Deep Learning-Based Multi-Compartment Brain Glioma Segmentation:Using U-Net, and Residual U-Net Architectures

BY:
Bala krishna Ragannagari

Introduction

Gliomas, particularly diffuse gliomas, are the most common malignant primary brain tumors in adults. Characterized by their infiltrative growth within the central nervous system, these tumors pose significant challenges in treatment and monitoring due to their variability in biological behavior and response to therapy. The cornerstone of post-treatment monitoring is MRI, which provides essential details on tumor size, location, and morphological changes. This project aims to segment multi-compartment gliomas from post-treatment MRI scans, focusing on identifying specific regions such as Enhancing Tissue (ET), Non-enhancing Tumor Core (NETC), Surrounding Non-enhancing FLAIR Hyperintensity (SNFH), and Resection Cavity (RC). We explore the efficacy of deep learning models, particularly U-Net and Residual U-Net, using the BraTS 2024 dataset to enhance automated glioma assessment and treatment planning.

This project focuses on segmenting multi-compartment gliomas from post-treatment MRI scans, aiming to identify regions such as:

- **Enhancing tissue (ET)**: Active tumor regions that show contrast enhancement.
- **Non-enhancing tumor core (NETC)**: Necrotic and cystic tumor areas.
- **Surrounding non-enhancing FLAIR hyperintensity (SNFH)**: Tumor infiltration and edema.
- **Resection cavity (RC)**: Space left after surgical removal of tumor tissue.

In this project, we explore and compare three deep learning models— **U-Net**, and **Residual U-Net**—for this segmentation task using the **BraTS 2024 dataset**. We aim to evaluate the models based on accuracy metrics like **Dice coefficient**,.And i worked on data handling ,preprocessing techniques and implementing the 3D U-Net architecture.

Dataset Overview

We employed the Brats 2024 dataset for our analysis of post-treatment glioma outcomes. This dataset encompasses records from 1350 patients, each with four distinct MRI scans and a segmented image for comprehensive analysis. The types of MRI scans included are:

- **T1c or T1-Gd (T1-weighted post-contrast):** Utilized for detecting Enhancing Tissue (ET), which appears bright and highlights active tumor cells and regions where the blood-brain barrier has been compromised.
- **T1w or T1n (T1-weighted or T1 native):** Important for identifying Non-enhancing Tumor Core (NETC), which shows up as dark on scans indicating areas of necrosis or cysts. These regions may appear brighter on T1 native scans compared to T1-Gd, without any enhancement.
- **T2w (T2-weighted):** Used to observe Surrounding Non-enhancing FLAIR Hyperintensity (SNFH), seen as hyperintense areas that include edema, infiltrative

tumor growth, and other treatment-related changes, while excluding signs of chronic ischemic damage.
- **T2f (T2-weighted FLAIR):** Also highlights SNFH, providing additional contrast that helps distinguish various pathological features from one another.

Image Specifications

Each MRI scan in the dataset is saved in a NIfTI (.nii) file format with the following dimensional structure:

- **First Index (182):** Represents 182 slices in the sagittal plane, providing a side view of the brain.
- **Second Index (216):** Consists of 216 slices in the coronal plane, offering a front view.
- **Third Index (182):** Comprises 182 slices in the axial plane, presenting a top-down view of the brain.

Tissue Classification in MRI Scans

- **Enhancing Tissue (ET):** Bright on T1-Gd images, signifying active tumor presence.
- **Non-enhancing Tumor Core (NETC):** Dark on T1 and T1-Gd images but brighter on T1 native images, indicative of necrotic or cystic regions.
- **Surrounding Non-enhancing FLAIR Hyperintensity (SNFH):** Hyperintense on T2 and T2f images, encapsulating edema and infiltrative tumor aspects.
- **Resection Cavity (RC):** Varies with the cavity's age; chronic cavities match the intensity of cerebrospinal fluid, while recent ones may contain air, blood, or proteinaceous materials and show variable signal characteristics.

Preprocessing Techniques

To enhance the performance and manage computational efficiency of our model on the Brats 2024 glioma MRI dataset, we implemented the following preprocessing steps:

Image Resizing: The MRI images were resized to (132, 132, 116) to simplify data handling and reduce computational load, while preserving essential features.

