Comparison of a Smaller Specialized Segmentation Model (U-Net) to an Advanced Fine-Tuned Model (TransUNet)

Vinay Kumar
MAI Programme
University of Amberg-Weiden
Amberg

July 3, 2024

1 ABSTRACT

The project involves segmenting MRI images using U-Net and TransUNet models. These models are essential in deep learning for identifying regions of interest in MRI scans, crucial for any medical application. The dataset consists of preprocessed MRI images with their masks, which were quality-checked and balanced. The experiments demonstrated that biomedical image segmentation tasks could be implemented using models like U-Net and TransUNet, where additional dependencies create compact layers through convolution and up-sampling. Data preprocessing, addressing issues like missing data and sample balancing, played a significant role, especially in image resizing. After training and validating the models, unbiased validation of the developed PyTorch models ensured accurate and robust MRI image segmentation. The primary challenges included handling missing data, unbalanced datasets, and resizing images. This project's success in medical image segmentation using U-Net and TransUNet paves the way for further investigation into various designs for broader applications in medical imaging.

2 INTRODUCTION

Medical image segmentation is one of the most significant tasks in medical imaging, enabling precise identification and location of regions or structures of interest. Accurate segmentation of MRI images is crucial for several clinical applications in disease diagnosis, treatment planning, and observation. This project implements deep learning techniques on U-Net and TransUNet architectures for high-accuracy MRI image segmentation. The focus on these models aims to improve the segmentation procedure by yielding more well-defined and reliable results.

Note: All these tasks were performed on OTH KI-Server.

3 PROJECT PLANNING

The project is divided into several phases to ensure structured progress and timely completion. Each phase is described below:

- Project Initiation and Setup
- Data Collection and Preprocessing
- Model Development
- Model Training and Evaluation
- Results Analysis and Visualization
- Conclusion and Future Aspects

3.1 Project Initiation and Setup

The initial phase of the project involved defining the objectives, scope, and deliverables. This included a comprehensive literature review to understand the current state of medical image segmentation and the selection of suitable deep learning models. The virtual environment was set up for this task, but due to the large dataset, all tasks were performed on OTH KI-Server, which already had the necessary software libraries and tools such as PyTorch installed.

3.2 Data Collection and Preprocessing

3.2.1 Source of Data

The MRI images and corresponding masks used in this project were downloaded from Kaggle. The dataset is available at this Kaggle link [1].
The dataset contains brain MRI images along with manual FLAIR abnormality segmentation masks. The images were obtained from The Cancer Imaging Archive (TCIA) and correspond to 110 patients included in The Cancer Genome Atlas (TCGA) lower-grade glioma collection with at least fluid-attenuated inversion recovery (FLAIR) sequence and genomic cluster data available. Tumor genomic clusters and patient data are provided in the 'data.csv' file.

3.2.2 Data Preprocessing

- Images and masks were read using OpenCV, a library for handling computer vision tasks. The code for data loading and preprocessing was utilized from this Kaggle Code [2].
- Some images were discovered that lacked corresponding masks, and similarly, there were masks without matching images. We used counters and lists to track these discrepancies, removing these incomplete data points from both the training and evaluation processes.
- The dataset was balanced by maintaining an equal number of positive samples (images with masks) and negative samples (images without masks). Achieving this balance is essential for training a robust segmentation model.
- A significant challenge was resizing the images while maintaining the integrity of the regions of interest. Images and masks were resized to a consistent dimension suitable for input into the U-Net and TransUNet models.
- Some images were discovered that lacked corresponding masks, and similarly, there were masks without matching images. Data augmentation techniques were applied to artificially increase the size of the training dataset and improve model generalization.

```
n_files = 0

# Iterate through subdirectories in the root directory
for directory in [os.path.join(root, x) for x in os.listdir(root) if os.path.isdir(os.path.join(root, x))]:
    for file in os.listdir(directory):
        file_path = os.path.join(directory, file)
        n_files += 1
        img_shape = np.array(cv2.imread(file_path)).shape
        img_dimensions.append(img_shape)

        if 'mask' not in file:
            mask_path = os.path.join(directory, f"{file.rsplit('.', 1)[0]}_mask.tif")
            if not os.path.exists(mask_path):
                no_mask += 1
                no_mask_files.append(file_path)

        else: # Store mask dimensions
            mask_shape = np.array(cv2.imread(file_path, cv2.IMREAD_UNCHANGED)).shape
            msk_dimensions.append(mask_shape)
            img_path = os.path.join(directory, f"{file.split('_mask')[0]}.tif")
            if not os.path.exists(img_path):
                no_file += 1
                no_files.append(file_path)

            mask_image = cv2.imread(file_path, cv2.IMREAD_UNCHANGED)
            if np.max(mask_image) > 0: # Non-empty mask
                num_nonempty_masks += 1
                if len(nonempty_mask_samples) < S:
                    nonempty_mask_samples.append(file_path)
            else: # Empty mask
                num_empty_masks += 1
                if len(empty_mask_samples) < S:
                    empty_mask_samples.append(file_path)
```

Figure 1: Data defining

- The MRI images show cross-sectional views of the brain, while the masks highlight specific regions of interest, such as tumor areas
- MRI images are in color, providing detailed anatomical information with various intensity levels to highlight different tissue types
- The images and masks are resized to a consistent dimension, in this case, 224x224 pixels.
- Some MRI images have corresponding masks with segmented regions (highlighted in yellow in the provided image). These regions represent abnormalities or areas of interest, such as tumors
- The dataset includes a variety of brain scans, showing different slices and anatomical structures

