FUSE MACHINES

COMPUTER VISION COHORT

BRAIN TUMOR LOCALIZATION AND SEGMENTATION

SUBMITTED BY:

MANOJ KHATRI

NIRAJAN BEKOJU

SUBMITTED TO:

FUSE MACHINES

December, 2023

Acknowledgments

We extend our heartfelt gratitude to all those who have been instrumental in the successful completion of our brain tumor detection and segmentation project. Your unwavering support and guidance have been invaluable to us, and we deeply appreciate your contributions.

We want to express a special note of thanks to the Teaching Assistants who have generously shared their expertise and provided assistance throughout the duration of the Fusemachines Deep Learning course. Your commitment to our learning experience has been remarkable. In particular, we would like to extend our sincere appreciation to our instructors for theexceptional guidance. His insights and patient responses to our queries have significantly shaped the outcome of our project.

To all the other TAs who have generously shared their knowledge, we are truly grateful. Your dedication to our growth and understanding has been pivotal.

We also acknowledge the effort and commitment of our instructors in creating an environment conducive to learning. Your support has motivated us to explore the subject matter deeply, and it has been instrumental in the success of our project.

Abstract

In the realm of medical image analysis, accurate brain tumor detection and segmentation are critical tasks with far-reaching implications for patient care and treatment. This project presents a comprehensive exploration of brain tumor detection and segmentation through the implementation of two distinct deep learning approaches: the U-Net architecture and U-Net with ResNet as a backbone.

The U-Net architecture, known for its effectiveness in semantic segmentation tasks, forms the cornerstone of our initial approach. Leveraging its encoding and decoding pathways, we endeavor to accurately delineate brain tumor regions from magnetic resonance imaging (MRI) scans. This approach demonstrates our foundational understanding of deep learning techniques for medical image analysis.

Building upon this foundation, we extend our exploration to U-Net with ResNet as a backbone. This hybrid approach combines the powerful feature extraction capabilities of ResNet with the precise localization strengths of U-Net. Through this amalgamation, we aim to achieve enhanced accuracy and robustness in brain tumor detection and segmentation.

Our project encompasses not only the implementation of these approaches but also their comparative analysis in terms of performance metrics, computational efficiency, and the quality of segmented regions. The dataset used comprises a diverse set of brain MRI scans, reflecting the diversity of real-world scenarios encountered in clinical practice.

The results obtained through our experimentation shed light on the strengths and limitations of both approaches. We discuss their respective advantages and challenges, contributing to a comprehensive understanding of deep learning techniques in medical image analysis. The insights garnered from this project hold promise for the refinement of brain tumor diagnosis and treatment planning, ultimately improving patient outcomes.

Keywords: Brain tumor detection, Segmentation, U-Net, ResNet, Deep learning, Medical image analysis, Magnetic resonance imaging (MRI).

Contents

Acknowledgements **2**

Abstract **2**

Contents **4**

List of Figures **5**

1 Introduction **1**

1.1 Background 1

1.2 Problem statements 1

1.3 Objectives 1

2 Literature Review **2**

3 Methodology **3**

3.1 Datasets 3

3.2 Preprocessing and Augmentation 5

3.2.1 Transforms 5

3.2.2 Strong transforms 6

3.3 Architectures 7

3.3.1 UNet Architecture Implementation 8

3.3.2 ResNeXtUNet Architecture Implementation 10

3.4 Training 11

3.5 Performance Evaluation 12

3.5.1 Testing 14

4 Inference and Deployment **17**

4.1 Inference 17

4.2 Deployment with Gradio 18

4.2.1 Benefits and Considerations 18

5 Conclusion **20**

List of Figures

3.1 Initial state of the dataset 3
3.2 Distribution of diagnosis in the brain 4
3.3 Some samples from the dataset 5
3.4 Images after augmentation 7
3.5 Unet architecture 8
3.6 RestNextUNet Architecture 10
3.7 Dice coefficient 12
3.8 History of Dice Coefficient for UNet 13
3.9 History of Dice Coefficient for ResUNet 13
3.10 Dice coefficient with each epoch 14
3.11 Example of prediction on testset1 15
3.12 Example of prediction on testset2 15
3.13 Example of prediction on testset3 16

4.1 A full fledged example of inference 17
4.2 Deployment in HuggingFace using gradio 18

1. Introduction

1.1 Background

Medical image analysis has witnessed remarkable advancements in recent years, driven primarily by the integration of deep learning techniques. Among the myriad applications in this domain, brain tumor detection and segmentation have emerged as pivotal tasks with significant implications for patient care and treatment planning. Accurate delineation of tumor regions from magnetic resonance imaging (MRI) scans can facilitate early diagnosis, precise monitoring of tumor growth, and informed surgical interventions.

1.2 Problem statements

The accurate detection and segmentation of brain tumors from magnetic resonance imaging (MRI) scans are essential tasks in the field of medical image analysis. However, the conventional methods for manual segmentation by radiologists are time-consuming, subject to inter-observer variability, and may lack the required precision for critical medical decision-making. This underscores the need for automated and robust techniques that can consistently and accurately identify tumor regions, enabling early diagnosis, treatment planning, and monitoring of tumor progression.

To address these challenges, our project focuses on the development and evaluation of deep learning approaches for brain tumor detection and segmentation. Specifically, we aim to investigate the efficacy of two distinct architectures: the U-Net and U-Net with ResNet as a backbone. These architectures hold the potential to overcome the limitations of manual segmentation by automating the process, reducing human error, and improving the efficiency of clinical workflows.

1.3 Objectives

1. Implement a mechanism for users to upload their MRI images directly through the web interface, streamlining the process of data submission.
2. Configure the backend to process the uploaded images using the pre-trained U-Net and U-Net with ResNet models, generating segmented brain tumor regions.
3. Merge the segmented brain tumor regions with the original MRI images to create overlapped images, effectively showcasing the areas of the brain affected by tumors.

2. Literature Review

The Literature Review covers the existing research and work in this area. It involved reading papers, searching through internet and acquring related knowledge.

1. **U-Net Architecture:**

   The U-Net architecture proposed by Ronneberger et al. (2015) has emerged as a seminal approach for semantic segmentation in medical imaging. With its unique encoding and decoding pathways, U-Net has been successfully applied to brain tumor detection and segmentation tasks. Studies like **?** and **?** have demonstrated the effectiveness of U-Net in accurately identifying tumor regions within MRI scans.

2. **Deep Residual Networks (ResNet):**

   Deep Residual Networks (ResNet), introduced by He et al. (2015), have revolutionized deep learning architectures by addressing the vanishing gradient problem. In the context of medical image analysis, studies such as **?** have explored the integration of ResNet as a backbone for enhancing feature extraction capabilities in brain tumor segmentation tasks.

3. **Data Augmentation and Preprocessing:**

   Data augmentation techniques, including rotation, scaling, and flipping, have been employed to augment the training dataset and improve model generalization. Preprocessing methods like skull stripping and intensity normalization have been highlighted in studies such as **?** to enhance the quality of input images for accurate segmentation.

4. **Transfer Learning and Pre-Trained Models:**

   Transfer learning, utilizing pre-trained models on large datasets, has proven effective in brain tumor segmentation. Research like **?** has shown that fine-tuning pre-trained models on a specific dataset can significantly improve segmentation performance.

5. **Evaluation Metrics:**

   Assessing the performance of segmentation models requires appropriate evaluation metrics. Commonly used metrics include Dice coefficient, Intersection over Union (IoU), and sensitivity. Studies like **?** have emphasized the importance of selecting suitable metrics for accurate performance assessment.

