github.gatech.edu/akulkarni74/CS7643-Project-Brain-Tumor-Segmentation/blob/unet_final_branch/main.ipynb

YOLO:
https://github.gatech.edu/akulkarni74/CS7643-Project-Brain-Tumor-Segmentation/blob/jose-working-branch/main.ipynb

DeepLab:
https://github.gatech.edu/akulkarni74/CS7643-Project-Brain-Tumor-Segmentation/tree/DeepLabV3Plus/main.ipynb

MaskRCNN:
https://github.gatech.edu/akulkarni74/CS7643-Project-Brain-Tumor-Segmentation/tree/main/main.ipynb

Data Source:
https://www.kaggle.com/datasets/shubhamgajjar/brain-tumor-classification-2d/

Work Division

Student	Contribution aspect	Details
Nikhil Raj (Unet Model)	Introduction and background	Jointly contributed in project conceptualization, problem statement, and overall motivation and U-net literature survey.
	Approach	Conceptualized, designed, and developed the complete UNET model architecture. Wrote the code for UNET, including image pre-processing and data loader.
	Experimental results	Conducted all the experiments for UNET architecture including optimization, data analysis, and characterization.
	Experience and project success	Jointly contributed to this section
Jose Martinez (YOLO model)	Intro/Background	Jointly contributed in problem statement, background, and overall motivation
	Approach	Conceptualized and developed all processing of YOLOv8-seg model architecture under the models/YOLOv8 directory.
	Experimental Results	Conducted all experiments for YOLOv8-seg model architecture, including data processing, analysis, and characterization.
	Experience/Project Success	Jointly contributed to this section
Evan Leendertse (DeepLab Model)	Intro/Background	Jointly contributed in project conceptualization, problem statement, and overall motivation
	Approach	Conceptualized, designed and developed the DeepLab V3+ model.
	Experimental Results	Conducted experiments on DeepLab V3+ model, using testing code created by Nikhil.
	Experience/Project Success	Jointly contributed to this section.
Amrut Kulkarni (Mask RCNN model)	Intro/Background	Jointly contributed in project conceptualization, problem statement, and overall motivation
	Approach	Conceptualized, designed, and developed the complete Mask R-CNN model architecture. Wrote the code for Mask R-CNN, including image pre-processing and data loader.
	Experimental Results	Trained Mask R-CNN, but due to difficulty setting it up and slow training due to memory demands, the testing side of Mask R-CNN could not be explored before the project was due. Despite contributing to the codebase, much of it was not able to produce output useful for comparison with UNet or other models.
	Experience/Project Success	Jointly contributed to this section.

7. Supplementary Section

7.1 Supplementary Information for U-Net

Table 2. Hyper-parameters and their respective levels optimized using design of experiments.

Parameter	levels
normal tissue weight	0.05, 0.1, 1
γ	0, 1.5, 2
learning rate	0.001, 0.01, 0.1
Weight decay	0, 0.0005, 0.001

Table 3. Results of manual hyperparameter tuning using fractional factorial design of experiments of UNET model with ReLU activation. Four hyper-parameters, each with three levels, were chosen for tuning as shown in the first four columns of the table. The last column shows the dice index for each setting. Each model was trained for 85 epochs with Kaiming gaussian initialization.

normal tissue weight	γ	learning rate	weight decay	dice index
0.05	0.0	0.001	0.0	0.60
0.05	2.0	0.1	0.0005	0.18
1.0	0.0	0.1	0.001	0.15
1.0	2.0	0.001	0.0	0.15
0.1	1.5	0.01	0.0005	0.25
1.0	1.5	0.01	0.0	0.15
0.05	1.5	0.01	0.0005	0.22
0.1	2.0	0.01	0.001	0.20
0.1	0.0	0.001	0.0	0.68
0.1	1.5	0.001	0.0	0.66
0.1	1.25	0.001	0.0	0.65