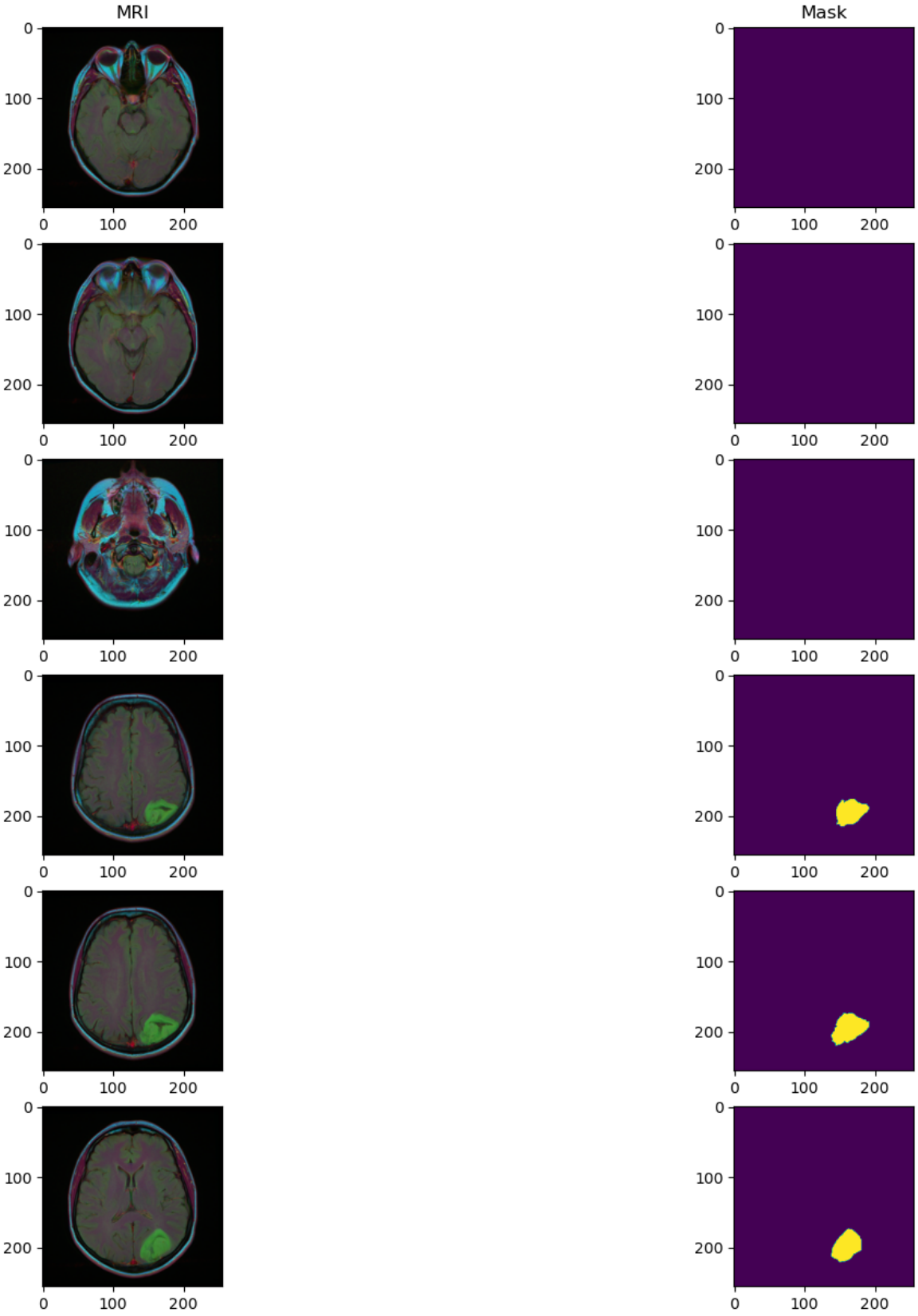

Figure 2: MRI Images

3.3 Data Loaders

Data loaders are essential for efficiently managing and feeding data into models during training and evaluation

- Training deep learning models on entire datasets at once is computationally impractical due to memory constraints
- To ensure that the model does not learn any unintended sequence or pattern in the data, data loaders can shuffle the dataset before each epoch
- Data loaders automatically create batches of data from the dataset, removing the need for manual batch creation and ensuring consistent batch sizes
- By leveraging multiple workers, data loaders can load and preprocess data in parallel, significantly speeding up the data pipeline and ensuring the GPU is efficiently utilized without waiting for data
- Data loaders can apply transformations and augmentations to the data in real-time during training
- There is a consistent and uniform way of handling data across different phases of the training pipeline (training and validation), ensuring that the model receives data in a standardized format

3.4 Model Defining

3.4.1 Overview of U-Net

U-Net is a convolutional neural network architecture designed for image segmentation tasks. It consists of two main parts: the contracting path (encoder) and the expanding path (decoder). This architecture is particularly effective for biomedical image segmentation due to its ability to capture both local and global features. The code is available at this where I have copied the architecture Github link [3]

- **DoubleConv:** This block performs two consecutive convolution operations, which help in learning more complex features. ReLU activations introduce non-linearity, allowing the model to capture non-linear patterns.
- **DownSample:** Combines double convolutions with max-pooling. Max-pooling reduces the spatial dimensions of the feature maps, helping the network to capture hierarchical features and reducing computational complexity.
- **UpSample:** Upsamples the feature maps using transposed convolution, then applies double convolutions. The upsampled feature map is concatenated with the corresponding feature map from the encoder, which helps in retaining spatial information.
- **UNet Class:**
  - **Encoder Path:** The encoder path consists of sequential downsampling blocks, which progressively reduce the spatial dimensions while increasing the depth of the feature maps.
  - **Bottleneck:** Acts as a bridge between the encoder and decoder, processing the most compressed form of the feature maps.
  - **Decoder Path:** The decoder path consists of upsampling blocks, which progressively increase the spatial dimensions while decreasing the depth of the feature maps

– **Output Layer:** A final convolutional layer reduces the number of channels to the desired number of output classes, producing the segmentation map.