Z-Score Normalization: This technique standardized the pixel intensity values across all scans, setting the mean to zero and standard deviation to one. This step is critical for improving model convergence and reducing sensitivity to input variations.

Data Stacking: Considering the 3D nature of MRI data, we combined all four MRI scans (T1c, T1n, T2w, T2f) for each patient into a single comprehensive dataset. This approach allows the model to learn from complex spatial relationships and enhances its ability to accurately classify tumor-related features.

Data Augmentation Techniques

To augment our dataset and increase the robustness of our model, we employed several data augmentation techniques:

Zooming: Applied a 20% zoom to emphasize critical features within the MRI scans, particularly focusing on tumor areas.

Cropping: Following zooming, 80% of each image was cropped to maintain focus on the central and most informative parts of the scans.

Rotation: Each scan was rotated by 45 degrees to introduce variability and better simulate different imaging conditions encountered in clinical settings.

These preprocessing and augmentation techniques are designed to optimize data quality and model training effectiveness, ensuring robustness and accuracy in our tumor classification efforts.

The use of rotation as an augmentation technique on different MRI images is seen in the image above. In each category, the photographs on the right display the augmented versions after rotation, while the ones on the left depict the original scans. There are noticeable differences between the original and rotated photos upon close inspection, especially inside the segmented images.

Modelling:

3D U-Net Model Architecture

The two primary paths of the 3D U-Net architecture—the analysis path and the synthesis path—are tailored for volumetric segmentation and are derived from the classic U-Net architecture.

The analysis path (encoder): consists of a sequence of convolutional layers, with two 3×3×3 convolutions in each layer. To add non-linearity, a ReLU activation function is then applied. To decrease the spatial dimensions and boost the representational capacity, a 2×2×2 max pooling operation with strides of two is conducted in each dimension after the convolutions.

Synthesis Path (Decoder): The synthesis or decoding path features layers of up-convolutions with a 2×2×2 kernel and strides of two in each dimension. Each up-convolution is followed by two 3×3×3 convolutions, each paired with a ReLU activation. This path gradually restores the spatial dimensions to match the original input size.

Shortcut Connections: To enhance feature propagation and reduce information loss, shortcut connections are employed from corresponding layers of equal resolution in the analysis path to the synthesis path. These connections help in preserving important spatial hierarchies necessary for accurate segmentation.