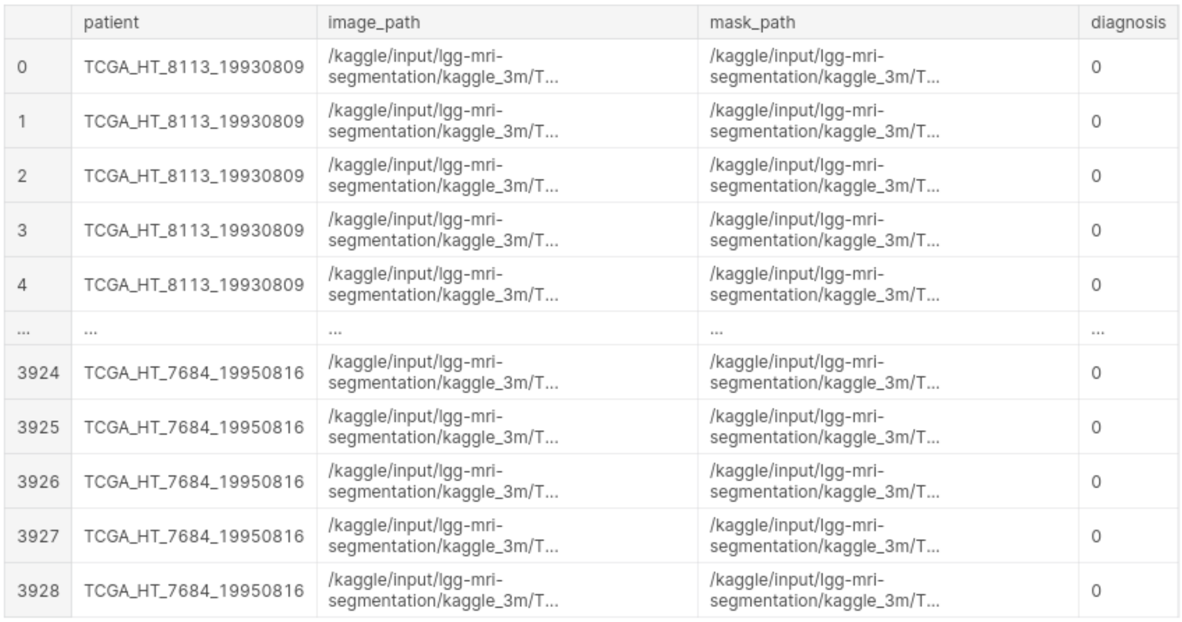
3. Methodology

We adopted a multi-step methodology, encompassing data augmentation for diverse training sets, pre-processing including skull stripping and intensity normalization, training U-Net and U-Net with ResNet using diverse MRI scans, and finally evaluating model performance using metrics like Dice coefficient and sensitivity. This approach aimed to enhance robustness, mitigate overfitting, and achieve accurate brain tumor segmentation in medical images.

3.1 Datasets

This dataset contains brain MR images together with manual FLAIR abnormality segmentation masks. The images were obtained from The Cancer Imaging Archive (TCIA). They 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.

The brain tumor imagery dataset was first used in Mateusz Buda, AshirbaniSaha, Maciej A. Mazurowski ”Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm.” Computers in Biology and Medicine, 2019.



	patient	image_path	mask_path	diagnosis
0	TCGA_HT_8113_19930809	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
1	TCGA_HT_8113_19930809	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
2	TCGA_HT_8113_19930809	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
3	TCGA_HT_8113_19930809	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
4	TCGA_HT_8113_19930809	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
...	...	...	...	...
3924	TCGA_HT_7684_19950816	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
3925	TCGA_HT_7684_19950816	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
3926	TCGA_HT_7684_19950816	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
3927	TCGA_HT_7684_19950816	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
3928	TCGA_HT_7684_19950816	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0

Figure 3.1: Initial state of the dataset

Source: https://wiki.cancerimagingarchive.net/display/Public/TCGA-LGG

In the initial stages of our project, the dataset consisted primarily of essential information such as patient IDs, file paths to the corresponding MRI images, associated masks, and diagnostic labels. However, to facilitate effective deep learning model training for brain tumor segmentation, a structured dataset with paired image-mask data was required. This necessitated a comprehensive data preprocessing and augmentation pipeline that leveraged tools like OpenCV and Pandas.

```
ax = df.diagnosis.value_counts().plot(
    kind='bar',
    stacked=True,
    figsize=(10, 6),
    color=["green", "blue"]
)
```

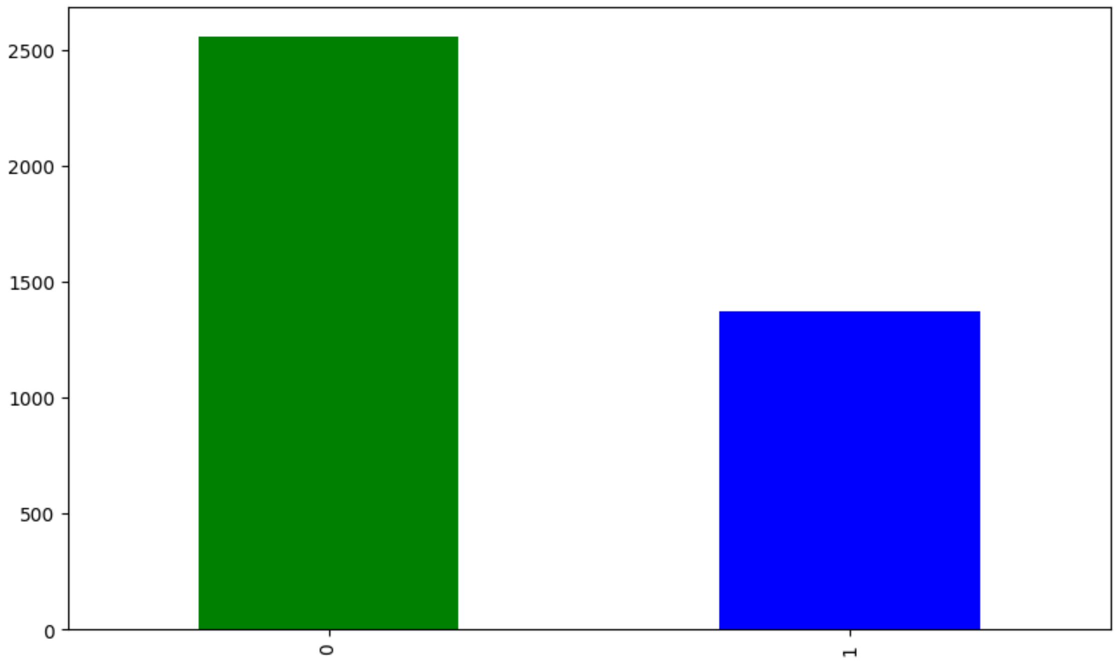


Figure 3.2: Distribution of diagnosis in the brain

Using OpenCV, we programmatically loaded the MRI images and their corresponding masks. The masks, which delineate tumor regions within the images, were crucial for training the segmentation models. By applying appropriate image processing techniques, we ensured that the masks aligned accurately with the original MRI images.

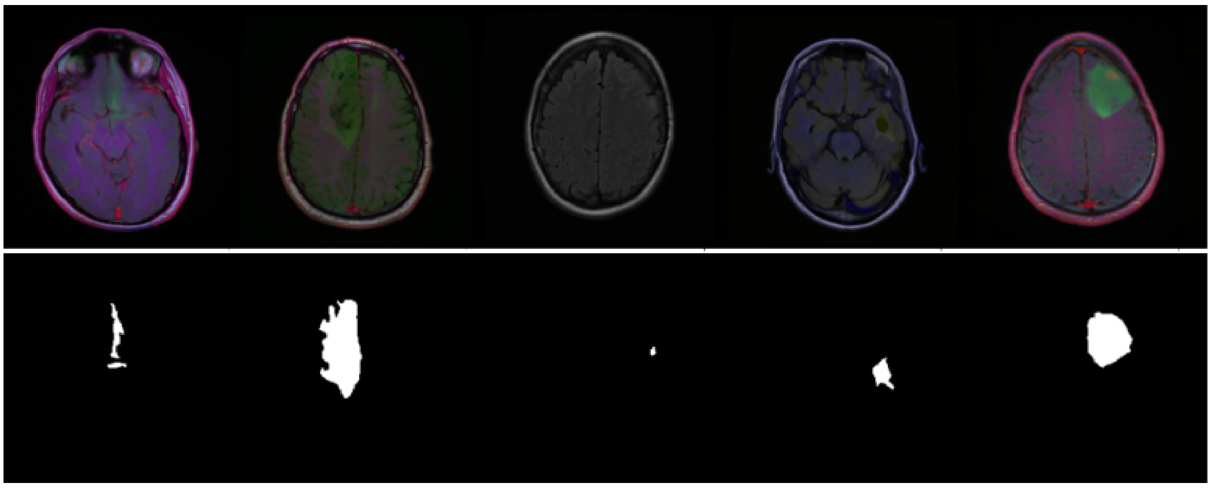


Figure 3.3: Some samples from the dataset

3.2 Preprocessing and Augmentation

In the context of training deep learning models for computer vision tasks, effective image preprocessing and augmentation play a pivotal role in enhancing model performance, generalization, and robustness. This report delves into the details of a comprehensive preprocessing and augmentation pipeline, exemplified by the provided code snippet.

