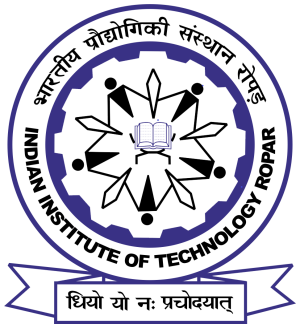
Brain Tumour Sub-Region Segmentation (BraTS21 Dataset)



End Semester Project Report

CS517: Digital Image Processing

Full Name	Enrollment No.
Malya Singh	2022CSZ0015
Seema	2022CSZ0006
Anshika	2021CSB1069

Course Instructor: Dr. Puneet Goyal

Department of Computer Science and Engineering
Indian Institute of Technology Ropar
Rupnagar, Punjab India

May 14, 2023

Contents

1 | Introduction

This project is based on **The Brain Tumor Segmentation (BraTS)** Continuous Challenge seeks to identify the current, state-of-the-art segmentation algorithms for brain diffuse glioma patients and their sub-regions.
This continuous challenge will culminate in an annual, state-of-the-field evaluation where we will ask participants to submit containerized versions of their best models which will be run against a held out testing data set, and the results presented at the annual MICCAI conference

2 | Objectives

The following are the objectives of the End Semester project challenge:

- Develop a method and produce segmentation labels of the different Gliomas.
- Identify various subregions of deadly Brain Tumour called Glioma.
- Test the developed methodology on provided BraTS21 dataset.

3 | Tools Used

For the implementation of this project, we used and explored various tools and technologies. Most of the work is done using the Python programming language in Google Colab.
Although, the project began with a simple segmentation model, various deep learning based models were utilized and explored in the implementation of this project. The models are enlisted below:

- **UNet Model:** based on Convolutional Neural Network architecture.
- **Swin UNETR Model:** Transformer-based Encoder-Decoder model
- **DeepLab Model:** based on Fully CNN architecture

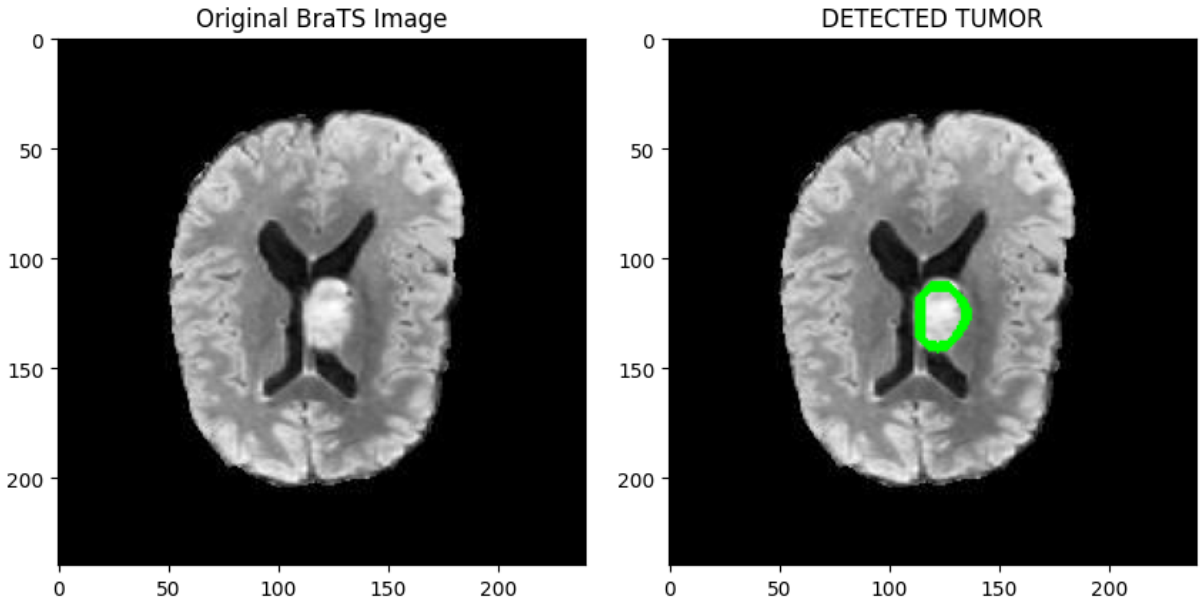
4 | Methodology

Here is the step-by-step description of our work done on this project

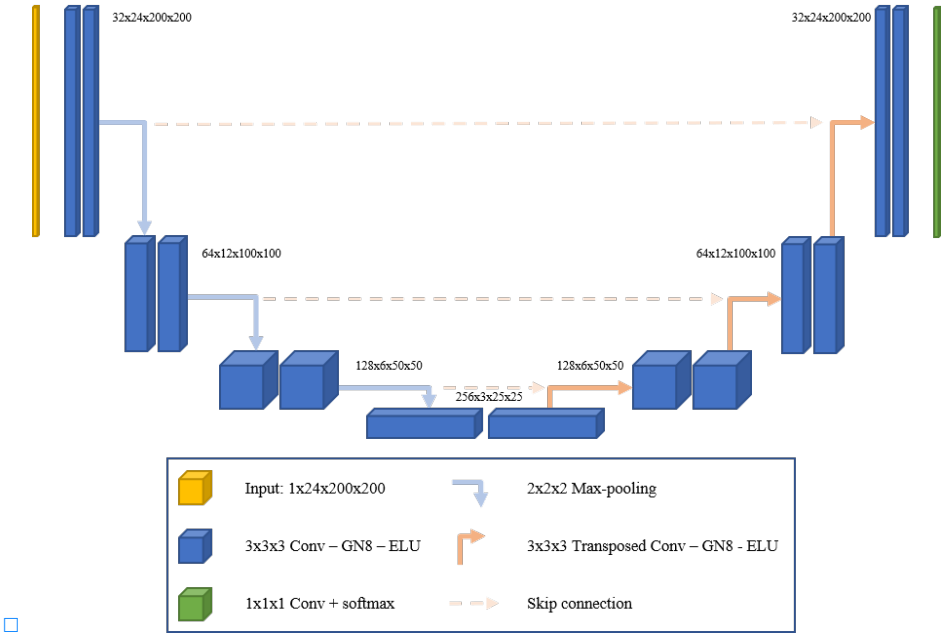
- We began with a simple model for segmentation. This model is based on the simple methodologies that we understood throughout the class and lab assignments.
  After reading the image, we notice by plotting its histogram that it needs to be enhanced. Hence, we apply some standard techniques for its enhancement.
  Gaussian blurring filter is applied on entire input image to obtain blurred version of it.
  For enhanced homogenity in the input image, the blurred image is added to the original grayscale image.
  Further, owing to the edge-preserving and denoising properties of the median filter, we apply it on the image.
  Then a kernel is generated as a morphological gradient. This improves the edge contrast across the entire image.
  Then thresholding is applied.
  Then we apply various morphological operations to develop a better region of interest after thresholding. Finally, we obtain the segmented model.

  Limitations: This method is case specific as for different data, it needs different thresholding values and morphological operations. This can lead to unreliable results in real life problems.

Hence we moved towards more deep learning based models.