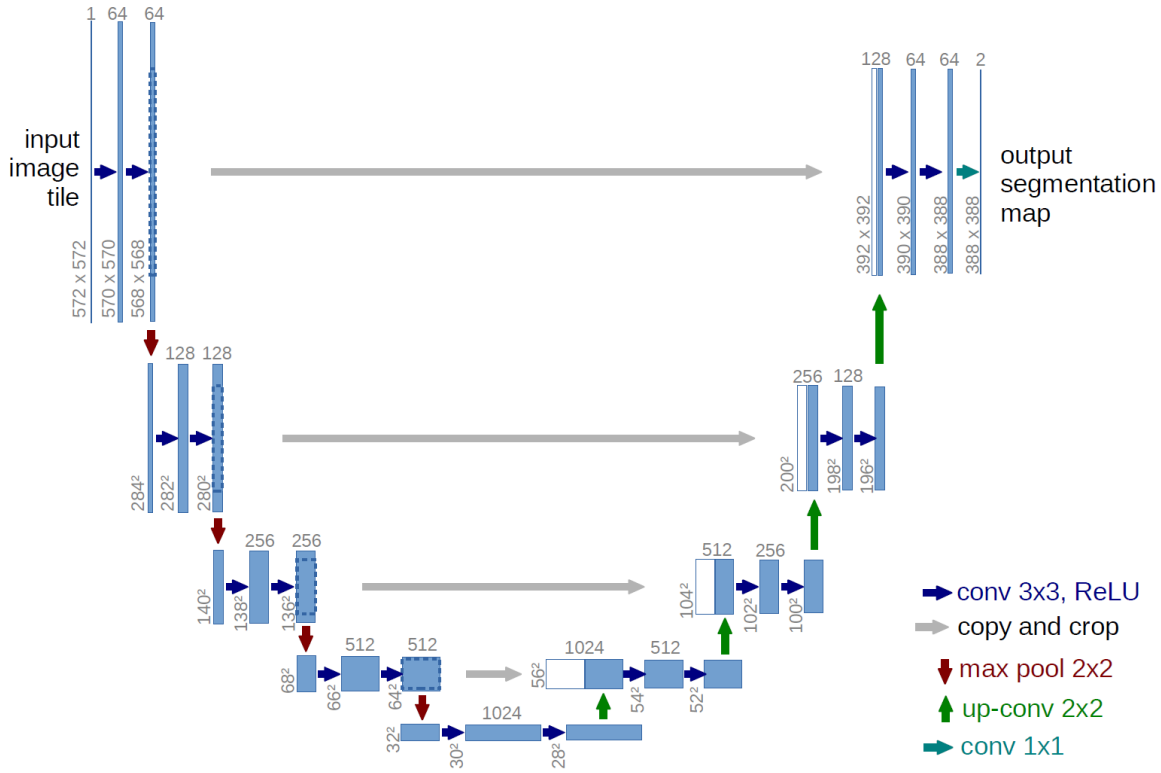

Figure 3: U-Net Architecture Source: 3

U-Net architecture is symmetrical in layout, featuring skip connections, and is very suitable for image segmentation. The following code snippet initializes such an architecture in PyTorch using double convolutions, downsampling, and upsampling processes connected by skip links. The architecture enables large-scale feature capturing.

To detect and distinguish between various parts within a picture, the U-Net architecture makes use of this specific characteristic. Because precise segmentation of anatomical structures is essential in medical picture interpretation, this makes it very helpful. In general, U-Net is an encoding-decoding structure whose downsampling path allows for context capturing and its upsampling path allows for exact localization.

The skip connections result in high resolution characteristics from the contracting path, these features need to be combined with the upsampled output for more accurate segmentation. The accuracy and reliability of the final segmentation results should increase because U-Net will be trained on a range of picture segmentation tasks in this PyTorch implementation.

```python
class UNet(nn.Module):
    def __init__(self, in_channels, num_classes):
        super().__init__()

        # Left side (contracting path / encoder)
        self.down_convolution_1 = DownSample(in_channels, 64)
        self.down_convolution_2 = DownSample(64, 128)
        self.down_convolution_3 = DownSample(128, 256)
        self.down_convolution_4 = DownSample(256, 512)

        # Bottom neck (bottleneck)
        self.bottle_neck = DoubleConv(512, 1024)

        # Right side (expanding path / decoder)
        self.up_convolution_1 = UpSample(1024, 512)
        self.up_convolution_2 = UpSample(512, 256)
        self.up_convolution_3 = UpSample(256, 128)
        self.up_convolution_4 = UpSample(128, 64)

        # Output layer
        self.out = nn.Conv2d(in_channels=64, out_channels=num_classes, kernel_size=1)

    def forward(self, x):
        # Down-sampling / encoder path
        down_1, p1 = self.down_convolution_1(x)  # First down-sampling
        down_2, p2 = self.down_convolution_2(p1)  # Second down-sampling
        down_3, p3 = self.down_convolution_3(p2)  # Third down-sampling
        down_4, p4 = self.down_convolution_4(p3)  # Fourth down-sampling

        # Bottleneck
        b = self.bottle_neck(p4)  # Bottleneck layer

        # Up-sampling / decoder path
        up_1 = self.up_convolution_1(b, down_4)  # First up-sampling
        up_2 = self.up_convolution_2(up_1, down_3)  # Second up-sampling
        up_3 = self.up_convolution_3(up_2, down_2)  # Third up-sampling
        up_4 = self.up_convolution_4(up_3, down_1)  # Fourth up-sampling

        # Output layer
        out = self.out(up_4)  # Output convolution to get the final prediction
        return out
```

Figure 4: U-Net Model

3.4.2 Overview of TransUNet

TransUNet is a hybrid architecture that leverages both Vision Transformers (ViTs) and Convolutional Neural Networks (CNNs). The ViT captures global context by processing the entire image as a sequence of patches, while the CNN decoder reconstructs the segmentation map by upsampling the feature maps obtained from the ViT

- The Transformer component helps in understanding the global context of the image, which is crucial for segmenting complex structures in medical images.
- By combining CNNs and Transformers, TransUNet can potentially achieve higher accuracy in segmentation tasks compared to using CNNs alone
- The architecture can be adapted for various image segmentation tasks beyond medical imaging