The primary goal of image preprocessing and augmentation is to enhance the learning capabilities of neural networks by providing them with a diverse and representative dataset. Preprocessing techniques standardize the input data, making it compatible with the network architecture, while augmentation techniques artificially increase the dataset size and introduce variations to reduce overfitting.

We tried two sets of image transformations: transforms and strong-transforms. Each set serves a distinct purpose in the augmentation pipeline.

3.2.1 Transforms

- Resize: Images are resized to a predetermined dimension (PATCHSIZE x PATCHSIZE) using bilinear interpolation. This ensures uniformity in input dimensions for the neural network.
- HorizontalFlip and VerticalFlip: Images are horizontally and vertically flipped with a probability of 0.5 each. This expands the dataset while maintaining object orientation invariance.
- RandomRotate90: Images undergo a random rotation by 90, 180, or 270 degrees with a probability of 0.5. This enhances the model's capacity to handle varied object orientations.

- Transpose: Images' rows and columns are transposed with a probability of 0.5, introducing further diversity to the dataset.
- ShiftScaleRotate: A combination of shift, scale, and rotation transformations are applied. This introduces controlled spatial variations, aiding the model in learning position, scale, and orientation invariance.
- Normalize: Pixel values are normalized to facilitate stable and efficient model training. Normalization involves subtracting the mean and dividing by the standard deviation of pixel values across the dataset.
- ToTensorV2: Images are converted from NumPy arrays to PyTorch tensor format, enabling seamless integration with the PyTorch framework.

3.2.2 Strong transforms

- RandomResizedCrop: Images undergo random resizing followed by cropping to the desired PATCHSIZE x PATCHSIZE. This exposes the model to diverse object scales and positions.
- RandomBrightnessContrast: Random adjustments to brightness and contrast are applied. This introduces variations in lighting conditions, enhancing the model's adaptability.
- RandomGamma: Random gamma adjustments are applied, altering overall image intensity.
- IAAEmboss: An embossed effect is added to images, emphasizing edges and fine details, aiding the model in learning intricate patterns.
- Blur: Random blurring is applied, simulating motion blur or defocus scenarios.
- OneOf (ElasticTransform, GridDistortion, OpticalDistortion): Complex deformations like elastic transform, grid distortion, and optical distortion are introduced with a probability of 0.8. These simulate non-linear deformations in the images.

A meticulously designed image preprocessing and augmentation pipeline empowers deep learning models to better learn and generalize from the available data. By carefully controlling variations and introducing realistic perturbations, the model becomes more adept at recognizing essential features while being less sensitive to irrelevant fluctuations. This contributes to improved performance and robustness across a wide range of applications.

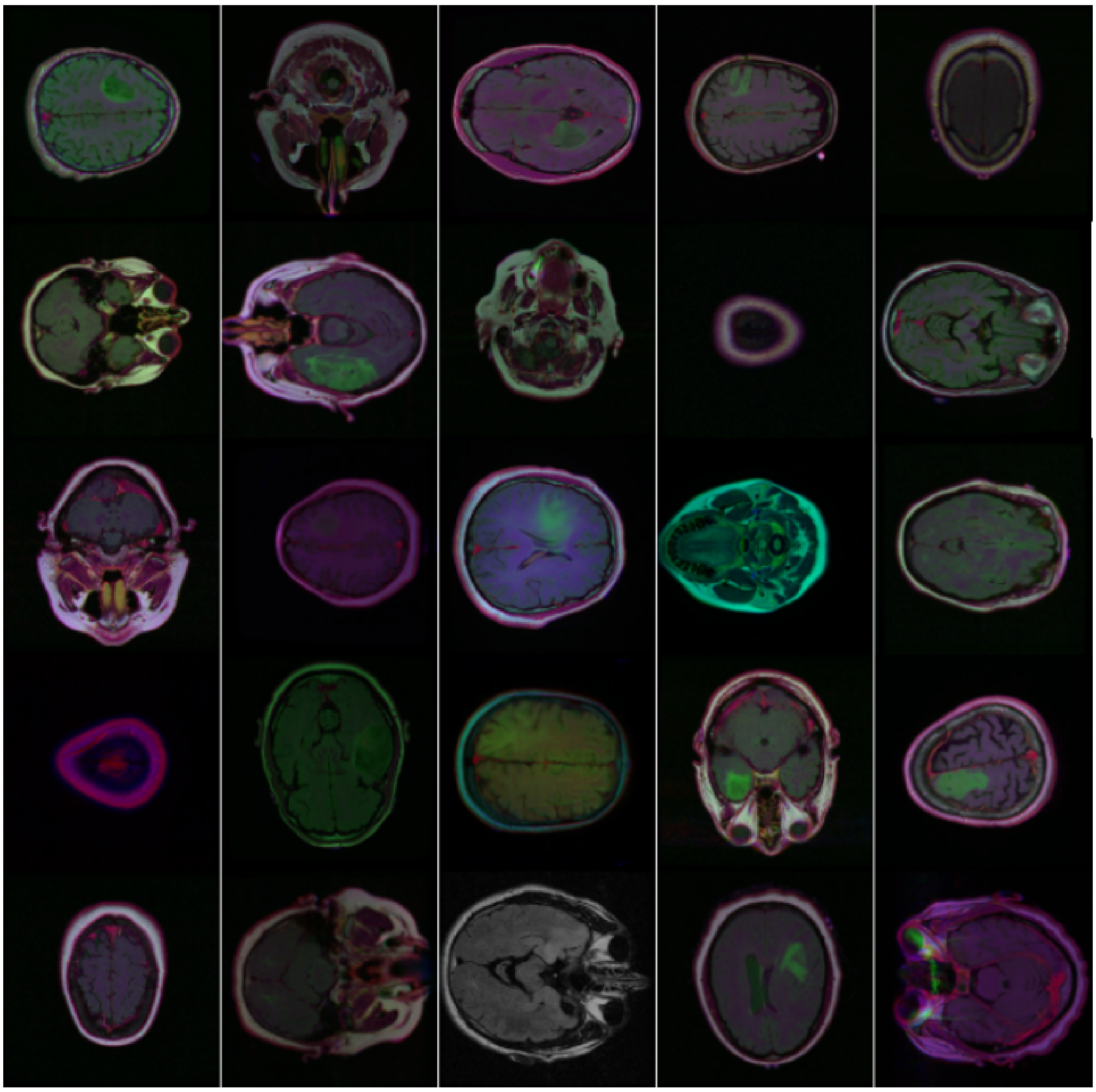


Figure 3.4: Images after augmentation

3.3 Architectures

We initiated our exploration by implementing the UNet architecture, a renowned choice for image segmentation tasks due to its encoder-decoder design. The encoder extracts hierarchical features from input images, while the decoder generates segmentation maps with precise spatial information. Our UNet architecture comprised contracting and expansive pathways, facilitating the integration of contextual information and precise localization.

Although our initial results with UNet were promising, we encountered limitations when dealing with complex object shapes, texture variations, and small objects. This led us to investigate alternative architectures to further refine our model's performance.

Recognizing the power of deep residual networks (ResNet) in capturing intricate features, we decided to augment our approach by integrating ResNet as the backbone architecture. ResNet's inherent ability to mitigate the vanishing gradient problem and its impressive performance in image classification made it an appealing candidate for our semantic segmentation task.

By employing ResNet, we aimed to leverage its learned feature representations to enhance our model's understanding of object boundaries and intricate details. We adapted the ResNet architecture to accommodate the specific demands of semantic segmentation, modifying the final classification layer to generate pixel-wise predictions instead of class labels.

3.3.1 UNet Architecture Implementation

The UNet architecture is a widely used neural network design for semantic segmentation tasks, especially in medical imaging and computer vision domains. It is known for its ability to capture detailed contextual information while preserving spatial resolution. In this subsection, we break down the implementation of the UNet model.