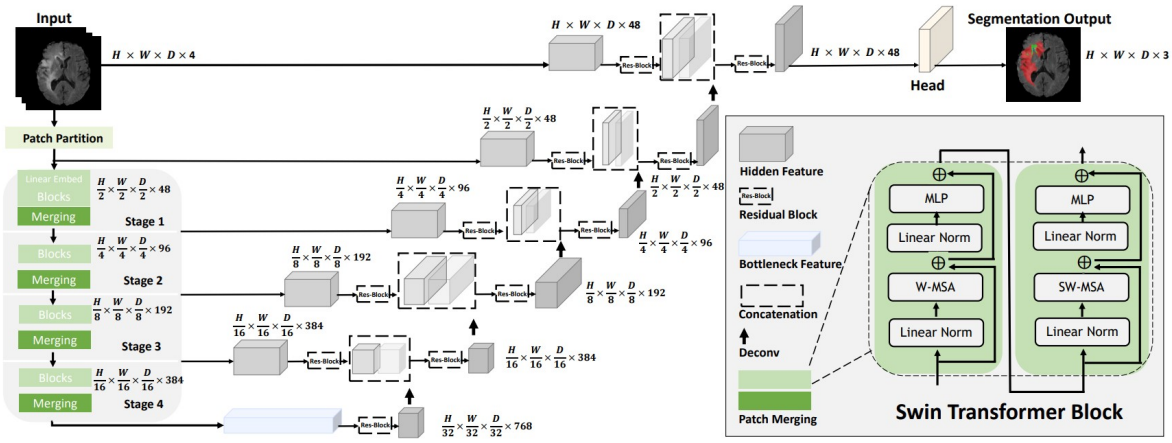
- As the simple segmentation using thresholding and Morphological Operations had various shortcomings, we moved to the deep learning based model called UNet Model.
  - UNet Model gets its name after its architecture. It is a U-shaped model that comprises of convolutional layers and two networks. It consists of an encoder followed by decoder. With the UNet model it is possible to solve two problems:
    - What part of image is the tumor?
    - Where is the tumor in the image?



UNet Model Architecture

- The popular "U-shaped" network architecture has achieved state-of-the-art performance benchmarks on different 2D and 3D semantic segmentation tasks and across various imaging modalities. However, due to the limited kernel size of convolution layers in FCNNs, their performance of modeling long-range information is sub-optimal, and this can lead to deficiencies in the segmentation of tumors

with variable sizes. On the other hand, transformer models have demonstrated excellent capabilities in capturing such long-range information in multiple domains, including natural language processing and computer vision.
We came across a novel segmentation model termed Swin UNEt TRansformers (Swin UNETR). In this model, the task of 3D brain tumor semantic segmentation is reformulated as a sequence to sequence prediction problem wherein multi-modal input data is projected into a 1D sequence of embedding and used as an input to a hierarchical Swin transformer as the encoder. The Swin transformer encoder extracts features at five different resolutions by utilizing shifted windows for computing self-attention and is connected to an FCNN-based decoder at each resolution via skip connections.

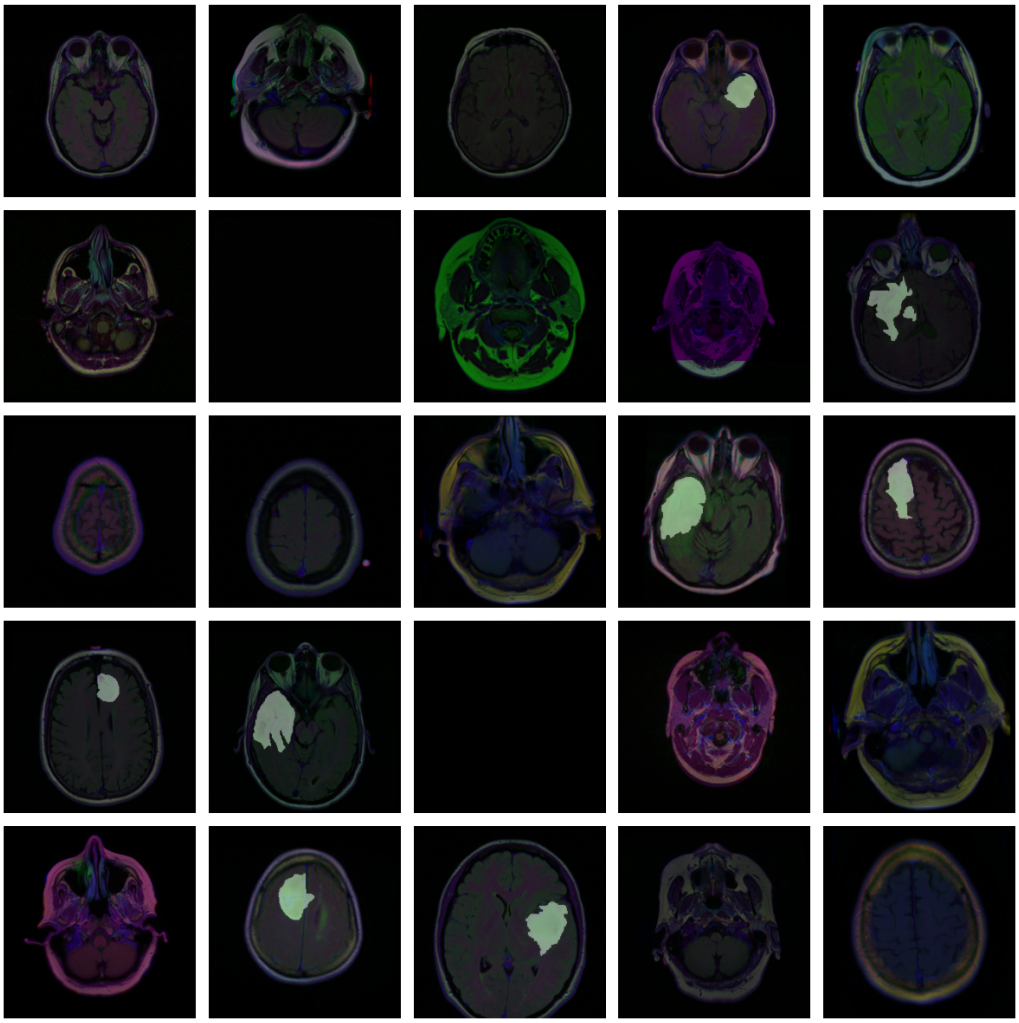


Overview of the Swin UNETR architecture. The input to our model is 3D multi-modal MRI images with 4 channels. The Swin UNETR creates non-overlapping patches of the input data and uses a patch partition layer to create windows with a desired size for computing the self-attention. The encoded feature representations in the Swin transformer are fed to a CNN-decoder via skip connection at multiple resolutions. Final segmentation output consists of 3 output channels corresponding to ET,WT and TC sub-regions.

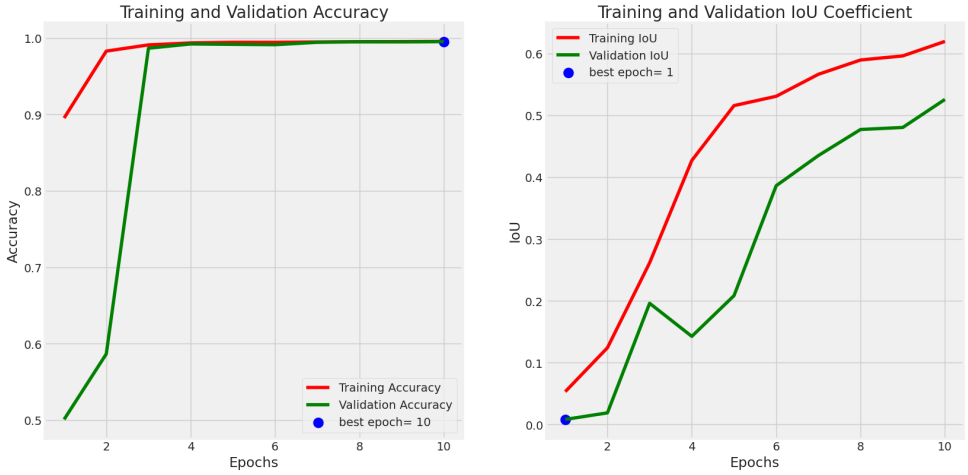
5 | Details of the Models

1. On training the BraTS21 Dataset on UNet Model, promising results were observed with great accuracy.
The structure of the attached model included:

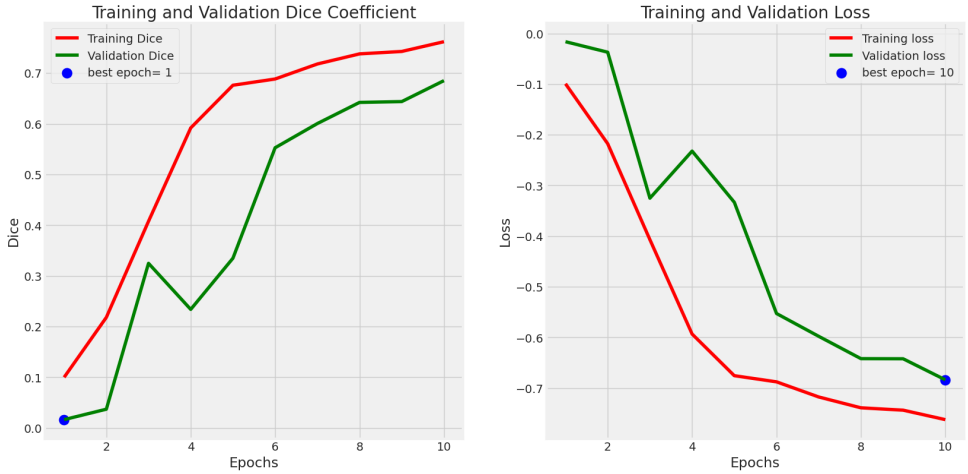
- Functions to create data frame from dataset :
  - *create_df* : Function to create data frame
  - *split_df* : Function to split data frame into train, valid, test
- Function to create image generators and augmentation : *create_gens*
- Functions that have UNet Structure: *unet*
- Functions for coefficients and loss :
  - *dice_coef* : function to create dice coefficient
  - *dice_loss* : function to create dice loss
  - *iou_coef* : function to create iou coefficient
- Function to show images sample : *show_images*
- Function to display training history : *plot_training*



Input Images from BraTS21 Dataset
All BraTS mpMRI scans are available as NIfTI files (.nii.gz) and describe a) native (T1) and b) post-contrast T1-weighted (T1Gd), c) T2-weighted (T2), and d) T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes, and were acquired with different clinical protocols and various scanners from multiple data contributing institutions.



(a) Training and Validation Accuracy a



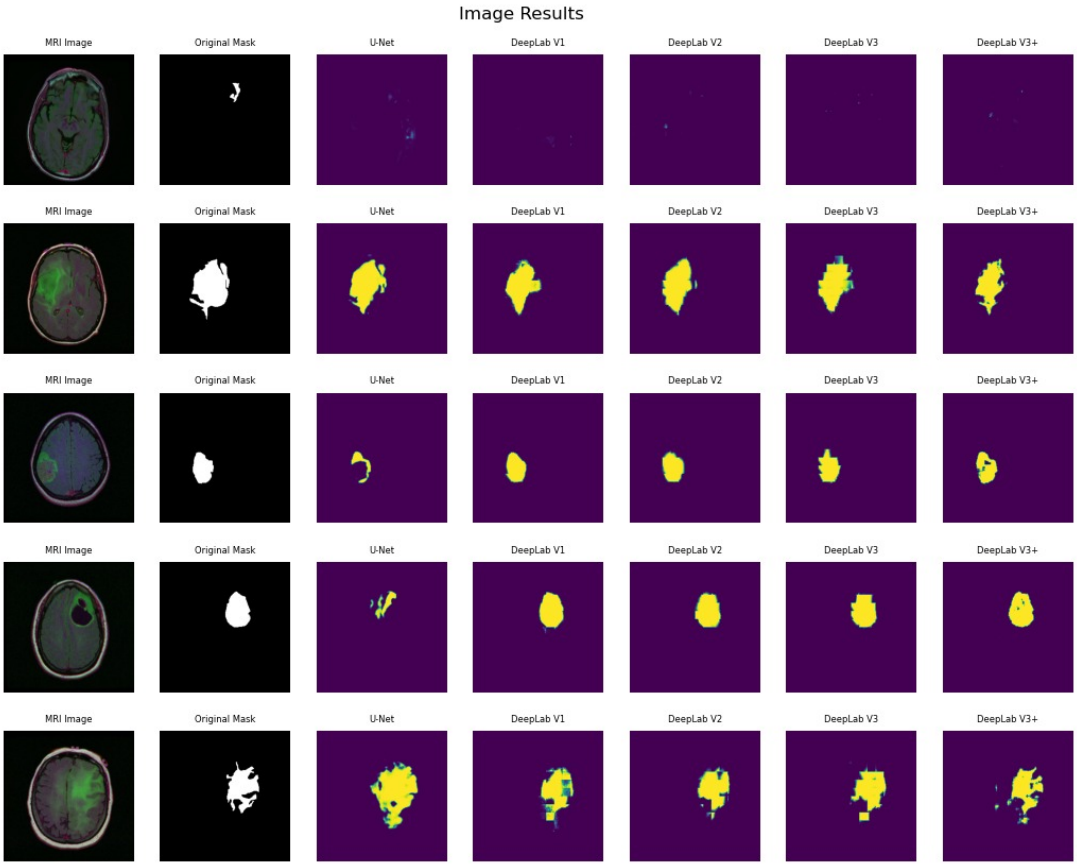
(b) Training and Validation Accuracy b

2. To test the accuracy of various deep learning models in order to develop a better model, we tested various models and simultaneously compared them as well.
   The structure of the attached DeepLab model has following dependencies included:

   - sklearn.model_selection
   - Tensorflow
   - Keras package: keras.utils, keras.preprocessing, keras.models, keras.optimizers, keras.layers

   The structure of the attached model included comparison and implementation of UNet, Deeplabv1, Deeplabv2, Deeplabv3, Deeplabv3+ .