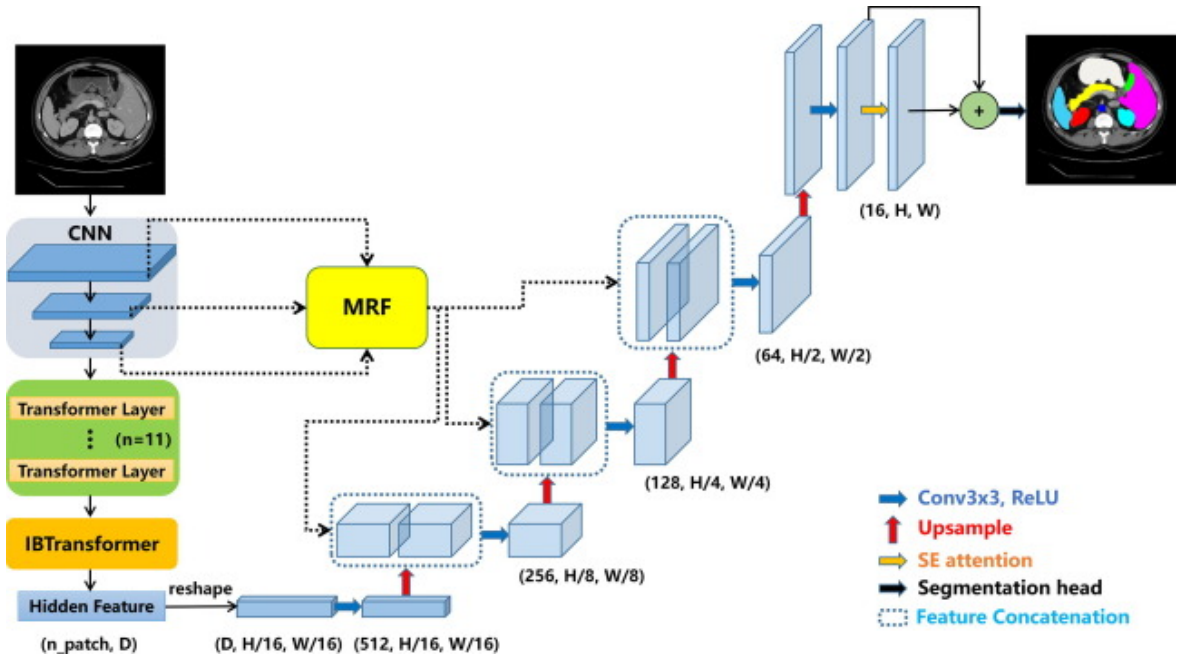

Figure 5: TransUNet Architecture Source: 4

- Combines the local feature extraction capabilities of CNNs with the global context understanding of Transformers.
- Utilizes a Transformer encoder to capture long-range dependencies in the input image.
- Uses a convolutional decoder (similar to U-Net) to reconstruct the high-resolution segmentation map.
- Incorporates skip connections to retain spatial information from the encoder layers.

```python
# Define the TransUNet model
class TransUNet(nn.Module):
    def __init__(self, num_classes):
        super(TransUNet, self).__init__()
        #Loads the ViT model with pre-trained weights, set to output only the feature maps.
        self.encoder = timm.create_model('vit_base_patch16_224', pretrained=True, features_only=True)

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(768, 512, kernel_size=2, stride=2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, num_classes, kernel_size=1)
        )

    def forward(self, x):
        features = self.encoder(x)
        x = features[-1]  # Use the last feature map from the encoder
        x = self.decoder(x)
        return x
```

Figure 6: TransUNet Model

I implemented a fine-tuning model using TransUNet for medical image segmentation. The TransUNet model combines a pre-trained Vision Transformer (ViT) encoder with a CNN decoder. The ViT encoder, loaded with pre-trained weights, extracts global features from the input images, while the CNN decoder upsamples these features to reconstruct high-resolution segmentation maps

Fine-tuning allows leveraging the learned features from the pre-trained ViT, enhancing the model's performance on the specific segmentation task. This hybrid approach captures both global context and local details, making it effective for accurate and detailed medical image segmentation.

3.5 Model Training and Evaluation

- The model training process involves training and evaluating two models, TransUNet and UNet, over multiple epochs to achieve optimal segmentation performance
- I have used separate AdamW optimizers for each model.
- Learning rates are set to 5×10^{-5} for TransUNet and 3×10^{-4} for UNet.
- The training loop iterates over a specified number of epochs (40 in this case). In each epoch, the models are trained on the training dataset and evaluated on the validation dataset
- During training, the models are set to training mode, and the optimizer gradients are reset before each batch (8 in this case).
- The forward pass computes model outputs, where the loss is calculated and afterward backpropagated for updating model parameters. It involves sending the input data through the network to receive predictions; these are compared with actual values to compute the loss
- The calculated loss will then be used to adjust the weights of the model in a process called backpropagation to improve the model's performance over time by minimizing the loss.

```python
# Instantiate the models
model1 = TransUNet(num_classes=1).to(device)
model2 = UNet(in_channels=3, num_classes=1).to(device)

# Define separate optimizers for each model
optimizer1 = optim.AdamW(model1.parameters(), lr=5e-5)
optimizer2 = optim.AdamW(model2.parameters(), lr=3e-4)

# Define the loss function
criterion = nn.BCEWithLogitsLoss()

# Training parameters
EPOCHS = 40
```

Figure 7: parameters

3.5.1 Evaluation

- The evaluation process assesses the performance of the trained models, TransUNet and UNet, on the validation dataset to ensure they generalize well to unseen data
- The evaluation is conducted at the end of each training epoch to monitor progress and make adjustments if necessary.
- **Matrics Calculation**
  - **Loss:** The same loss function **(BCEWithLogitsLoss)** used during training is applied to quantify the difference between predictions and actual masks.
  - **IoU (Intersection over Union):** Measures the overlap between predicted and ground truth masks, providing insight into the model's accuracy. 1 indicates perfect overlap, and 0 indicates no overlap
  - **Dice Coefficient:** Another overlap metric that is particularly sensitive to class imbalances, often used in medical image segmentation. It gives a better measure of accuracy when there are unequal class distributions
  - **Precision, Recall, and F1 Score:** Precision measures the proportion of true positive predictions, recall measures the proportion of actual positives correctly identified, and F1 score provides a balance between precision and recall
- Matrics helps us in understanding how well the model identifies the regions of interest versus the background, which is critical for applications like medical image segmentation where accuracy is paramount.

```python
# Define the dice_coefficient function
def dice_coefficient(outputs: torch.Tensor, labels: torch.Tensor, threshold=0.5):
    SMOOTH = 1e-6
    outputs = (outputs > threshold).float()
    labels = labels.float()
    intersection = (outputs * labels).sum((1, 2))
    dice = (2. * intersection + SMOOTH) / (outputs.sum((1, 2)) + labels.sum((1, 2)) + SMOOTH)
    return dice.mean()

# Define the iou_pytorch function
def iou_pytorch(outputs: torch.Tensor, labels: torch.Tensor, threshold=0.5):
    SMOOTH = 1e-6
    outputs = (outputs > threshold).int()
    labels = labels.int()
    intersection = (outputs & labels).float().sum((1, 2))
    union = (outputs | labels).float().sum((1, 2))
    iou = (intersection + SMOOTH) / (union + SMOOTH)
    return iou.mean()
```

```python
# Define the precision, recall, and f1_score functions
def precision_recall_f1(outputs: torch.Tensor, labels: torch.Tensor, threshold=0.5):
    SMOOTH = 1e-6
    outputs = (outputs > threshold).float()
    labels = labels.float()

    true_positives = (outputs * labels).sum((1, 2))
    predicted_positives = outputs.sum((1, 2))
    actual_positives = labels.sum((1, 2))

    precision = (true_positives + SMOOTH) / (predicted_positives + SMOOTH)
    recall = (true_positives + SMOOTH) / (actual_positives + SMOOTH)
    f1 = 2 * (precision * recall) / (precision + recall + SMOOTH)

    return precision.mean(), recall.mean(), f1.mean()
```