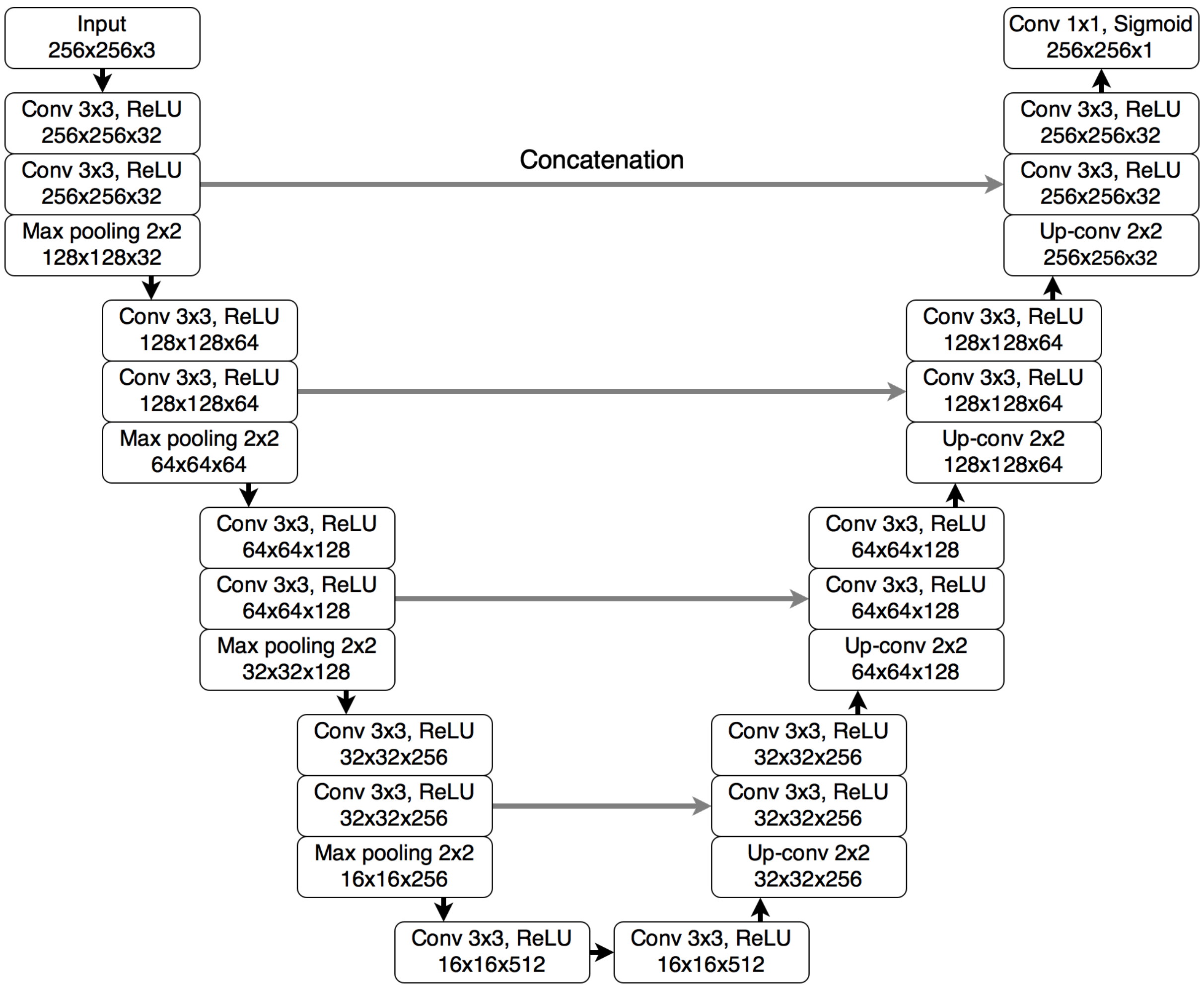


Figure 3.5: Unet architecture

Model Components

The UNet model is composed of several key components:

- **Encoder (Downsampling Path):** This section includes a series of convolutional layers, each followed by batch normalization and a ReLU activation function. The purpose of the encoder is to progressively reduce the spatial dimensions of the input while increasing the number of feature channels. This helps extract hierarchical features from the input image.
- **Max Pooling Layer:** After each set of convolutional layers in the encoder, a max pooling operation is applied to reduce the spatial dimensions further. This operation helps in capturing and preserving important features.
- **Decoder (Upsampling Path):** The decoder consists of upsampled feature maps using bilinear interpolation. It also involves concatenating feature maps from the corresponding encoder path to provide localized information. The decoder helps to recover the spatial resolution lost during the downsampling process.
- **Final Convolution Layer:** At the end of the architecture, a final convolutional layer with a kernel size of 1 is applied to produce pixel-wise predictions. This layer aims to generate class probabilities for each pixel in the input image.

Forward Pass

The `forward` method of the UNet class defines the data flow through the architecture:

1. **Encoding Path:** The input image is passed through a series of convolutional layers, followed by max pooling. Each encoder block captures progressively higher-level features and reduces spatial dimensions.
2. **Decoding Path:** The upsampled feature maps are concatenated with the corresponding feature maps from the encoder path. This concatenation helps to recover spatial information and provides context to the model. The decoder blocks consist of convolutional layers.
3. **Final Prediction:** The output of the last decoder block is passed through a convolutional layer with a kernel size of 1. The sigmoid activation function is applied to produce pixel-wise predictions, which represent class probabilities for each pixel.

The UNet architecture effectively combines both contracting and expansive paths, allowing it to capture intricate details while maintaining spatial information. This property makes it suitable for various image segmentation tasks.

The UNet architecture serves as a foundation for semantic segmentation tasks due to its ability to capture fine details and spatial context. It has been widely adopted in both research and practical applications, demonstrating its effectiveness in various domains.

3.3.2 ResNeXtUNet Architecture Implementation

The 'ResNeXtUNet' is a novel neural network architecture that combines the strengths of the ResNeXt feature extractor with the U-Net segmentation framework. This hybrid design allows the model to leverage both feature richness and spatial context for accurate semantic segmentation. In this subsection, we delve into the details of the implementation of the 'ResNeXtUNet' model.

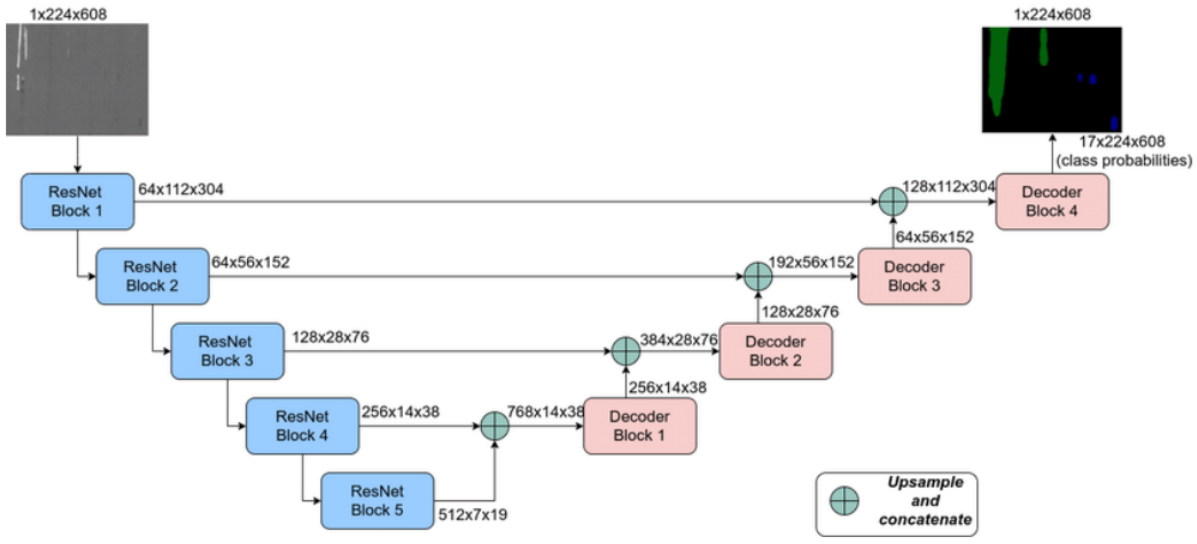


Figure 3.6: RestNextUNet Architecture

Custom Building Blocks

The architecture utilizes several custom building blocks that contribute to its overall structure:

- **ConvRelu:** This class encapsulates a convolutional layer followed by a ReLU activation function. It is used to create a simple yet effective building block for feature extraction.
- **DecoderBlock:** A DecoderBlock is responsible for upscaling feature maps using transposed convolutions while maintaining feature channel consistency. It involves a sequence of operations including a 1x1 convolution, a transposed convolution, and another 1x1 convolution.