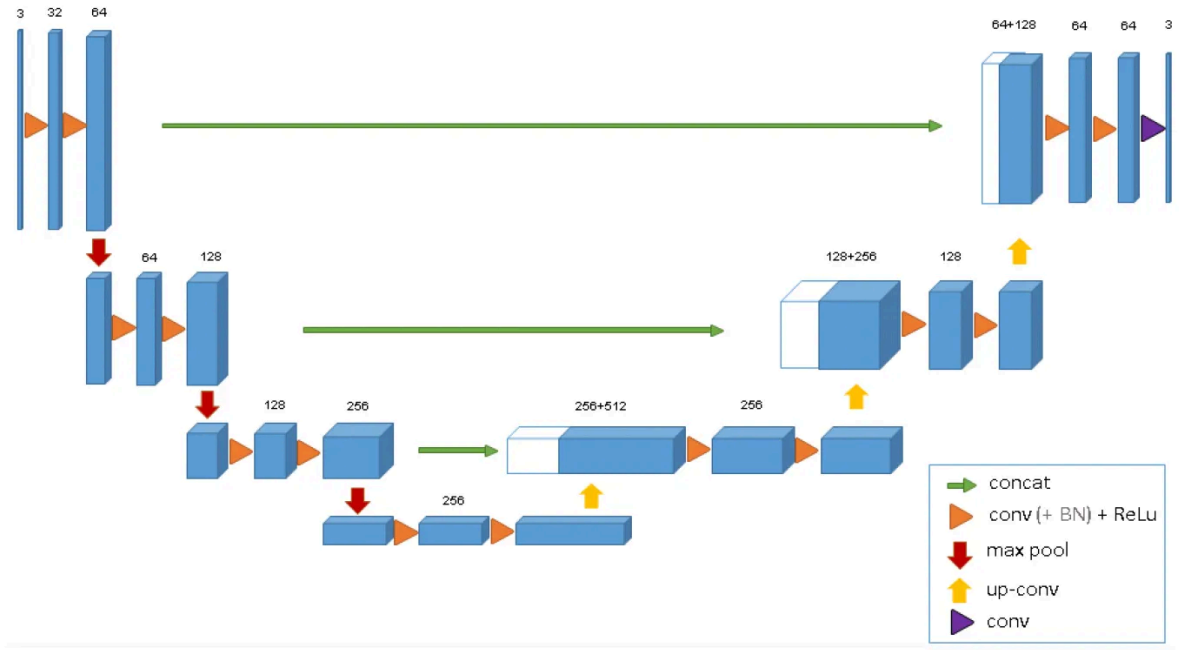


Fig: Standard U-Net architecture

Model summary:

```
UNet3D(
  (inc): DoubleConv(
    (double_conv): Sequential(
      (0): Conv3d(4, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
      (4): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
    )
  )
  (down1): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool3d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (1): DoubleConv(
        (double_conv): Sequential(
          (0): Conv3d(64, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(128, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (4): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): ReLU(inplace=True)
        )
      )
    )
  )
  (down2): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool3d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (1): DoubleConv(
        (double_conv): Sequential(
          (0): Conv3d(128, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(256, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (4): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): ReLU(inplace=True)
        )
      )
    )
  )
  (down3): Down(
    (maxpool_conv): Sequential(
      (0): MaxPool3d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (1): DoubleConv(
        (double_conv): Sequential(
          (0): Conv3d(256, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
          (3): Conv3d(512, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
          (4): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (5): ReLU(inplace=True)
        )
      )
    )
```

```
  )
  (up1): Up(
    (up): Upsample(scale_factor=2.0, mode='trilinear')
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv3d(1024, 512, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (1): BatchNorm3d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv3d(512, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (4): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )
  (up2): Up(
    (up): Upsample(scale_factor=2.0, mode='trilinear')
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv3d(512, 256, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (1): BatchNorm3d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv3d(256, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (4): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )
  (up3): Up(
    (up): Upsample(scale_factor=2.0, mode='trilinear')
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv3d(256, 128, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (1): BatchNorm3d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv3d(128, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (4): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )
  (up4): Up(
    (up): Upsample(scale_factor=2.0, mode='trilinear')
    (conv): DoubleConv(
      (double_conv): Sequential(
        (0): Conv3d(128, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (1): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv3d(64, 64, kernel_size=(3, 3, 3), stride=(1, 1, 1), padding=(1, 1, 1))
        (4): BatchNorm3d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): ReLU(inplace=True)
      )
    )
  )
  (outc): OutConv(
    (conv): Conv3d(64, 5, kernel_size=(1, 1, 1), stride=(1, 1, 1))
  )
)
```

3D U-Net Architecture Overview

Input Specifications: The model processes inputs of dimension (4, 166, 132, 132), suitable for volumetric medical imaging data.

Encoding Path: Consists of four encoder stages, each with two convolutions followed by batch normalization and ReLU activation. Feature depths increase progressively through the stages: 64, 128, 256, and 512, with each stage followed by a max pooling operation to reduce spatial dimensions.

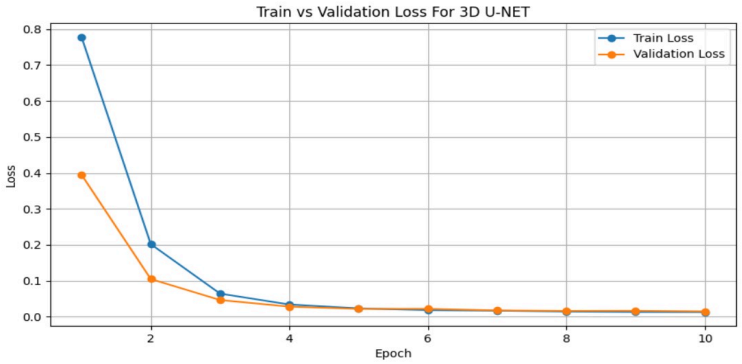
Bottleneck: Deep feature processing occurs here with a feature depth of 512, focusing on extracting the most abstract features.