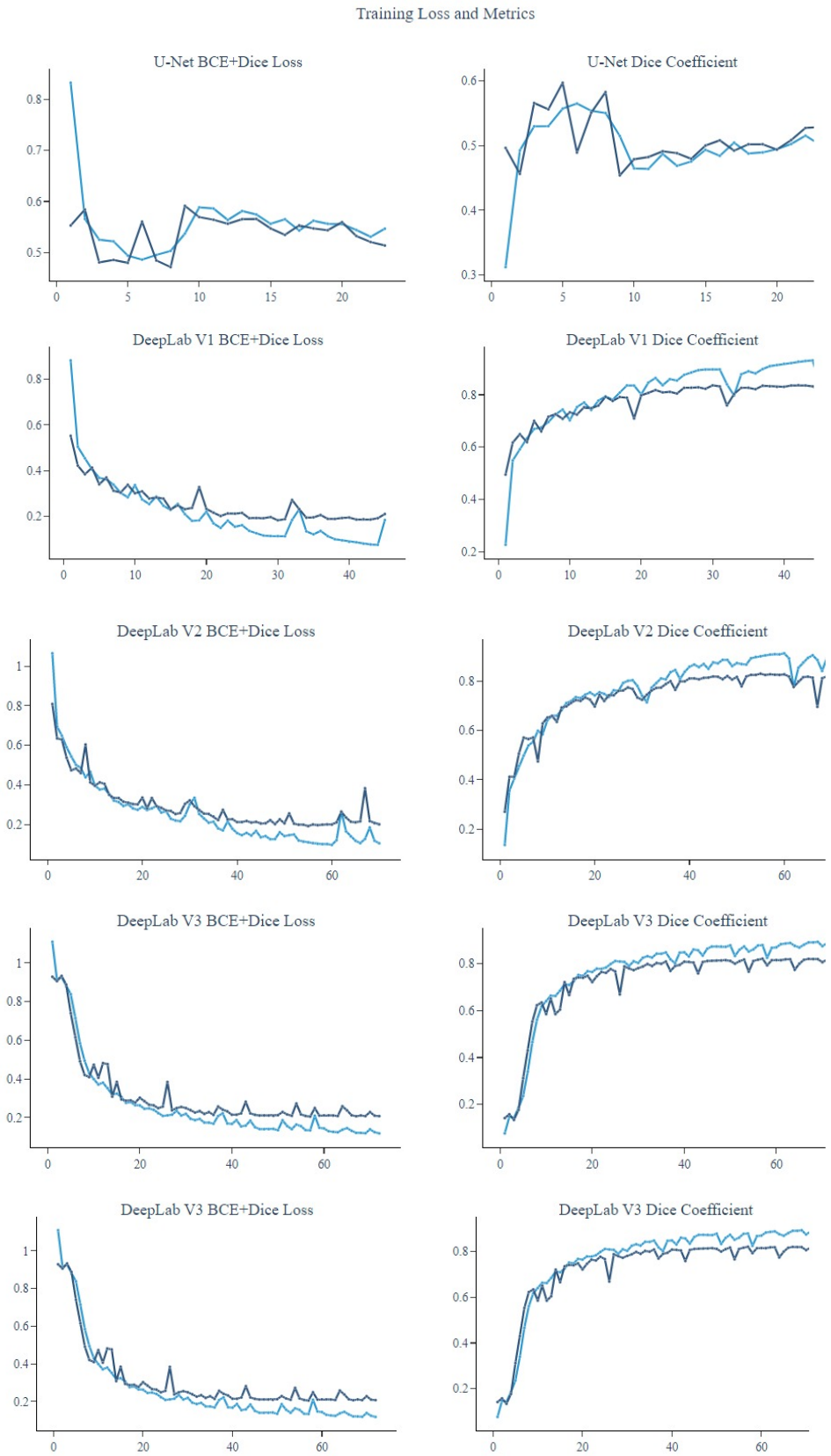
   - Preparation
   - Dataset
   - Image Segmentation
     - *Helper Functions*
     - *UNet*
     - *Deeplabv1*
     - *Deeplabv2*
     - *Deeplabv3*
     - *Deeplabv3+*
   - Testing Metrics : *Dice Coefficient*



Comparisons of various Deep Learning models against BraTS Dataset

	model	dice_coef
0	U-Net	0.350428
1	DeepLab V3+	0.651723
2	DeepLab V1	0.772613
3	DeepLab V2	0.806792
4	DeepLab V3	0.817676

All BraTS mpMRI scans are available as NIfTI files (.nii.gz) and describe a) native (T1) and b) post-contrast T1-weighted (T1Gd), c) T2-weighted (T2), and d) T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes, and were acquired with different clinical protocols and various scanners from multiple data contributing institutions.



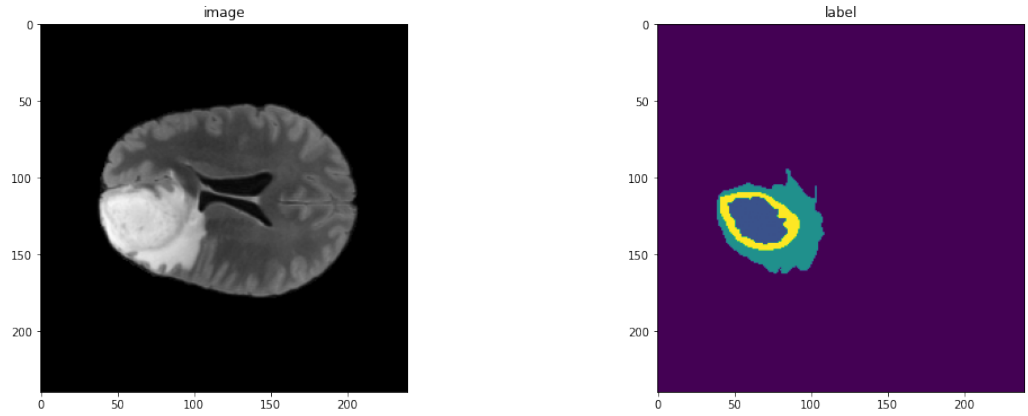
(a) Plots of Training Metrics for the Compared Deep Learning Models

3. On training the BraTS21 Dataset on SWIN UNeTR Model, promising results were observed with great accuracy.
The structure of the attached SWIN UNetR model has following dependencies included:

- MONAI version: 0.9.0rc3
- Numpy version: 1.22.2
- Pytorch version: 1.10.0

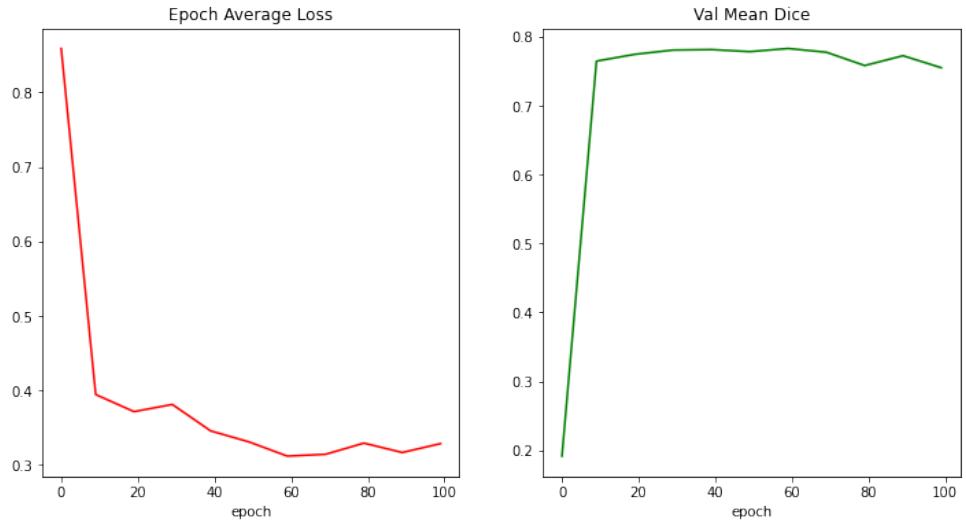
The structure of the attached model included:

- Setup average meter, fold reader, checkpoint saver : Class AverageMeter
  - *datafold_read*
  - *save_checkpoint*
- Setup dataloader : *get_loader*
- Set dataset root directory hyperparameters
- Checking data shape and visualize
- Creating SWIN UNetR Model
- Optimizer and Loss Functions

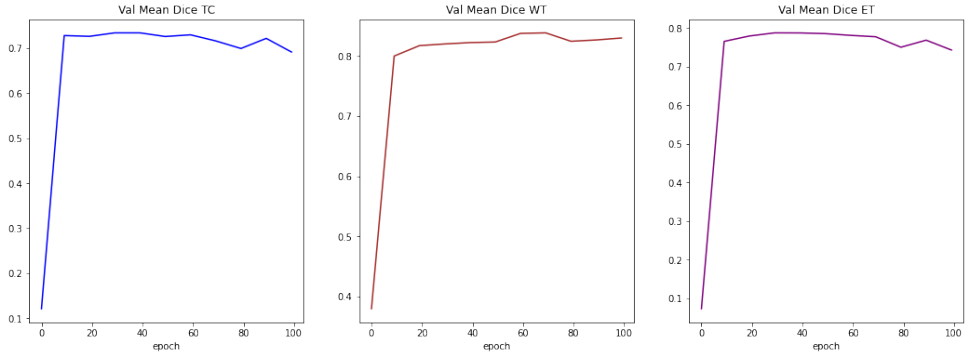


Input Images from BraTS21 Dataset as read by SWIN Model

All BraTS mpMRI scans are available as NIfTI files (.nii.gz) and describe a) native (T1) and b) post-contrast T1-weighted (T1Gd), c) T2-weighted (T2), and d) T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes, and were acquired with different clinical protocols and various scanners from multiple data contributing institutions.