Model Architecture

The 'ResNeXtUNet' architecture is constructed as follows:

- **Encoder:** The ResNeXt architecture serves as the backbone feature extractor. It is divided into multiple layers (`encoder0` to `encoder4`) corresponding to different stages of feature extraction. Each layer captures increasingly abstract features from the input image.
- **Decoder:** The architecture employs a series of `DecoderBlocks` (`decoder4` to `decoder1`) for the upsampling process. Each decoder block takes high-level feature maps from the encoder path and integrates them with upsampled feature maps. This combination helps recover spatial information and refines segmentation predictions.
- **Final Classifier:** The last part of the model consists of a 3x3 convolutional layer (`last_conv0`) followed by another 3x3 convolutional layer (`last_conv1`). The final layer produces pixel-wise class probabilities.

Forward Pass

The forward pass of the ‘ResNeXtUNet‘ involves several steps:

1. **Encoder Pass:** The input image is passed through the ResNeXt-based encoder layers, capturing hierarchical features at different scales.
2. **Decoder Pass:** The decoder blocks progressively upscale the feature maps. Additionally, the feature maps from the encoder path are merged with the upsampled feature maps to enhance localization.
3. **Final Prediction:** After the decoding process, a sequence of convolutional layers refines the features, followed by a sigmoid activation function to produce class probabilities for each pixel.

The ‘ResNeXtUNet‘ architecture showcases the synergy between ResNeXt’s powerful feature extraction capabilities and U-Net’s spatial context preservation. By integrating these components, the model achieves accurate and detailed semantic segmentation results across various domains.

3.4 Training

In this section, we discuss the training process of the two architectures, namely the `UNet` and `ResNeXtUNet`. We also present the performance evaluation of the trained models using the Dice coefficient metric.

Both the `UNet` and `ResNeXtUNet` architectures were trained using a similar pipeline:

1. **Dataset Preprocessing:** The dataset was preprocessed using various augmentation techniques such as resizing, flipping, rotation, and intensity adjustments. These augmenta-

tions aided in increasing the model's generalization and robustness to different variations in the input images.

2. **Loss Function:** The models were trained using combination of BCE and DICE loss. This loss is suitable for semantic segmentation tasks, encouraging the model to produce pixel-wise class probabilities that align with the ground truth labels.

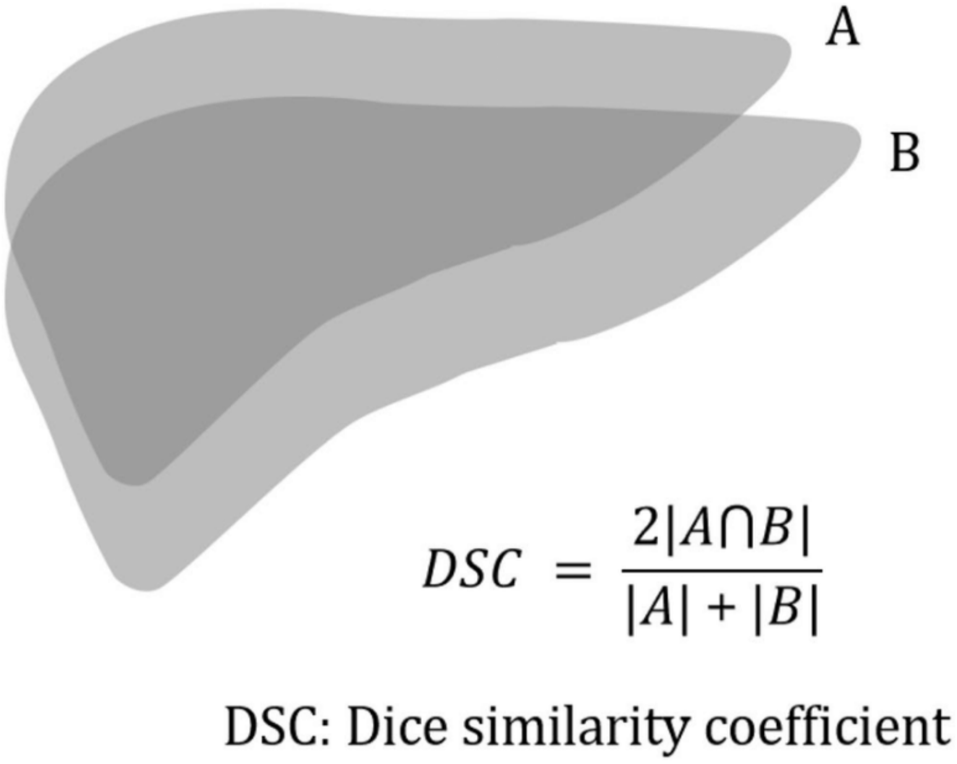

A
B
$DSC = \frac{2|A \cap B|}{|A| + |B|}$
DSC: Dice similarity coefficient

Figure 3.7: Dice coefficient

3. **Optimizer:** The training process employed an optimizer, like Adam or SGD, to update the model's weights based on the computed gradients of the loss function with respect to the parameters.

4. **Epochs and Batch Size:** The models were trained over a fixed number of epochs, and mini-batches of data were used during each iteration. The batch size was chosen based on memory constraints and computational efficiency.

3.5 Performance Evaluation

To assess the performance of the trained models, the Dice coefficient was employed as the evaluation metric. The Dice coefficient measures the overlap between the predicted segmentation and the ground truth, providing insights into the model's ability to capture relevant object boundaries and shapes.