Figure 8: Matrics defining Source: 2

Confusion matrices play a key role in this process by providing detailed insights into the model's classification performance, identifying specific errors, and facilitating the calculation of important metrics like precision, recall, F1 score, IoU, and Dice coefficient

These metrics are vital for understanding the model's strengths and weaknesses, guiding improvements, and ensuring robust segmentation performance.

3.6 Results Analysis and Visualization

3.6.1 Result Analysis

- **Train Loss**
  - The training loss for both models (TransUNet and UNet) decreases significantly in the initial epochs, indicating that the models are learning to fit the training data effectively
  - As training progresses, the loss continues to decrease but at a slower rate, eventually stabilizing towards the later epochs
  - The overall trend shows a steady reduction in loss, which is a positive indication of effective learning

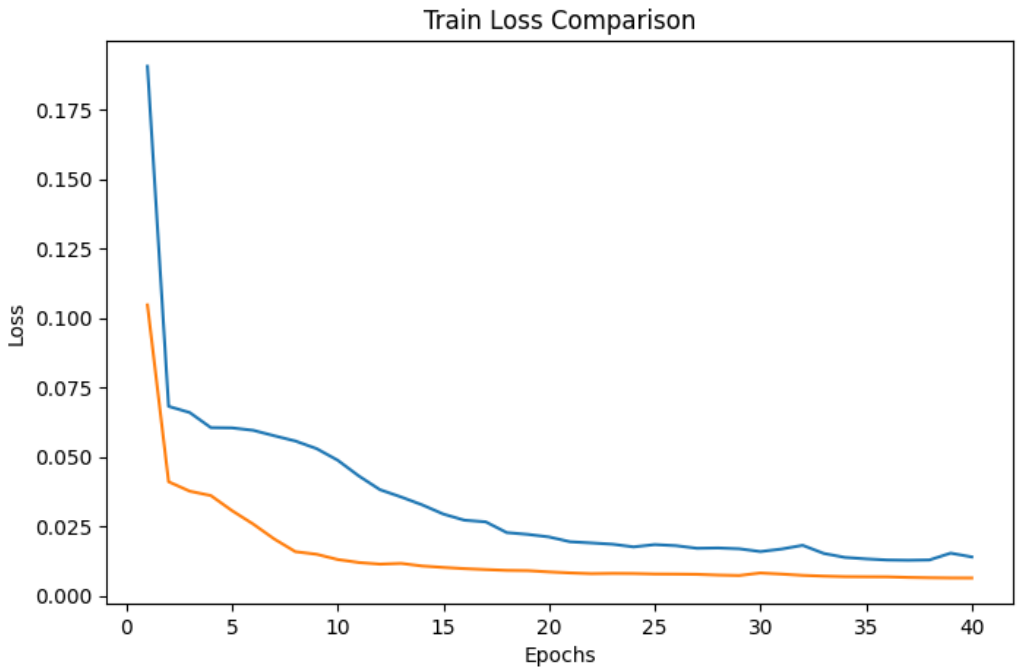

Figure 9: Train Loss (Blue refers to U-Net and orange refers to TransUNet)

- **Validation Loss**
  - The validation loss for both models also shows a decreasing trend over the epochs, demonstrating that the models are generalizing well to unseen data
  - There are some fluctuations in the validation loss, particularly in the middle epochs, which could be due to the models encountering more challenging samples in the validation set
  - The validation loss stabilizes towards the later epochs, with both models achieving relatively low loss values, indicating good generalization

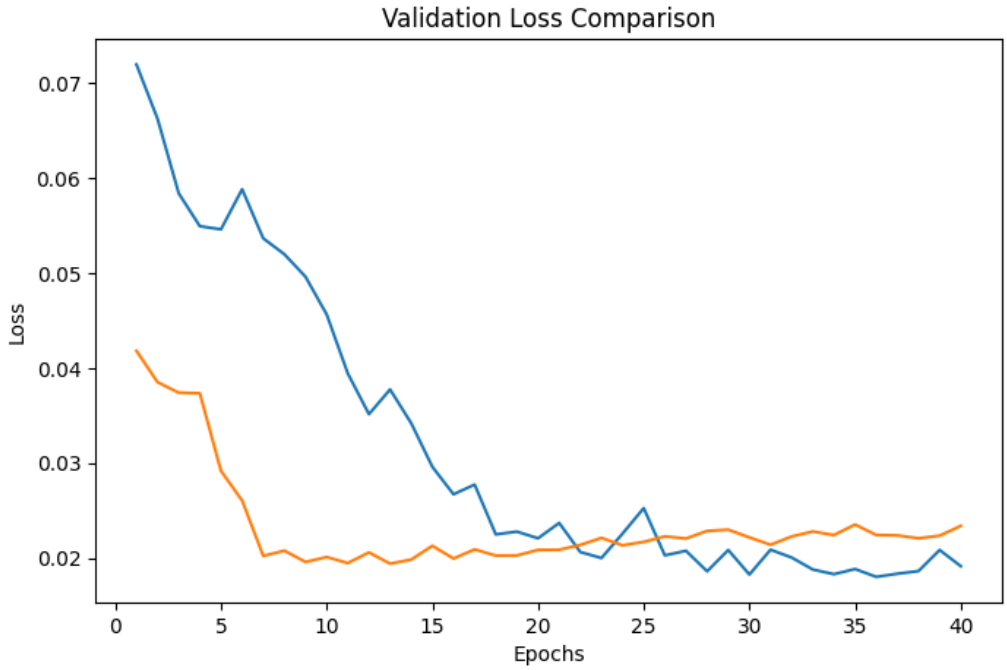

Figure 10: Val Loss (Blue refers to U-Net and orange refers to TransUNet)