Table 4. p-values obtained for hyper-parameters upon fitting a simple linear regression model on results shown in Table XX. Only normal tissue weight is shown to be statistically significant at 95% confidence interval.

normal tissue weight	γ	learning rate	weight decay
0.02	0.06	0.10	0.5

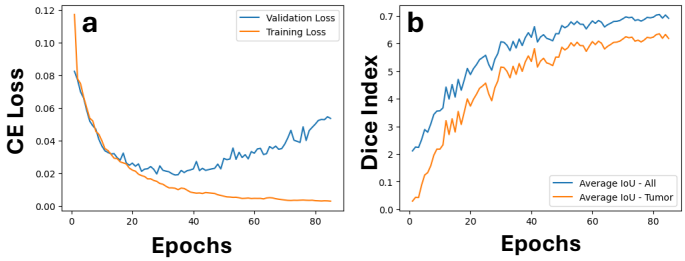

Figure 7. (a) Learning curves and (b) Dice index for UNET with Leaky ReLU activation function after each convolution operation.

7.2 Supplementary Information for YOLOv8 model

7.2.1 Strengths & Weaknesses

YOLOv8 segmentation is a widely adaptable model that can generalize easily to a wide range of datasets. As a one-stage model, inference runtime is significantly reduced when compared against other image segmentation models. In addition, YOLOv8 models are fast, accurate, and robust, making them favorable models to leverage, especially for our initiative. Finally, it can perform object detection, classification, and segmentation in unison, making it a well-rounded model.

On the contrary, YOLOv8-seg performance can suffer if the image sizes are of higher or lower resolution than the ideal 640px by 640px size. This is because the model is optimized to operate under the ideal resolution. Compared to two-stage segmentation models, YOLOv8 is not as adaptable to lower or higher resolution image data, thus potentially limiting the accuracy power of the overall model.

7.2.2. Data Preprocessing for YOLOv8

To fully accomplish the image segmentation process for YOLOv8, the data is split into two directories, images and labels. Each directory will have respective train/validation/test subdirectories for us to randomly shuffle our image/mask files and split 80%/10%/10%, respectively. We will also rename each file and set it to the proper file format to help ensure the exact MRI/mask pairs are created to process the data into their respective images/labels folders.

Finally, a yaml file is created to denote the path where the images and labels directories are located. This contains the data path as well as 4 different tumor classes and IDs for the YOLOv8-seg model to properly identify the data.

7.2.3 Experimental Setup for YOLOv8-seg

7.2.3.1 Configuration

YOLOv8-seg can be accessed via the Ultralytics package in Python. A preset yolov8s-seg.pt file can be leveraged as the initial instance of the model. This is the initial configuration and architecture of the YOLOv8-seg model as noted in Figure 2a. Once our data is fully preprocessed with our yolo_data.yaml file, we can begin defining our YOLO model and train with the proper parameters and hyper parameters.

7.2.3.2 Initialization

The “yolov8s-seg.pt” is a pretrained model from the Ultralytics package. It was initially trained with the COCO dataset, thus grounded on transfer learning literature. This is where speed and accuracy come into play - with pretrained weights, we can further fine-tune our YOLO model to our dataset with the respective hyperparameters.

7.2.3.3 Hyperparameter Tuning

YOLOv8-seg has about 48 different hyperparameters to fine-tune and configure with respect to our dataset. However, for the purposes of our initiative, we will only focus on 4 specific hyperparameters: initial learning rate (lr0), final learning rate (lrf), momentum, and weight decay. After fine-tuning the model over several iterations and examining the overall performance of each training run, the optimal hyperparameter values are as follows:

Table 5. YOLOv8-seg initial & optimal hyperparameters.