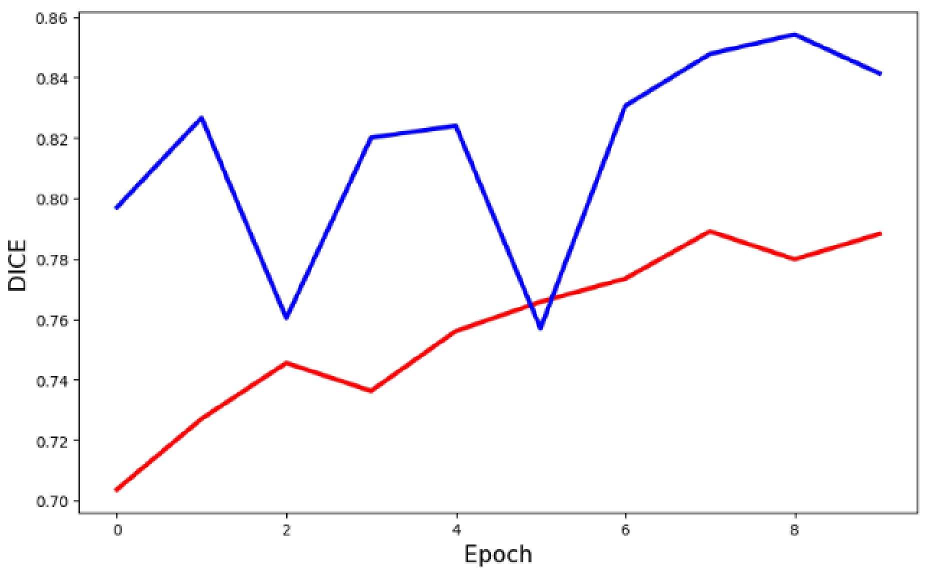


Figure 3.8: History of Dice Coefficient for UNet

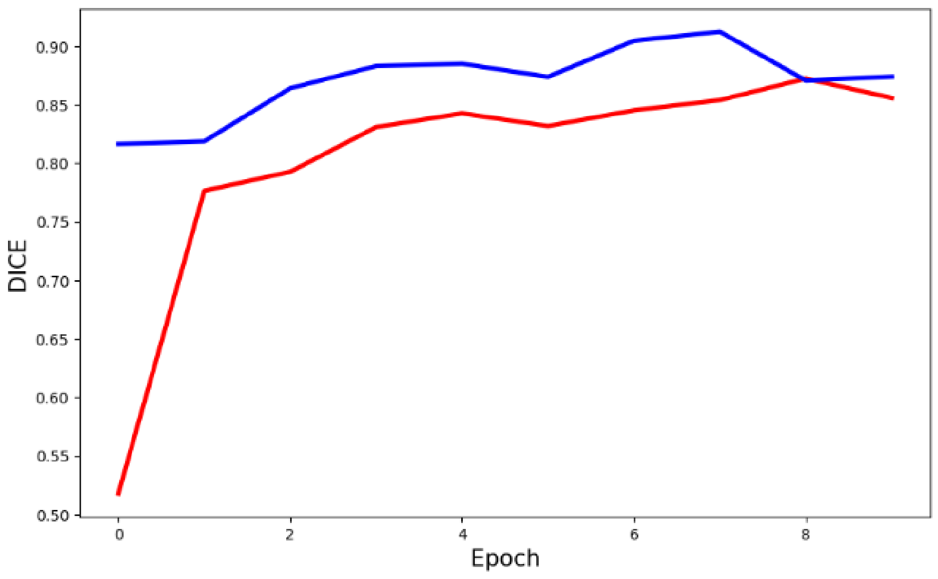
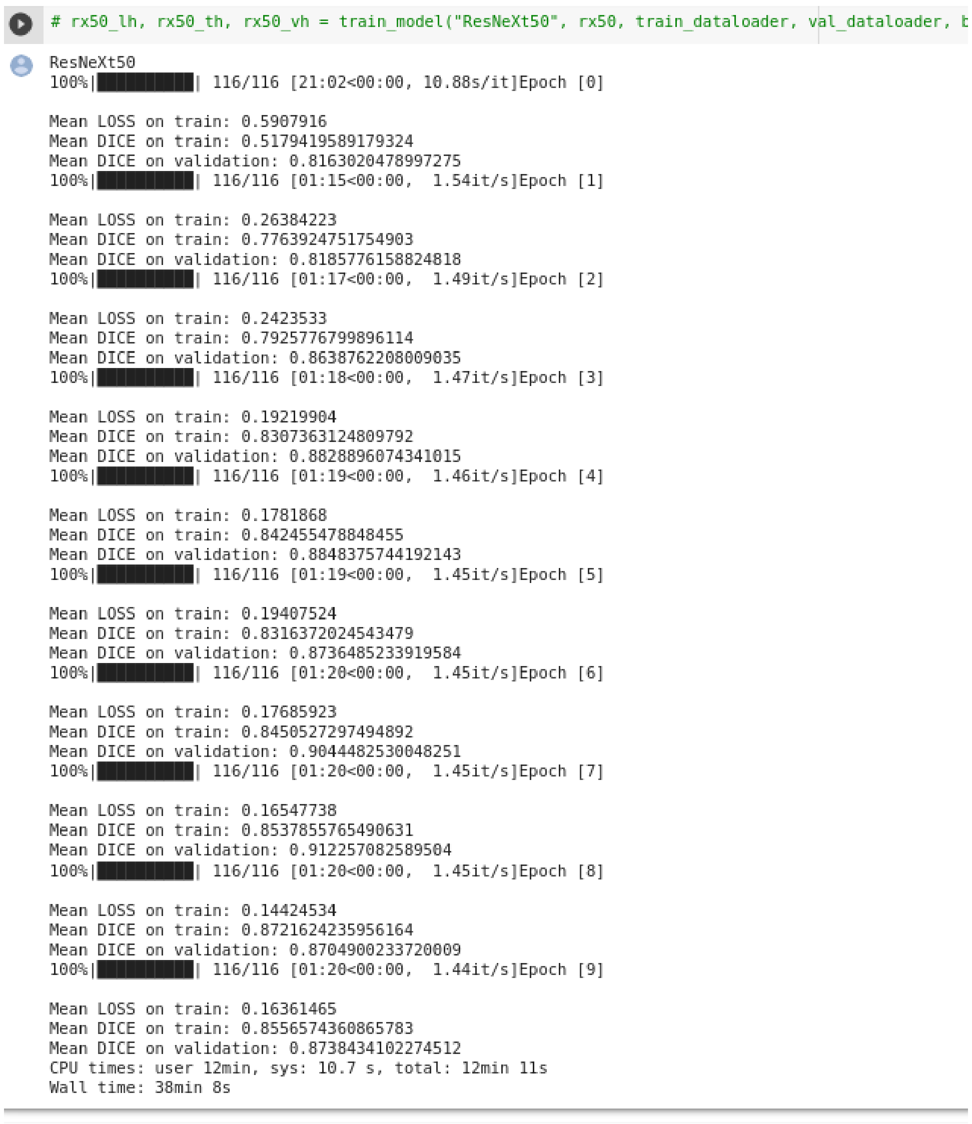


Figure 3.9: History of Dice Coefficient for ResUNet

Figure 4.1 illustrates the history of the Dice coefficient during the training process. The x-axis represents the training epochs, while the y-axis indicates the Dice coefficient values. The plot showcases the model's progression in terms of segmentation accuracy over training epochs.

The training process of both architectures resulted in models capable of producing meaningful segmentations. The Dice coefficient plot indicates the convergence and improvement in segmentation performance.



```
# rx50_lh, rx50_th, rx50_vh = train_model("ResNeXt50", rx50, train_dataloader, val_dataloader, b
```

```
ResNeXt50
100%|██████████| 116/116 [21:02<00:00, 10.88s/it]Epoch [0]

Mean LOSS on train: 0.5907916
Mean DICE on train: 0.5179419589179324
Mean DICE on validation: 0.8163020478997275
100%|██████████| 116/116 [01:15<00:00,  1.54it/s]Epoch [1]

Mean LOSS on train: 0.26384223
Mean DICE on train: 0.7763924751754903
Mean DICE on validation: 0.8185776158824818
100%|██████████| 116/116 [01:17<00:00,  1.49it/s]Epoch [2]

Mean LOSS on train: 0.2423533
Mean DICE on train: 0.7925776799896114
Mean DICE on validation: 0.8638762208009035
100%|██████████| 116/116 [01:18<00:00,  1.47it/s]Epoch [3]

Mean LOSS on train: 0.19219904
Mean DICE on train: 0.8307363124809792
Mean DICE on validation: 0.8828896074341015
100%|██████████| 116/116 [01:19<00:00,  1.46it/s]Epoch [4]

Mean LOSS on train: 0.1781868
Mean DICE on train: 0.842455478848455
Mean DICE on validation: 0.8848375744192143
100%|██████████| 116/116 [01:19<00:00,  1.45it/s]Epoch [5]

Mean LOSS on train: 0.19407524
Mean DICE on train: 0.8316372024543479
Mean DICE on validation: 0.8736485233919584
100%|██████████| 116/116 [01:20<00:00,  1.45it/s]Epoch [6]

Mean LOSS on train: 0.17685923
Mean DICE on train: 0.8450527297494892
Mean DICE on validation: 0.9044482530048251
100%|██████████| 116/116 [01:20<00:00,  1.45it/s]Epoch [7]

Mean LOSS on train: 0.16547738
Mean DICE on train: 0.8537855765490631
Mean DICE on validation: 0.912257082589504
100%|██████████| 116/116 [01:20<00:00,  1.45it/s]Epoch [8]

Mean LOSS on train: 0.14424534
Mean DICE on train: 0.8721624235956164
Mean DICE on validation: 0.8704900233720009
100%|██████████| 116/116 [01:20<00:00,  1.44it/s]Epoch [9]

Mean LOSS on train: 0.16361465
Mean DICE on train: 0.8556574360865783
Mean DICE on validation: 0.8738434102274512
CPU times: user 12min, sys: 10.7 s, total: 12min 11s
Wall time: 38min 8s
```

Figure 3.10: Dice coefficient with each epoch

In conclusion, the training and evaluation process provided insights into the performance of the `UNet` and `ResNeXtUNet` architectures. The Dice coefficient metric serves as a valuable tool for quantifying segmentation quality and tracking model performance over training epochs.

3.5.1 Testing

The performance of ResUNet was evaluated on the presplitted test set of whose results are satisfactory.