- **Validation IoU**
  - Both models show an increasing trend in IoU over the epochs, indicating improved performance.
  - TransUNet (represented in orange) quickly achieves high IoU, reaching above 0.92 within the first 10 epochs and stabilizing around 0.92-0.93
  - UNet (represented in blue) shows a gradual increase in IoU, stabilizing around 0.90-0.91 towards the end of the training period

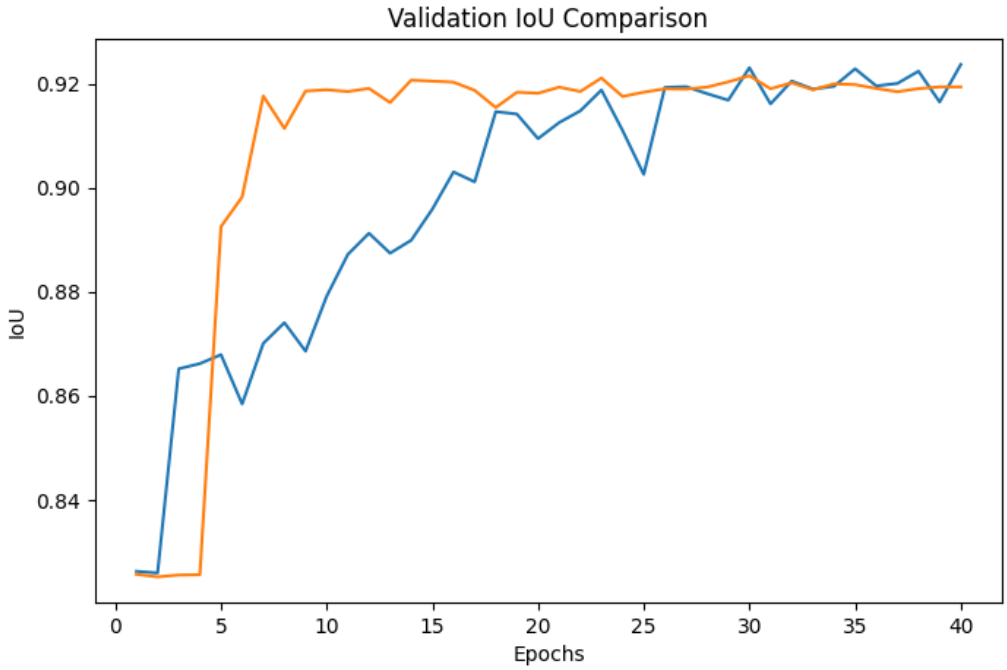

Figure 11: IoU Comparison (Blue refers to U-Net and orange refers to TransUNet)

- **Validation Precision**
  - Both models show high precision throughout the training period, with values oscillating between 0.96 and 1.0
  - TransUNet (in orange) maintains very high precision (>0.98) consistently, with minor fluctuations
  - UNet (in blue) shows more variability in precision, with occasional drops below 0.97 but generally staying above 0.97
  - The high precision of TransUNet indicates it makes fewer false positive errors, consistently predicting positive regions accurately

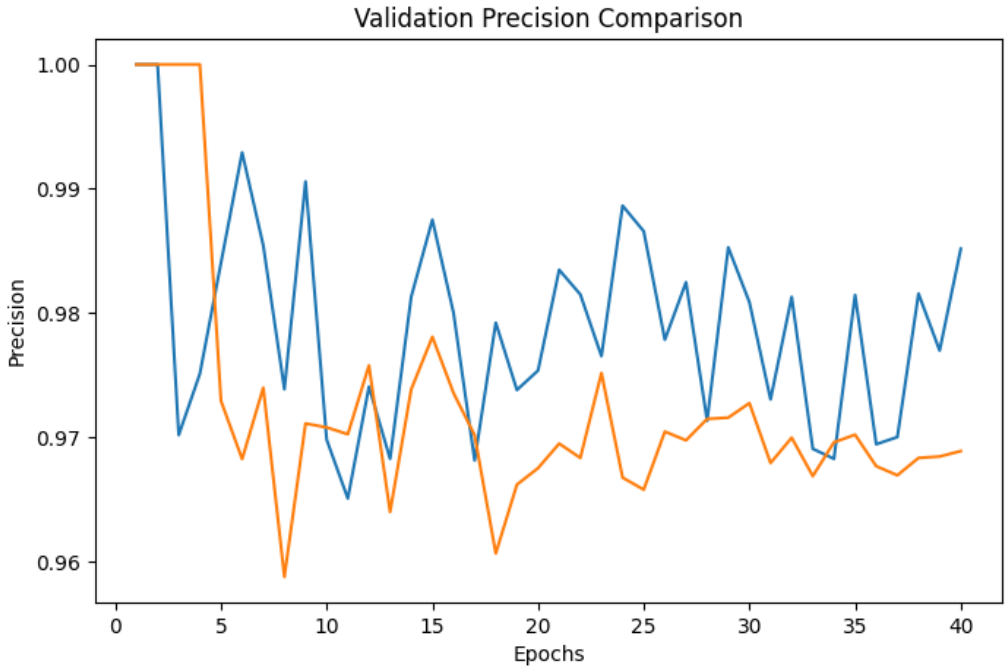

Figure 12: Precision Comparison (Blue refers to U-Net and orange refers to TransUNet)

- **Validation Recall**
  - Both models show an increasing trend in recall over the epochs, indicating improved ability to identify positive regions
  - TransUNet (in orange) achieves high recall quickly, stabilizing around 0.93-0.94
  - UNet (in blue) shows a gradual increase in recall, reaching around 0.91 towards the end of the training period