(a) Training and Validation Accuracy a



(b) Training and Validation Accuracy b

6 | Results

Here are the results of the tested UNet model:

Figure 6.1: Output 1 of UNetR Model

Figure 6.2: Output 2 of UNetR Model

Figure 6.3: Output 3 of UNetR Model

Here are the results of the tested SWIN UNetR model:

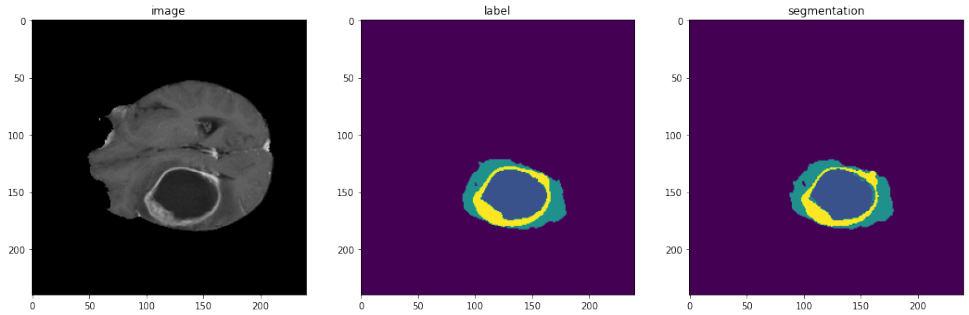


Figure 6.4: Sub-Region Segementation using SWIN UNetR Model

7 | Learning Outcomes

In this project we tried to develop a model to solve one of the prominent medical problems, ehich is Brain Tumor Sub-region segmentation.
Starting from a simple model based on simple thresholding and morphological operations.
Futher, we explored UNet model, with which we could segment simply the tumor region.
In order to have better insight on which model works the best, we compared various deep learning models, namely UNet, Deeplab V1, Deeplab V2, Deeplab V3, Deeplab V3+.
Finally, we explored SWIN UNetR model, with which we could segment the subregions as well.

8 | References

1. Hatamizadeh, A., Nath, V., Tang, Y., Yang, D., Roth, H., Xu, D. (2022). Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images. ArXiv. /abs/2201.01266
2. Nasim, M. A., Munem, A. A., Islam, M., Palash, M. A., Haque, M. M., Shah, F. M. (2022). Brain Tumor Segmentation using Enhanced U-Net Model with Empirical Analysis. ArXiv. /abs/2210.13336
3. Chen, L., Papandreou, G., Schroff, F., Adam, H. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. ArXiv. /abs/1706.05587

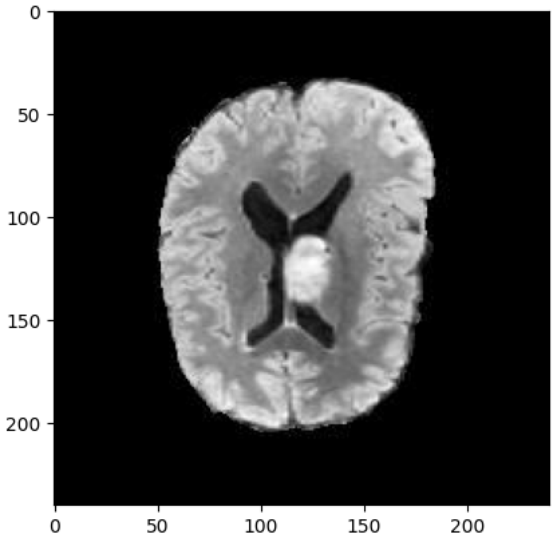
Brain Tumor Segmentation Model based on Simple Thresholding and Morphological Operations.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

Image loading

The image from BraTS dataset is converted into grayscale as it is easier to work with it.

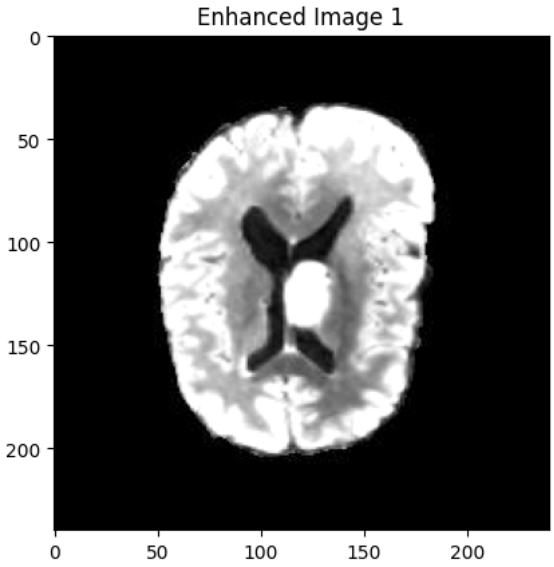
```
img = cv.imread("/content/image.0084.jpg", cv.IMREAD_GRAYSCALE)
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.show()

X = img.shape[0]

copy = np.copy(img)
```



Applying Gaussian Blurring to Image

Displaying Histogram of the real and enhances image for better homogenity

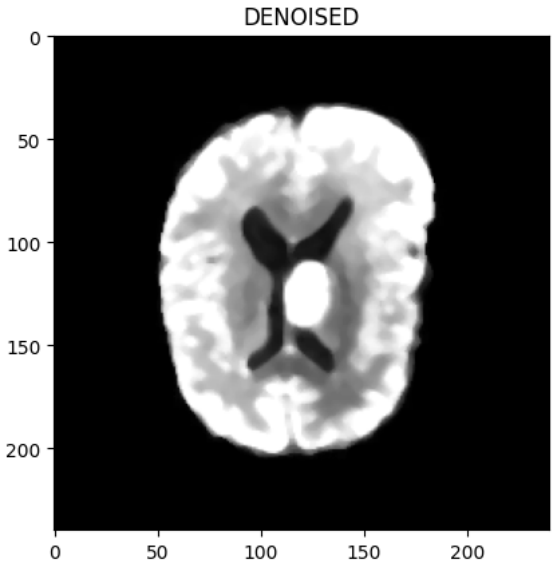
```
blur = cv.GaussianBlur(copy,(5,5),2)
enh = cv.add(copy,(cv.add(blur,-100)))
plt.imshow(enh, cmap='gray', vmin=0, vmax=255),plt.title("Enhanced Image 1")
plt.show()
```



Denoising the image

Applying Median Filter for denoising

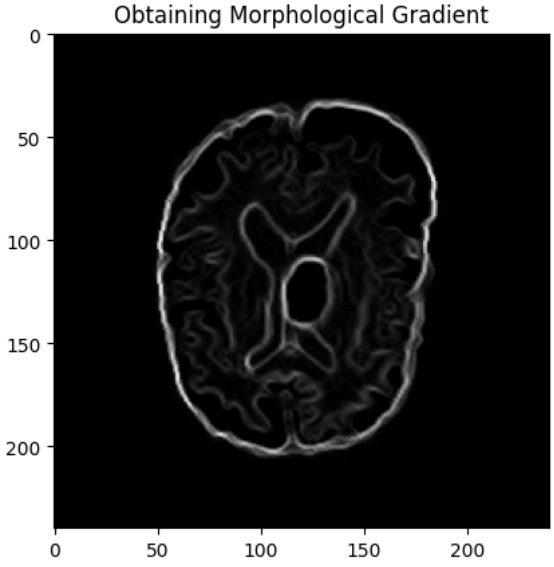
```
median = cv.medianBlur(enh,5)
plt.imshow(median, cmap='gray', vmin=0, vmax=255),plt.title("Denoised Image After Median Filter ")
plt.show()
```