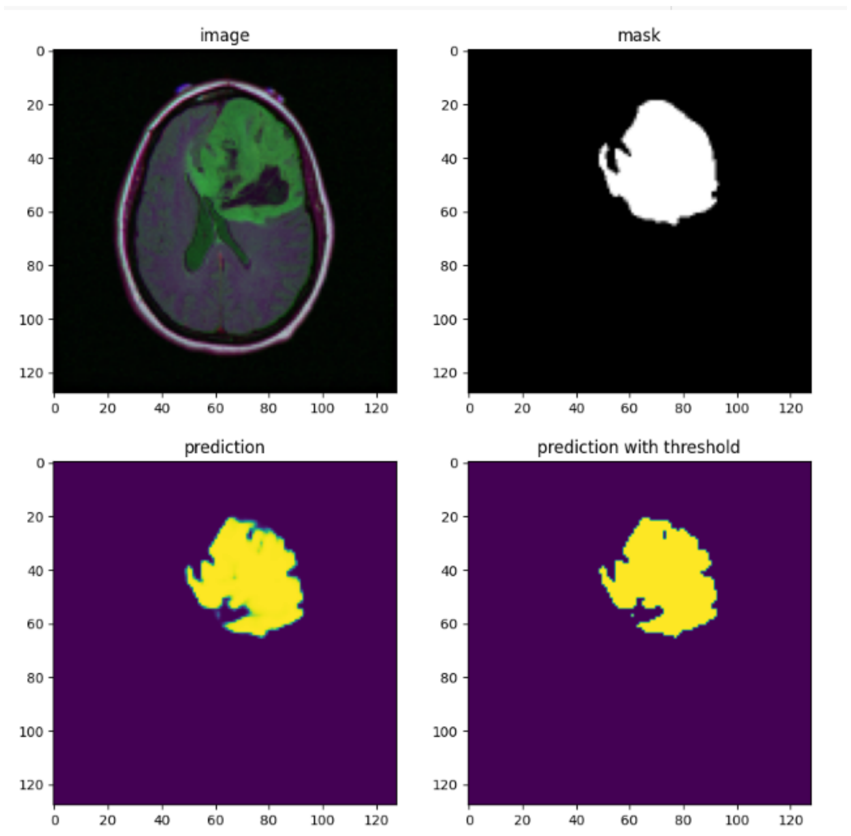


Figure 3.11: Example of prediction on testset1

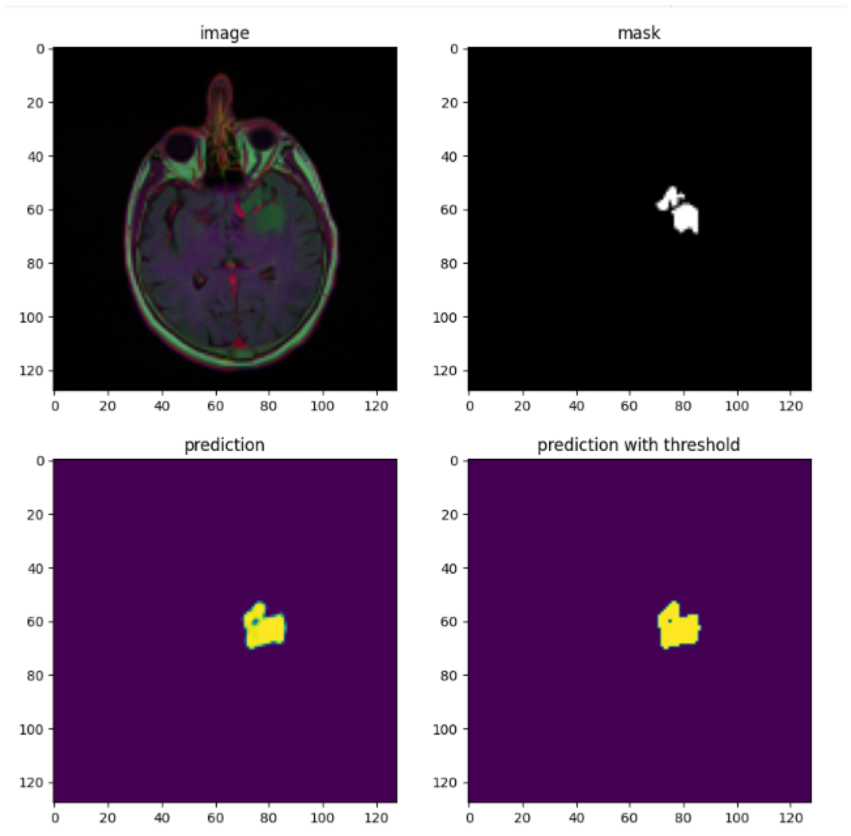


Figure 3.12: Example of prediction on testset2

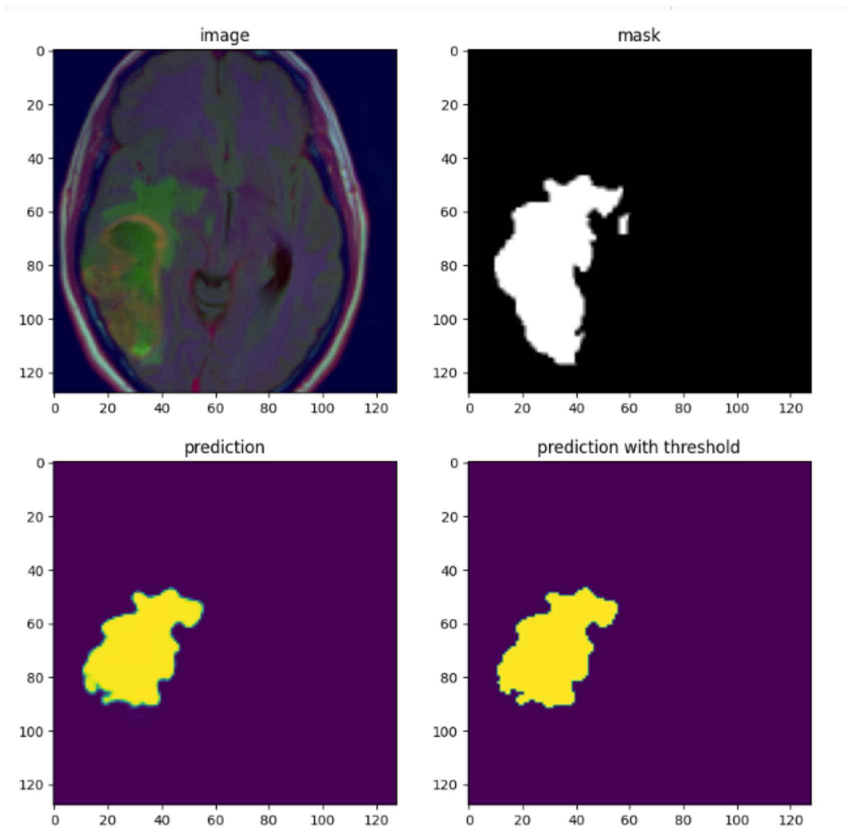


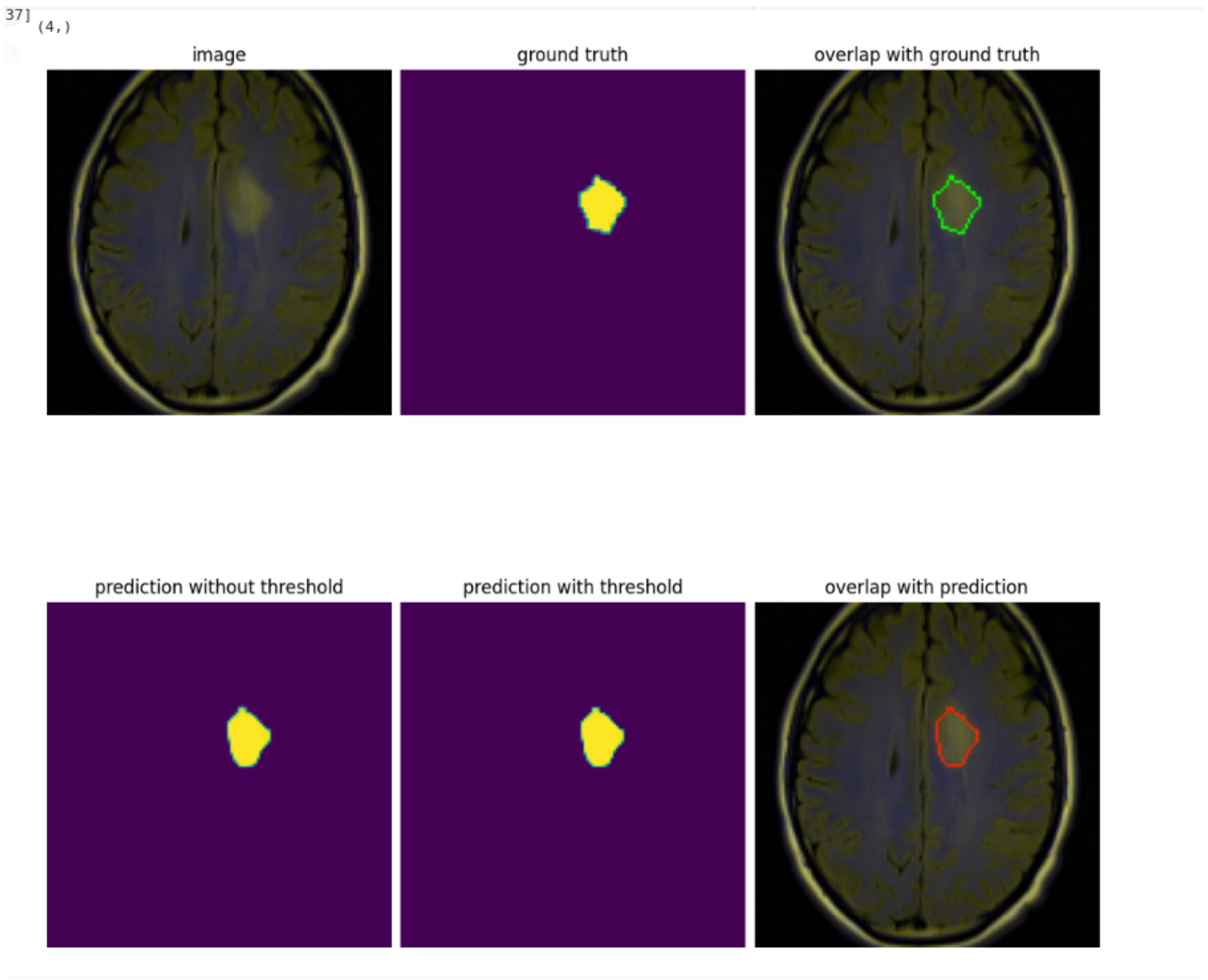
Figure 3.13: Example of prediction on testset3

4. Inference and Deployment

In this section, we delve into the process of performing inference with the trained models and deploying them using the Gradio library and Hugging Face's Transformers library.

4.1 Inference

After training the `UNet` and `ResNeXtUNet` architectures, the next step is to deploy these models to make predictions on new, unseen data. This process, often referred to as inference, involves passing input images through the models to generate corresponding segmentation masks.



37]
(4,)
image
ground truth
overlap with ground truth
prediction without threshold
prediction with threshold
overlap with prediction

Figure 4.1: A full fledged example of inference

4.2 Deployment with Gradio

Gradio is a user-friendly library that facilitates the creation of interactive web interfaces for machine learning models. It enables developers to build simple and intuitive UI components that allow users to upload images, interact with models, and visualize the model's predictions.

To deploy the models using Gradio, the following steps were taken:

1. **Model Loading:** The trained models were loaded using Hugging Face's Transformers library. This involved loading the model architecture and weights saved during training.

2. **User Interface Design:** Gradio provides a variety of UI components such as image uploaders, output display areas, and buttons. These components were utilized to design an interactive interface.

3. **Prediction Function:** A prediction function was defined, taking input images, preprocessing them as required, and passing them through the loaded models to generate segmentation masks.

4. **Interactive UI Creation:** Gradio's UI components were combined with the prediction function to create an interactive interface. Users could upload images, trigger predictions, and visualize the model's output.

5. **Web Deployment:** The Gradio interface was deployed to a web server, making it accessible to users through a web browser. This allowed users to perform segmentation on their own images using the trained models.

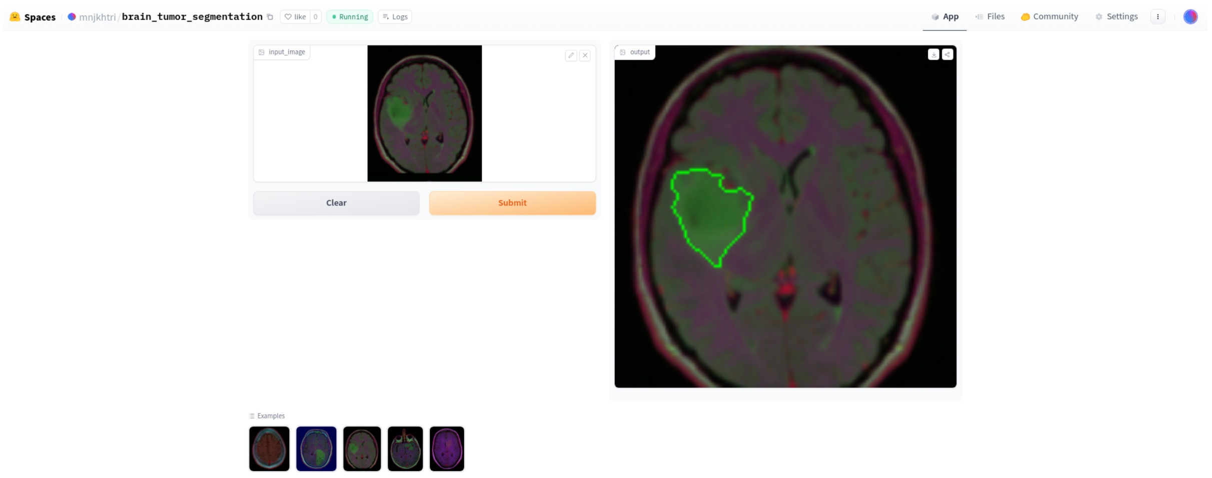


Figure 4.2: Deployment in HuggingFace using gradio

4.2.1 Benefits and Considerations

The combined usage of Hugging Face's Transformers library and Gradio offers several benefits:

- **Ease of Use:** Both libraries simplify the process of model deployment and interaction, enabling developers to quickly create usable interfaces.