Parameter	Initial value	Optimal value
Initial learning rate	0.1	0.014
momentum	0.937	0.974
Weight decay	0.0005	0.0004

7.2.4 YOLOv8 results

The figure below shows the training and validation loss curves as well as the Dice Index for YOLOv8-seg. Given the summary statistics outputted by *Ultralytics*, the training and validation segmentation loss values were used to output the graph as the package's loss function relies on sigmoid and BCE loss. For the Dice Index, the mAP50(M) statistic is used here as this is the mask's mean average precision at the IoU threshold of 0.5, and it contains a strong correlation between raw IoU.

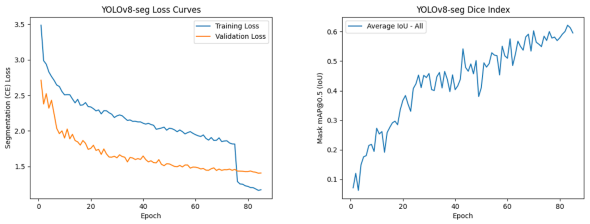

Figure 8: Learning curves and Dice index for YOLOv8-seg model architecture

Training loss stayed consistently decreasing up until the 77th iteration, where we see a significant decrease in the loss that is lower than the validation loss. This may be attributed to the package's variable hyperparameter changes, where some additional hyperparameters that were not explicitly stated in our script may be iterated to decrease the loss.

Average IoU shows significant epoch-over-epoch changes, showcasing possible high volatility in the IoU after each epoch. This may be attributed to the final learning rate or the slightly higher momentum value used when applying gradient changes at the end of each epoch. Because the goal of YOLO is to help generalize and converge much faster, these hyperparameters (and more) will be augmented after each iteration in order to achieve that goal.

After training the model over 85 epochs, the summary statistics are as follows. Note that the Dice Index scores are raw scores and not the mAP@0.5 (M) statistic as mentioned earlier. This is to help compare between other models in the next chapter.

Table 6: Overall and Per-Class Dice Index and F1-scores for YOLOv8-seg

Tissue type	Dice Index	F1-score
Normal Tissue	0.636	0.647
Astrocytoma	0.687	0.667
Glioblastoma	0.619	0.578
Gliomas	0.629	0.7004
Meningioma	0.622	0.7502

In terms of the Dice Index results, YOLOv8-seg tends to be consistent across all classes. Although Astrocytoma performed the best, all classes stayed relatively consistent amongst each other, having a total range of .069 between the least and greatest index. This helps support the notion that YOLO will generalize consistently fast whilst maintaining overall performance.

In terms of F1 score, Meningioma performed the best out of all the classes whereas Glioblastoma performed the least. The reasoning behind this performance may be due to the dataset, as each MRI scan varies between tumor types. Some images may have color, or others may present in black/white only. The direction of the scan may also be another factor involved as each scan has the patient facing a particular direction.

At the end of training, the YOLOv8-seg model produces a joint prediction of object classification, detection, and segmentation based on validation data. Here are a few examples of the ground truth mask vs prediction mask from our model:

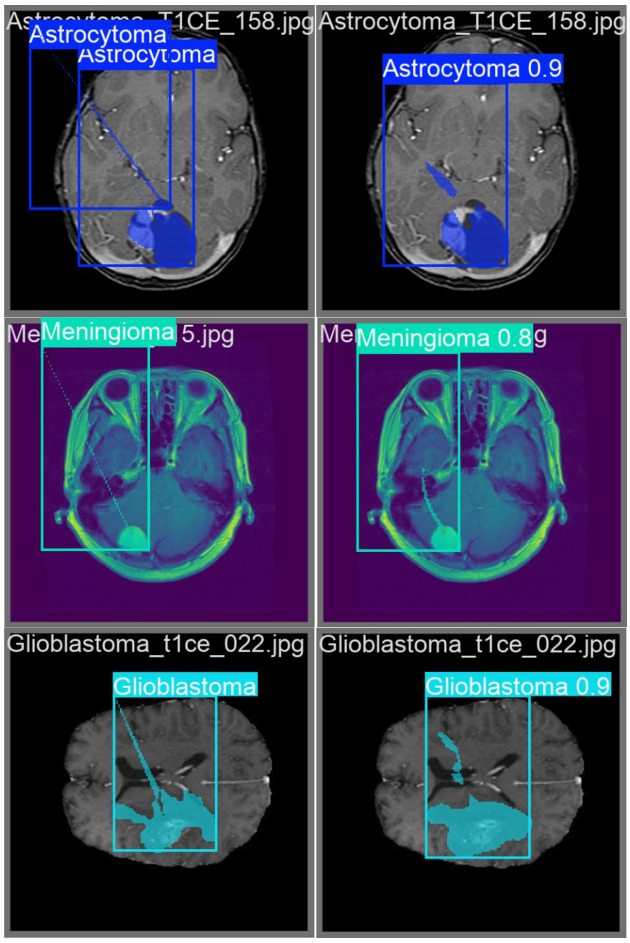

Figure 9: YOLOv8-seg Mask Pairs for Ground Truth (left) vs. Prediction (right)

Overall, as a one-stage segmentation model with pretrained weights, YOLOv8-seg performed quite well with our MRI image dataset. It's a highly efficient model that can generalize relatively fast and be able to perform multiple object functionalities at once. There may need to be more research done for all the hyperparameters included in its model structure, and more testing may have to be done to test different optimizers. Regardless, it was able to understand and learn about our data quite well, and the results of the model, given the computational power, were insightful to interpret, making it a great tool to use for (but not solely) object segmentation.

5.2.1 Experience and Challenges with YOLOv8

Working on YOLOv8 was an insightful experience because we got to learn how the data should be best tailored for its architecture. It felt intuitive to leverage the necessary components needed to perform our object segmentation analysis, and the results outputted from our model were remarkable.

Given the complexity behind the architecture, it was best to leverage the v8-seg architecture from the *ultralytics* library as the model components would take an extensive amount of time to replicate onto a custom implementation. The pretrained weights were also helpful in the process as the model can be grounded on transfer learning literature from the COCO dataset. Overall, from a user experience perspective, it felt naturally straightforward to leverage, dissect, understand, and implement for our objective.

Comparing our optimal YOLOv8-seg model against the U-Net model, U-Net outperforms YOLO from an overall perspective and for the Gliomas and Meningioma classes. YOLO slightly fares better when segmenting on the Astrocytoma and Glioblastoma classes, and this may be due to the consistent image structure from both Astrocytoma and Glioblastoma datasets. Both class MRI images are an aerial black/white image, thus helping the YOLO model learn consistent features alongside the segmentation masks. Another reason why U-Net performs better than YOLO is due to the initial weight that occurs. U-Net began training with random Gaussian weights whereas YOLO pretrained on the COCO dataset. Although the YOLO model has a form of transfer learning involved, it may be that the gradient descent with random Gaussian weights pushed the new weights downward in a steeper slope instead of laterally, thus performing faster than the pretrained COCO weights.

One of the main challenges involved with operating on the YOLO model is the hyperparameter tuning. In total, YOLOv8 has about 48 hyperparameters to fine-tune. If given the time to research all different hyperparameters and identify what each

parameter does to impact the model, we can properly fine tune all of them. However, for the specific purpose of our initiative and given our coursework, it was best to only keep things simple and tune only 4 of the 48 hyperparameters. This ensures consistency across comparison analyses and eliminates the need to dive into extraneous investigations.

Another challenge identified was the export of custom statistics. The *ultralytics* library outputs various statistics after training the YOLO model. However, when it came to creating a specific plot or calculating metrics such as the dice index, it required a separate script to be written in order to extract the desired information. It was not as intuitive as the model is marketed to be, but it wasn't as difficult to extract any customized metrics that are needed for our comparison analysis. More research may need to be done within this package to determine how to gather custom metrics in a simpler manner.