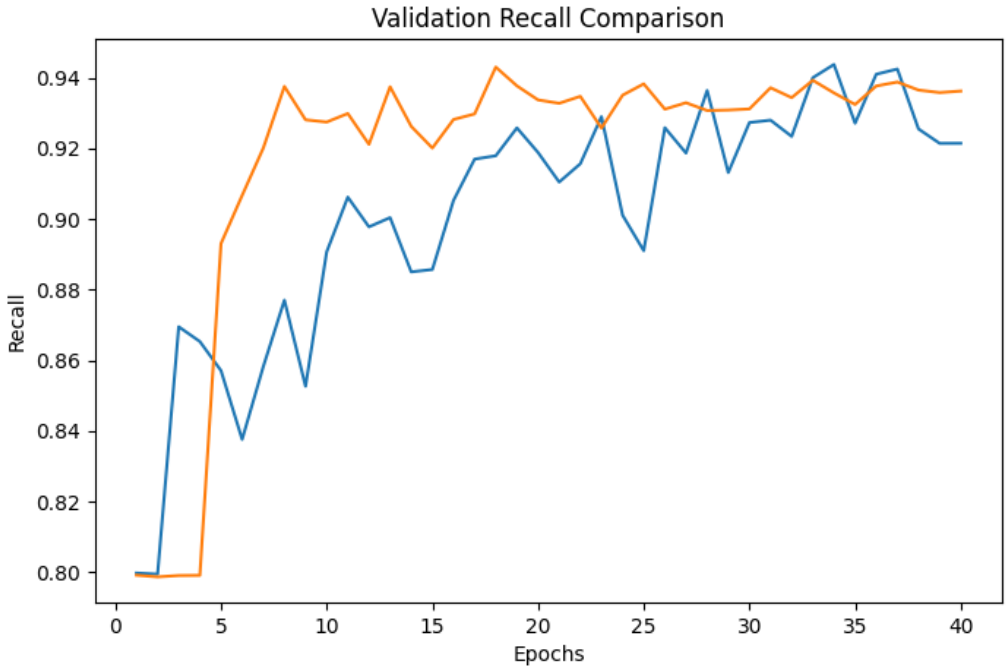

Figure 13: Recall Comparison (Blue refers to U-Net and orange refers to TransUNet)

- **Summary**
  - Shows rapid improvement and high performance across all metrics
  - Consistently achieves high IoU, precision, and recall, indicating strong segmentation capabilities with fewer errors
  - These metrics highlight the effectiveness of TransUNet in capturing both global and local features, leading to better segmentation performance compared to UNet
  - The high and consistent values across IoU, Precision, and Recall metrics underscore TransUNet's robustness and reliability in medical image segmentation tasks