Getting Morphological Gradient

To obtain Structuring element

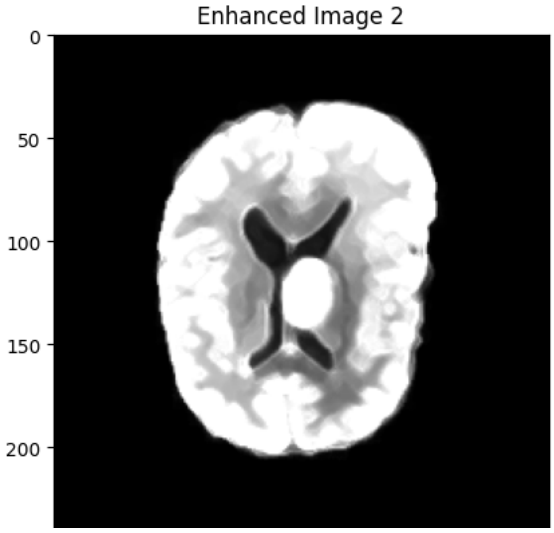
```
kernel = cv.getStructuringElement(cv.MORPH_CROSS,(3,3))
gradient = cv.morphologyEx(median, cv.MORPH_GRADIENT, kernel)
plt.imshow(gradient, cmap='gray', vmin=0, vmax=255),plt.title("Obtaining Morphological Gradient")
plt.show()
```



Further Enhancement

Obtaining better edges

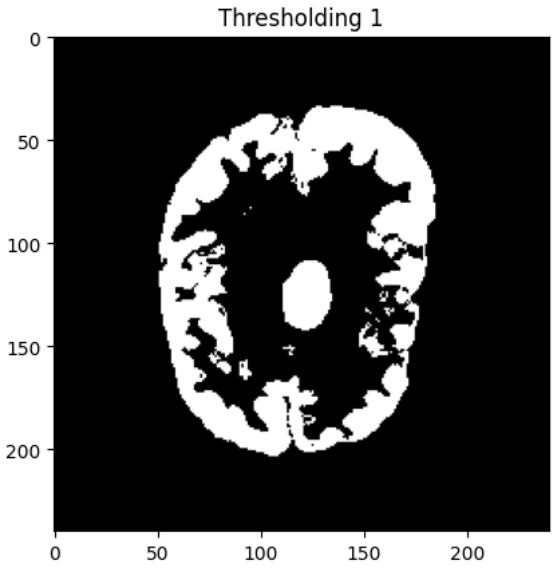
```
enh2 = cv.add(median,gradient)
plt.imshow(enh2, cmap='gray', vmin=0, vmax=255),plt.title("Enhanced Image 2")
plt.show()
```



Applying Thresholding

to segment the tumor

```
t = np.percentile(enh2,85)
ret,th = cv.threshold(enh2, t, 255, cv.THRESH_BINARY)
plt.imshow(th, cmap='gray', vmin=0, vmax=255),plt.title("Thresholding 1")
plt.show()
```



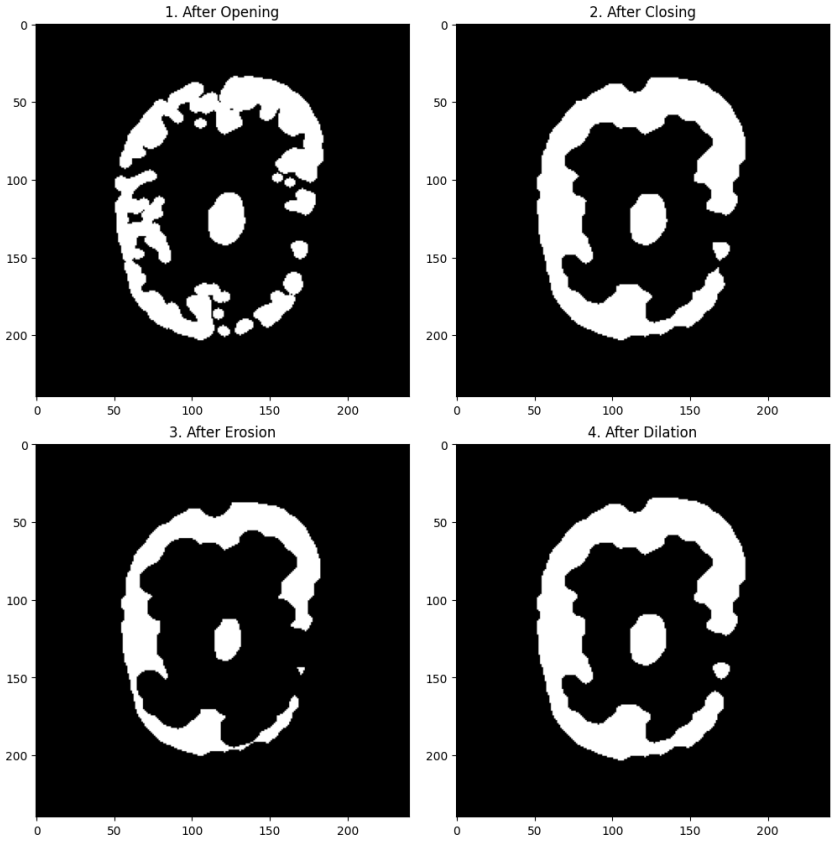
Applying Morphology operations

1. Opening
2. Closing
3. Erosion
4. Dilation

```
kernel_c = cv.getStructuringElement(cv.MORPH_ELLIPSE,(int((5*X)/100),int((5*X)/100))) #
kernel_e = cv.getStructuringElement(cv.MORPH_ELLIPSE,(int((3*X)/100),int((3*X)/100))) #
ker = cv.getStructuringElement(cv.MORPH_ELLIPSE,(int((7*X)/100),int((7*X)/100)))

plt.figure(figsize=(10,10),constrained_layout = True)

opening = cv.morphologyEx(th, cv.MORPH_OPEN, kernel_e) # to eliminate small uninteresting structures
plt.subplot(221),plt.imshow(opening, cmap='gray', vmin=0, vmax=255),plt.title("1. After Opening")

closing = cv.morphologyEx(opening, cv.MORPH_CLOSE, kernel_c) # to merge the remaining structures that may have been divided
plt.subplot(222),plt.imshow(closing, cmap='gray', vmin=0, vmax=255),plt.title("2. After Closing")

erosion = cv.erode(closing,kernel_e,iterations = 1)
plt.subplot(223),plt.imshow(erosion, cmap='gray', vmin=0, vmax=255),plt.title("3. After Erosion")
```

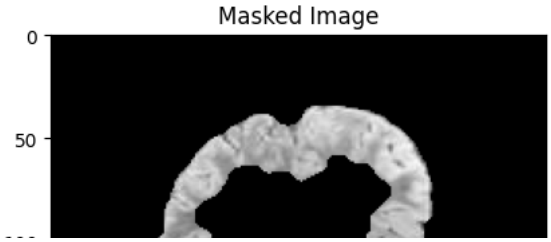
```
dilation = cv.dilate(erosion,kernel_e,iterations = 1)
plt.subplot(224),plt.imshow(dilation, cmap='gray', vmin=0, vmax=255),plt.title("4. After Dilation")
```

```
(<Axes: title={'center': '4. After Dilation'}>,
 <matplotlib.image.AxesImage at 0x7fea8df42560>,
 Text(0.5, 1.0, '4. After Dilation'))
```