7.3 Supplementary Information for Mask R-CNN ResNet50

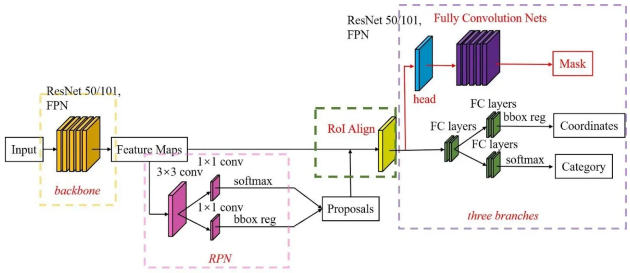

Fig 10. Mask R-CNN architecture

7.3.1 Background

Mask R-CNN is the latest in a line of R-CNN based models developed by Kaiming He and his team at FAIR. R-CNNs (Region-based Convolutional Neural Networks) perform Object detection by splitting the input image into ~2000 regions and passing each one sequentially through CNNs to classify each region. While this method works, it is very memory intensive, so each subsequent version developed by He and his FAIR team, Fast R-CNN and Faster R-CNN, served to make the process more efficient. Fast R-CNN Improved on this by taking the entire image and regions at the same time. Faster R-CNN improved on this further using what is called the Region Proposal Network (RPN). The RPN takes an image as input and extracts features from it and finds specific Regions Of Interest in the image through ROI pooling. It passes those areas of interest through layers meant to classify the areas and calculate bounding box dimensions. Mask R-CNN is a slight modification of Faster R-CNN to both improve performance and to include an extra output head to calculate the segmentation mask. Instead of ROI Pool, Mask R-CNN uses ROI Align, which aligns the regions of interest with the feature map extracted from the image to improve performance.

7.3.2 Strengths & Weaknesses

Compared to R-CNN, Fast R-CNN, and Faster R-CNN, Mask R-CNN is more efficient and allows for segmentation as well as the class labeling and bounding box object detection of Faster R-CNN. The feature map generated by the backbone network and RPN preserve the granularity of R-CNN's analysis of an image while making it more efficient. ROI Align provides greater performance than ROI Pooling since it aligns the regions generated by the RPN to the feature space of the original image, reducing the chance of an object class going undetected or false detections from proposed regions not fully encompassing relevant extracted features. However, due to high memory usage, Mask R-CNN is still a memory expensive model to train. The size of the datasets had to be significantly reduced to keep training time manageable.

7.3.3 Data Preprocessing

Due to the high memory requirements for Mask R-CNN, a smaller number of images had to be used per class, between 300-400, depending on what different imaging types were used in each class. After this there were 1258 images across all 4 classes. This was split into training and validation sets, with 1006 training images and 252 validation images.

7.3.4 Experimental setup for Mask R-CNN ResNet50 and Results

7.3.4.1 Configuration

This project used maskrcnn_resnet50_fpn from Pytorch's torch vision library. By default, the model is pretrained on the Cocos dataset, which includes benefits that come with transfer learning from that dataset.

7.3.4.2 Initialization

The Mask R-CNN used for this assignment was initialized using a pretrained ResNet-50 backbone with weights in subsequent layers randomly initialized. This allows for some benefits of transfer learning but also allows the model to adjust the remaining weights to an optimal point for our dataset. A pre-trained backbone also reduces the amount of training time needed since the backbone only requires fine tuning.

7.3.4.3 Hyperparameter Tuning

Table 7. Mask R-CNN initial & optimal hyperparameters.

Parameter	Initial value	Optimal value
Initial learning rate	0.1	0.00001
ROI Threshold	0.5	0.00001

7.3.5 Mask R-CNN Results

Due to difficulty getting Mask R-CNN up and running and the memory requirements slowing down training, nothing beyond training and validation could be performed for this project. The calculation for loss is slightly different for Mask R-CNN compared to other models used for segmentation since it is the sum of across all of Mask R-CNN's output heads, classification/labeling loss, bounding box loss, and mask loss.