3.6.2 Result Visualization

- **Result of U-Net**

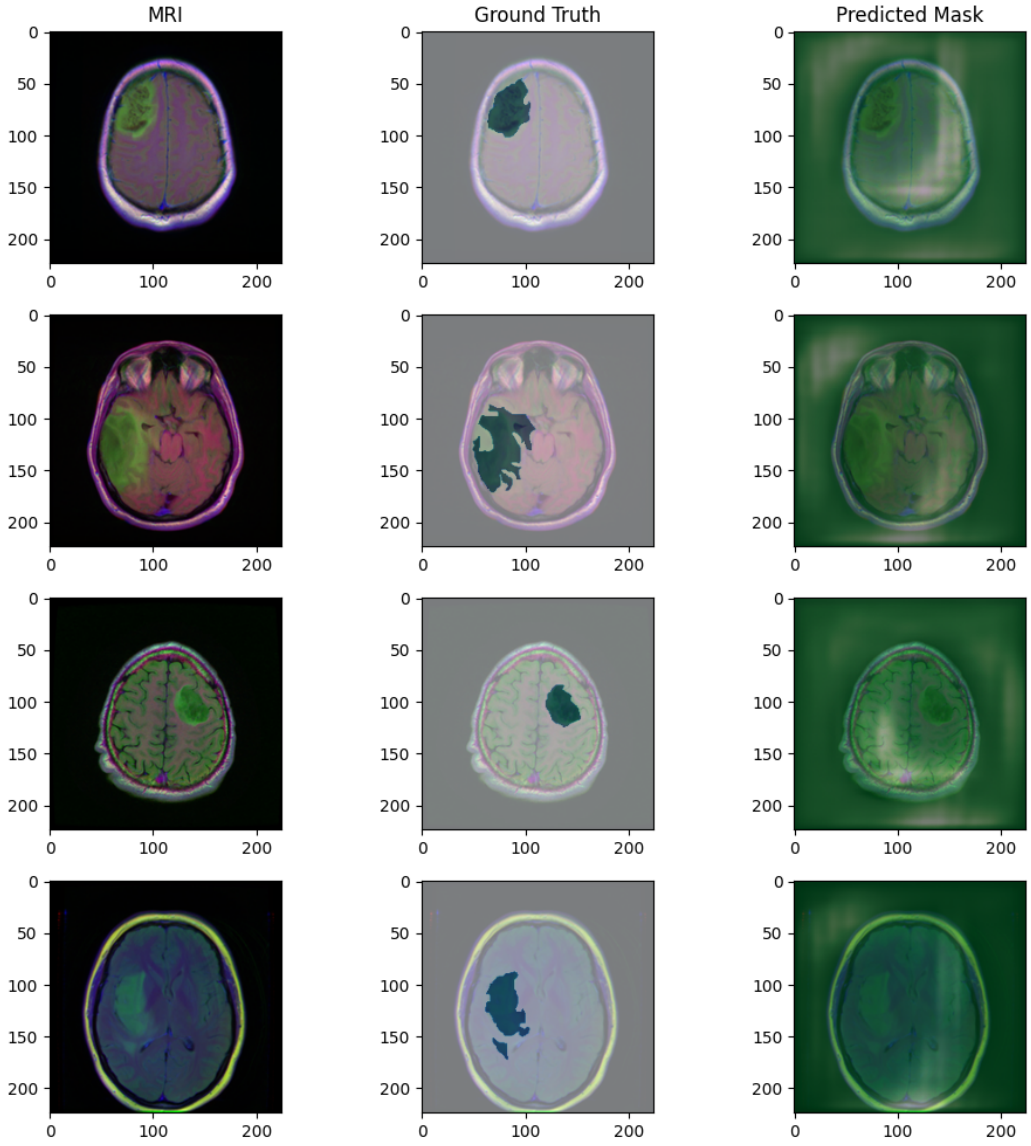

Figure 14: Unet Result

- **Result of TransUNet**

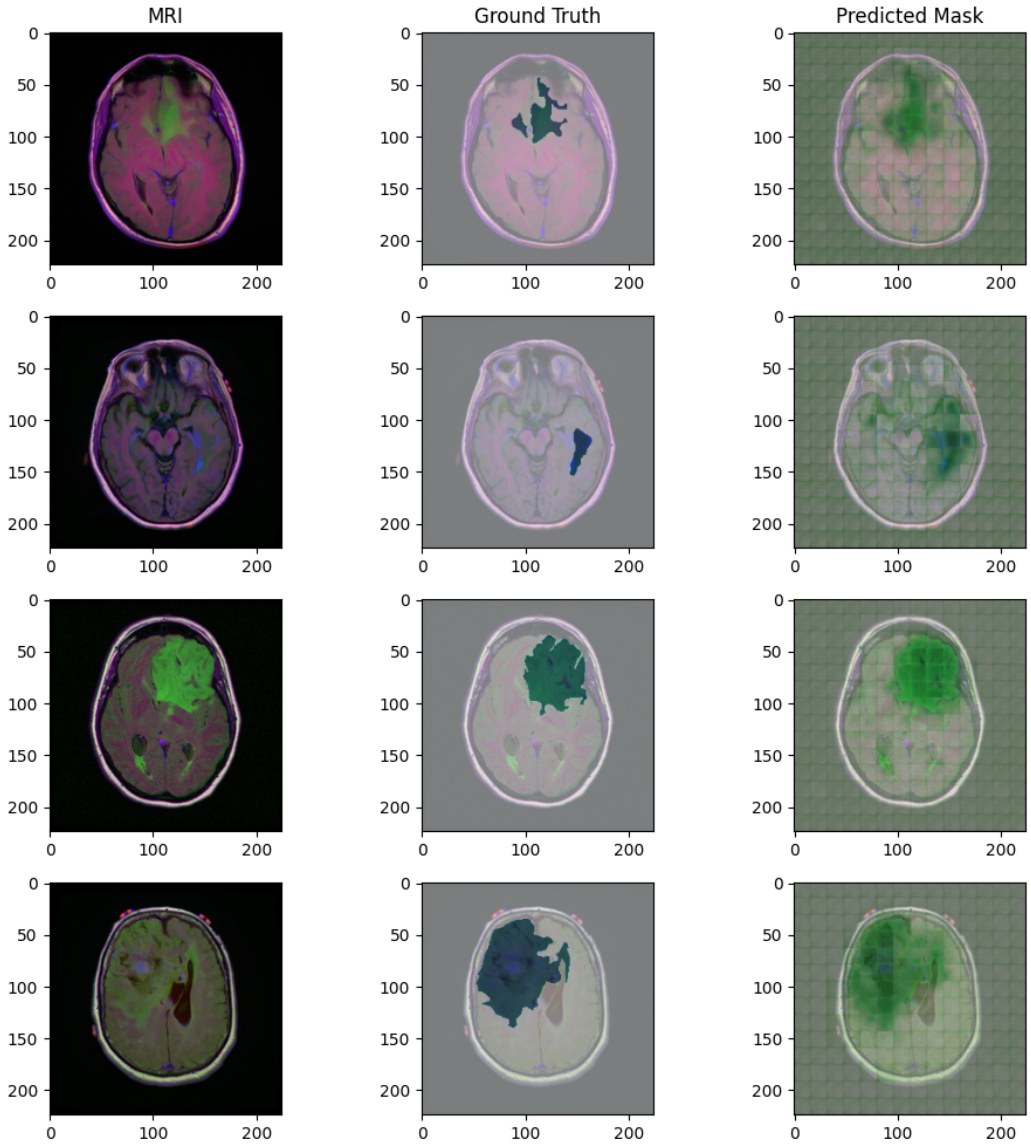

Figure 15: TransUnet result

Summary of both Model

- Visual comparisons show that both models produce accurate segmentation masks for many cases
- TransUNet's predictions generally show better alignment and detail compared to UNet, indicating its robustness in handling complex segmentation tasks
- TransUNet proves to be a more reliable and accurate model for medical image segmentation, providing superior performance across both quantitative metrics and qualitative visualizations

3.7 Conclusion and Future Aspects

3.7.1 Conclusion

In this project, I have successfully developed and evaluated two deep learning models, TransUNet and UNet, for the task of medical image segmentation. My primary goal was to accurately segment MRI images to identify regions of interest, such as tumors, which is critical for diagnosis and treatment planning

Overall, the TransUNet model outperformed the UNet model, demonstrating its potential as a powerful tool for medical image segmentation. The results emphasize the importance of integrating transformer-based architectures to enhance segmentation accuracy and robustness.

3.7.2 Future Aspects

- Explore synthetic data generation methods to further augment the dataset and address class imbalances
- Experiment with different transformer architectures and pre-trained models to evaluate their impact on segmentation quality
- Incorporate additional imaging modalities (e.g., CT scans, PET scans) to create a multi-modal segmentation framework
- Optimize the model for real-time segmentation applications, ensuring fast and efficient processing of high-resolution medical images
- Develop methods to enhance the interpretability of the model's predictions, providing insights into the decision-making process
- Implement explainable AI techniques to ensure the model's outputs are transparent and trustworthy for clinical use.

Visual results indicated that while both models produced accurate segmentations, TransUNet exhibited better alignment and detail, especially in complex regions

These future directions aim to improve model accuracy, robustness, and clinical applicability, ultimately enhancing patient care

REFERENCES

[1] Kaggle. *Dataset of MRI images and masks*. Available at: `https://www.kaggle.com/`.

[2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv preprint arXiv:1505.04597*. Available at: `https://arxiv.org/abs/1505.04597`.

[3] Github. *Implementation of U-Net model* . Available at: `https://github.com/uygarkurt/UNet-PyTorch/blob/main/unet.py`.

[4] Github. *Implementation of TransUNet model* . Available at: `https://github.com/mkara44/transunet_pytorch`.