Masking

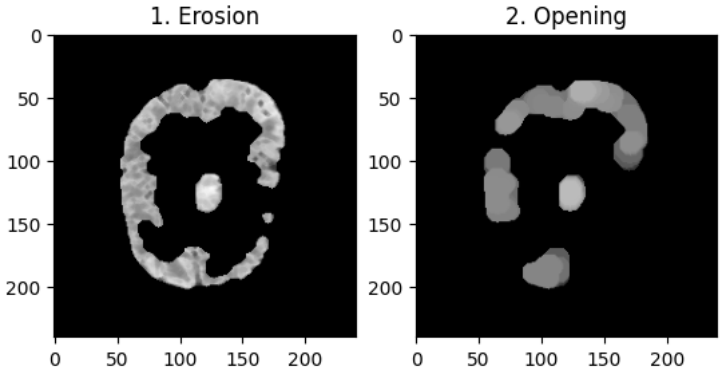
```
masked = cv.bitwise_and(copy, copy, mask=dilation)
plt.imshow(masked, cmap='gray', vmin=0, vmax=255),plt.title("Masked Image")
plt.show()
```



Applying Morphological Operations again

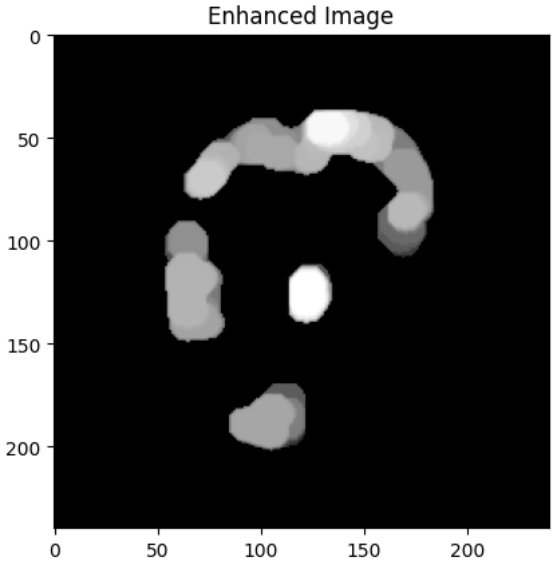
1. Erosion
2. Opening

```
s_erosion = cv.erode(masked,kernel,iterations = 1)
plt.subplot(121),plt.imshow(s_erosion, cmap='gray', vmin=0, vmax=255),plt.title("1. Erosion")

final = cv.morphologyEx(s_erosion, cv.MORPH_OPEN, ker)
plt.subplot(122),plt.imshow(final, cmap='gray', vmin=0, vmax=255),plt.title("2. Opening")
plt.show()
```



Enhancement by applying Gaussian Filter

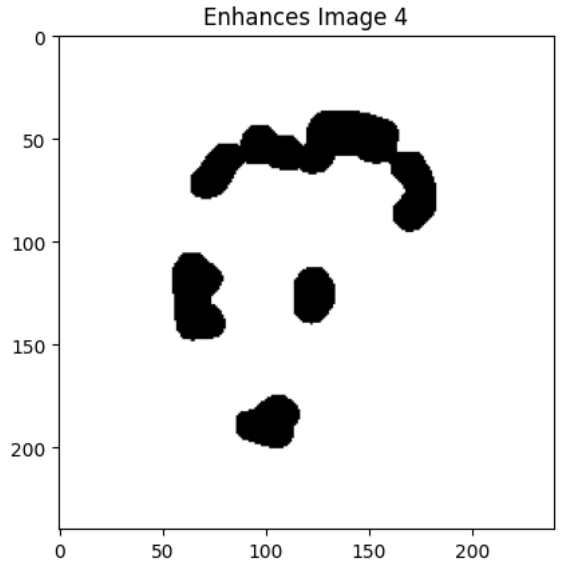
```
blur3 = cv.GaussianBlur(final,(3,3),0)
enh3 = cv.add(final,(cv.add(blur3,-100)))
plt.imshow(enh3, cmap='gray', vmin=0, vmax=255),plt.title("Enhanced Image 3")
plt.show()
```



Second thresholding

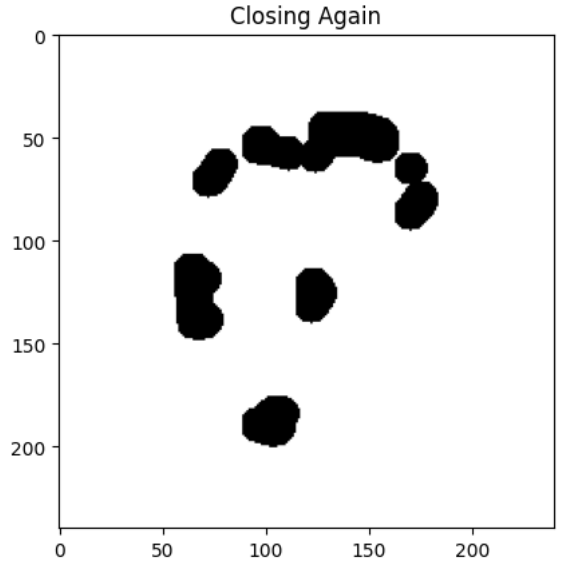
```
upper = np.percentile(enh3,92)
res = cv.inRange(enh3, 0, upper)
```

```
plt.imshow(res, cmap='gray', vmin=0, vmax=255),plt.title("Enhances Image 4")
plt.show()
```



Final morphology step

```
fin = cv.morphologyEx(res, cv.MORPH_CLOSE, cv.getStructuringElement(cv.MORPH_ELLIPSE,(int((7*X)/100),int((7*X)/100))))
plt.imshow(fin, cmap='gray', vmin=0, vmax=255),plt.title("Closing Again")
plt.show()
```

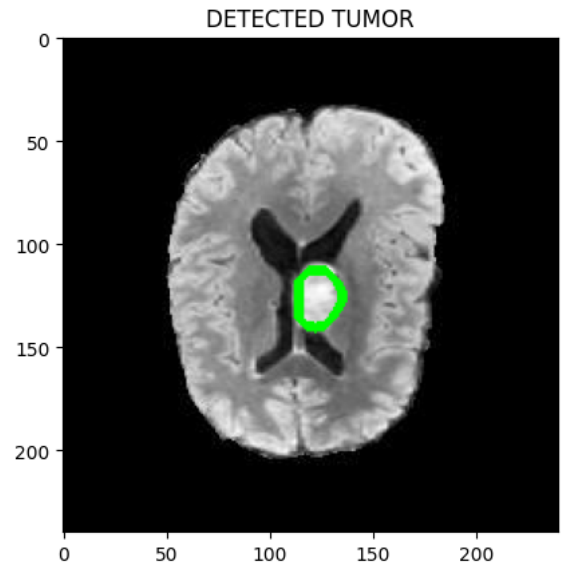


Contouring Step

In order to manage the contour found at the outer borders of the entire image, an if structure has been utilized. The outer contour is the result of morphology and masking operations performed on the image. Lastly the area and perimeter of the tumor is computed if detected.

```
copy_rgb = cv.cvtColor(copy,cv.COLOR_BGR2RGB) # necessary step in order to print out the original image colors correctly

contours, hierarchy = cv.findContours(fin, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

if len(contours) > 1:
    cnt = contours[1]
    if len(contours) > 2:
        cv.drawContours(copy_rgb, contours, 2, (0,255,0), 3)
        plt.imshow(copy_rgb),plt.title("DETECTED TUMOR")
        plt.show()
    else:
        cv.drawContours(copy_rgb, contours, 1, (0,255,0), 3)
        plt.imshow(copy_rgb),plt.title("DETECTED TUMOR")
        plt.show()

    area = int(cv.contourArea(cnt))
```

```
    area = int(cv.contourArea(cnt))
    perimeter = int(cv.arcLength(cnt,True))

    print("Area:",area,"px")
    print("Perimeter:",perimeter,"px")
else:
    print("No tumor detected")
```




```
Area: 577 px
Perimeter: 91 px
```

```
# import system libs
import os
import time
import random
import pathlib
import itertools
from glob import glob
from tqdm import tqdm_notebook, tnrange

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
%matplotlib inline
from skimage.color import rgb2gray
from skimage.morphology import label
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from skimage.io import imread, imshow, concatenate_images

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model, load_model, save_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import Input, Activation, BatchNormalization, Dropout, Lambda, Conv2D, Conv2DTranspose, MaxPooling2D, concat

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')
```

```
modules loaded
```