7.4 Supplementary Information for DeepLabV3+ model

7.4.1 ResNet-50 Backbone

This model most famously extracts features from the 1st and 4th block of the ResNet-50 pretrained convolution model, although other backbones are available, and it is possible to make use of features from other blocks of the backbone.

7.4.2 Atrous Spatial Pyramid Pooling

DeepLabV3+ models apply the concept of Atrous convolutions, which uses an R-value to gather context through varying ranges of the image. The *r* value inserts *r* - 1 zeros in between consecutive filter values. This means that although the convolution size may be 3 x 3 values, it will span a length of five values on each axis of the image if *r* is equal to two.

These Atrous convolutions, consisting of feature maps from various ranges are then concatenated together for later upsampling and convolutions.

7.4.3 Configuration & Initialization

The DeepLabV3+ model can access the ResNet-50 model through PyTorch, leveraging pretrained weights based on ImageNet data. By calling features of block one and block four, you may apply the remaining transformations on these features.

7.4.4 Hyperparameter Tuning

Similar to Section 3.1.3, class weights were applied to address the class imbalance present in the dataset. However, when lowering the weights from the 1st class, where there is no tumor data, the model performed far worse on that category with no significant increase in other category performances. Therefore, we remained with the weights adjusted to match the class imbalance.

Table 8. DeepLabV3+ initial & optimal hyperparameters.

Parameter	Initial value	Optimal value
Learning rate	0.1	.0001
Batch Size	32	16
Class Weights	1, 1, 1, 1, 1	0.58, 1.32, 0.87, 0.691, 1.54
Weight decay	0.0005	0.0004

4.3 DeepLabv3

The DeepLabV3+ overall resulted in dice scores with an overlap of about 50% between the three categories, 98% with the normal, and most prevalent data and 31% with Meningioma. Despite parameter tuning and weight adjustments, this model continued to struggle with Meningioma whereas other models studied performed just as well or better on this category than others.

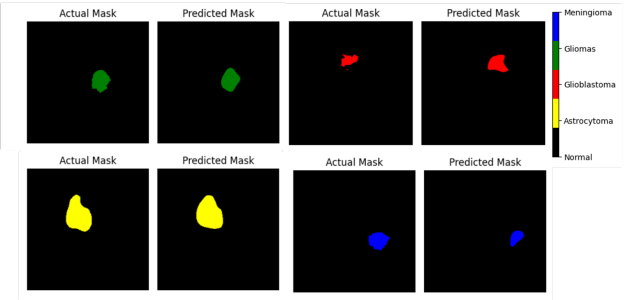

Figure 11. Examples of predicted and actual masks for different tumor types.

As this model is very compute heavy, we decided to cut the epochs after twenty, the loss curve only marginally decreased for training and did not reliably decrease for validation loss. Additionally, the Dice index began to approach stagnation.

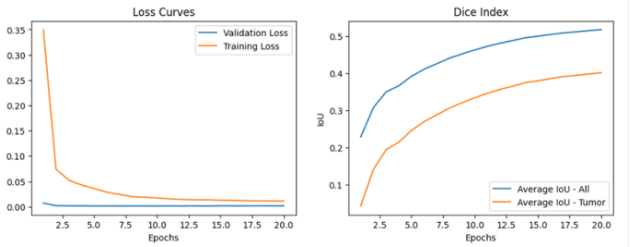

Figure 12 a) Learning curves and (b) Dice index for DeepLabV3+ model.

Table 9. Performance matrices for the optimal hyper-parameters

Tissue type	Dice Index	F1-score
Normal Tissue	0.984	0.992
Astrocytoma	0.518	0.683
Glioblastoma	0.523	0.687
Gliomas	0.521	0.685
Meningioma	0.310	0.473