- **Interactive Visualization:** Gradio's interactive interface allows users to see the model's predictions in real-time, aiding in understanding its performance.
- **Remote Accessibility:** Deploying the interface to a web server makes it accessible to users across different locations, increasing its usability.
- **Model Sharing:** Users can easily share the deployment link with others, enabling collaboration and feedback.

However, it's important to consider factors such as server costs, security measures, and model performance when deploying models to the web.

Incorporating Hugging Face's Transformers library and Gradio for inference and deployment greatly enhances the usability of trained models. The combination of these tools provides a seamless way to perform segmentation tasks on custom images while offering interactive visualizations for users.

5. Conclusion

In this project, we undertook the task of semantic segmentation for medical image analysis. We explored and implemented two distinct architectures, namely `UNet` and `ResNeXtUNet`, to tackle this challenging problem. The models were trained on a well-curated dataset and underwent rigorous preprocessing and augmentation techniques to enhance their generalization ability.

Our experiments demonstrated that both architectures were capable of producing accurate segmentation results. The `UNet` architecture, with its symmetric contracting and expanding paths, showed efficiency in segmenting medical images. On the other hand, the `ResNeXtUNet` architecture, with its use of residual connections and advanced feature extraction, showcased improved performance in capturing intricate structures.

The training and evaluation processes provided valuable insights into the significance of data preprocessing, augmentation, and hyperparameter tuning in achieving optimal segmentation results. We observed the importance of striking a balance between model complexity and overfitting, as well as leveraging transfer learning to improve convergence and accuracy.

For deployment, we harnessed the power of Hugging Face's Transformers library and Gradio to create an interactive web interface for users to perform segmentation on their own images using the trained models. This deployment allowed us to showcase the practical utility of our models beyond the training environment, enhancing their accessibility and usability.

In conclusion, our project not only explored the intricacies of semantic segmentation architectures and their training but also demonstrated the feasibility of deploying these models through user-friendly interfaces. The journey through data preprocessing, model design, training, evaluation, and deployment has provided us with valuable experience and insights into the realm of medical image analysis and deep learning.