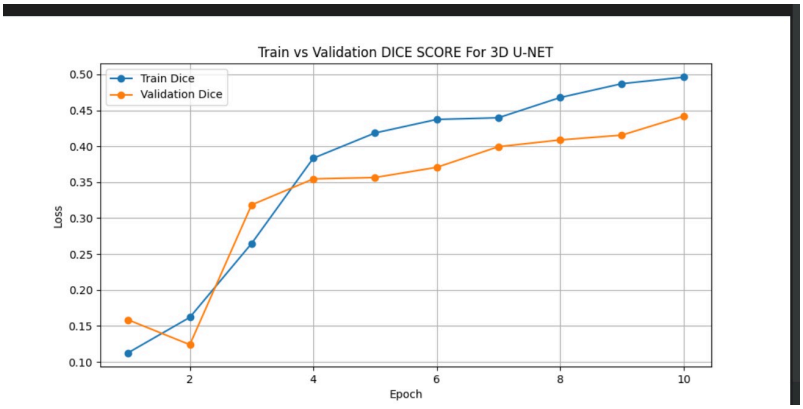
Decoding Path: Includes four decoders that upsample and concatenate features from corresponding encoders using skip connections, crucial for restoring spatial details lost during encoding.

Output: The final convolutional layer maps features to five output classes, representing different segmentation labels.

Significance: This architecture excels in precise volumetric segmentation by maintaining high-resolution details through skip connections, essential for medical image analysis.

Results:






```
Validation Loss: 0.0185
Validation Dice Scores: [0.09596776191859811, 0.6938994060505826, 0.2970572503655307, 0.5244747322478809]
```

metric	value
Validation Loss	0.0164
Dice Scores	0.45
Enhancing Tumor (ET)	0.095
Non-enhancing Tumor Core (NETC)	0.297
Surrounding Non-enhancing FLAIR Hyperintensity (SNFH)	0.693
Resection Cavity (RC)	0.524

Model Training and Validation Performance Overview

Loss Reduction Over Epochs:

- **Training Loss:** Initiated at 0.8, the training loss saw a significant decrease, concluding at approximately 0.004 by the tenth epoch. This notable reduction reflects the model's effective adaptation and learning from the training data.
- **Validation Loss:** Commencing at 0.4, the validation loss consistently declined to 0.0105 by the tenth epoch, indicating the model's robust generalization capabilities on unseen data.

Dice Coefficient Analysis:

- **Overall Performance:** The Dice scores for both training and validation demonstrated consistent improvement over ten epochs. Training Dice scores ascended from roughly 0.15 to 0.45, while validation scores rose from about 0.10 to 0.40. This steady progress underscores the model's consistent learning and generalization.
- **Class-specific Performance:**
    - **Class 2 and Class 4:** Representing specific tissue types, these classes exhibited superior segmentation performance, highlighted by higher Dice scores.
    - **Class 1 and Class 3:** Corresponding to enhancing tissue and non-enhancing tumor core, these classes recorded lower Dice scores. The results suggest potential areas for enhancement, possibly necessitating further model refinement or additional data to boost accuracy.

Summary:

Implemented the standard 3D U-Net architecture, along with preprocessing techniques.The model which initially included 3 encoders and 3 decoders.. This configuration was chosen to evaluate how our dataset performed with a commonly used model setup in biomedical image segmentation. After verifying the initial model's performance.we have tried to enhance the model architecture by extra encoder and decoder and check how the model performs .After comparing the results of both models ,the model with 4 encoders and 4 decoders is performing good and able to identify different classes where previous model could not classify .Inorder to enhance the performance of the model more we introduced residual blocks in the u net architecture

Originally intended to run for 20 epochs, each exceeding an hour, technical constraints limited the training to just 10 epochs. Despite these challenges, I employed early stopping to optimize the training duration and successfully computed the performance metrics for the available epochs. Where we got a validation loss of 0.016 and aa dice co efficient of 0.45.

There are few more improvements that can be implemented in future like more advanced preprocessing techniques like identifying the location of the tumor using segmented image and and apply some transformation and also there are many advancements happening like vision transformer and many more.We will try to implement it using different architectures.

Percentage of code used from other sources:

I used 40% of code from others sources . Remaining 60% i implemented on my own , but i used chat gpt to resolve my errors which I could not understand .

References:

1) https://arxiv.org/pdf/1606.06650
2) https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10635180&tag=1
3) https://www.sciencedirect.com/science/article/pii/S1361841524002056