Create needed functions

Function to create dataframe from dataset

```
# function to create dataframe
def create_df(data_dir):
    images_paths = []
    masks_paths = glob(f'{data_dir}/*/*_mask*')

    for i in masks_paths:
        images_paths.append(i.replace('_mask', ''))

    df = pd.DataFrame(data= {'images_paths': images_paths, 'masks_paths': masks_paths})

    return df

# Function to split dataframe into train, valid, test
def split_df(df):
    # create train_df
    train_df, dummy_df = train_test_split(df, train_size= 0.8)

    # create valid_df and test_df
    valid_df, test_df = train_test_split(dummy_df, train_size= 0.5)

    return train_df, valid_df, test_df
```

Function to create image generators and augmentation

```
def create_gens(df, aug_dict):
    img_size = (256, 256)
    batch_size = 40
```

```
    img_gen = ImageDataGenerator(**aug_dict)
    msk_gen = ImageDataGenerator(**aug_dict)

    # Create general generator
    image_gen = img_gen.flow_from_dataframe(df, x_col='images_paths', class_mode=None, color_mode='rgb', target_size=img_size,
                                            batch_size=batch_size, save_to_dir=None, save_prefix='image', seed=1)

    mask_gen = msk_gen.flow_from_dataframe(df, x_col='masks_paths', class_mode=None, color_mode='grayscale', target_size=img_size,
                                            batch_size=batch_size, save_to_dir=None, save_prefix= 'mask', seed=1)

    gen = zip(image_gen, mask_gen)

    for (img, msk) in gen:
        img = img / 255
        msk = msk / 255
        msk[msk > 0.5] = 1
        msk[msk <= 0.5] = 0

        yield (img, msk)
```

Function that have Unet structure

```
def unet(input_size=(256, 256, 3)):
    inputs = Input(input_size)

    # First DownConvolution / Encoder Leg will begin, so start with Conv2D
    conv1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(inputs)
    bn1 = Activation("relu")(conv1)
    conv1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(bn1)
    bn1 = BatchNormalization(axis=3)(conv1)
    bn1 = Activation("relu")(bn1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(bn1)

    conv2 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(pool1)
    bn2 = Activation("relu")(conv2)
    conv2 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(bn2)
    bn2 = BatchNormalization(axis=3)(conv2)
    bn2 = Activation("relu")(bn2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(bn2)

    conv3 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(pool2)
    bn3 = Activation("relu")(conv3)
    conv3 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(bn3)
    bn3 = BatchNormalization(axis=3)(conv3)
    bn3 = Activation("relu")(bn3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(bn3)

    conv4 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(pool3)
    bn4 = Activation("relu")(conv4)
    conv4 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(bn4)
    bn4 = BatchNormalization(axis=3)(conv4)
    bn4 = Activation("relu")(bn4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(bn4)

    conv5 = Conv2D(filters=1024, kernel_size=(3, 3), padding="same")(pool4)
    bn5 = Activation("relu")(conv5)
    conv5 = Conv2D(filters=1024, kernel_size=(3, 3), padding="same")(bn5)
    bn5 = BatchNormalization(axis=3)(conv5)
    bn5 = Activation("relu")(bn5)

    """ Now UpConvolution / Decoder Leg will begin, so start with Conv2DTranspose
    The gray arrows (in the above image) indicate the skip connections that concatenate the encoder feature map with the decoder, which h
    """ After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more prec

    up6 = concatenate([Conv2DTranspose(512, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn5), conv4], axis=3)
    conv6 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(up6)
    bn6 = Activation("relu")(conv6)
    conv6 = Conv2D(filters=512, kernel_size=(3, 3), padding="same")(bn6)
    bn6 = BatchNormalization(axis=3)(conv6)
    bn6 = Activation("relu")(bn6)

    up7 = concatenate([Conv2DTranspose(256, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn6), conv3], axis=3)
    conv7 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(up7)
    bn7 = Activation("relu")(conv7)
    conv7 = Conv2D(filters=256, kernel_size=(3, 3), padding="same")(bn7)
    bn7 = BatchNormalization(axis=3)(conv7)
    bn7 = Activation("relu")(bn7)

    up8 = concatenate([Conv2DTranspose(128, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn7), conv2], axis=3)
    conv8 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(up8)
```

```
    bn8 = Activation("relu")(conv8)
    conv8 = Conv2D(filters=128, kernel_size=(3, 3), padding="same")(bn8)
    bn8 = BatchNormalization(axis=3)(conv8)
    bn8 = Activation("relu")(bn8)

    up9 = concatenate([Conv2DTranspose(64, kernel_size=(2, 2), strides=(2, 2), padding="same")(bn8), conv1], axis=3)
    conv9 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(up9)
    bn9 = Activation("relu")(conv9)
    conv9 = Conv2D(filters=64, kernel_size=(3, 3), padding="same")(bn9)
    bn9 = BatchNormalization(axis=3)(conv9)
    bn9 = Activation("relu")(bn9)

    conv10 = Conv2D(filters=1, kernel_size=(1, 1), activation="sigmoid")(bn9)

    return Model(inputs=[inputs], outputs=[conv10])
```

Functions for coefficients and loss

```
# function to create dice coefficient
def dice_coef(y_true, y_pred, smooth=100):
    y_true_flatten = K.flatten(y_true)
    y_pred_flatten = K.flatten(y_pred)

    intersection = K.sum(y_true_flatten * y_pred_flatten)
    union = K.sum(y_true_flatten) + K.sum(y_pred_flatten)
    return (2 * intersection + smooth) / (union + smooth)

# function to create dice loss
def dice_loss(y_true, y_pred, smooth=100):
    return -dice_coef(y_true, y_pred, smooth)

# function to create iou coefficient
def iou_coef(y_true, y_pred, smooth=100):
    intersection = K.sum(y_true * y_pred)
    sum = K.sum(y_true + y_pred)
    iou = (intersection + smooth) / (sum - intersection + smooth)
    return iou
```

Function to show images sample

```
def show_images(images, masks):
    plt.figure(figsize=(12, 12))
    for i in range(25):
        plt.subplot(5, 5, i+1)
        img_path = images[i]
        mask_path = masks[i]
        # read image and convert it to RGB scale
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # read mask
        mask = cv2.imread(mask_path)
        # sho image and mask
        plt.imshow(image)
        plt.imshow(mask, alpha=0.4)

        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

Function to display training history

```
def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and losses with the best epoch in both of them.
    '''

    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_iou = hist.history['iou_coef']
    tr_dice = hist.history['dice_coef']
    tr_loss = hist.history['loss']

    val_acc = hist.history['val_accuracy']
    val_iou = hist.history['val_iou_coef']
```

```
val_dice = hist.history['val_dice_coef']
val_loss = hist.history['val_loss']

index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
index_iou = np.argmax(iou_coef)
iou_highest = val_iou[index_iou]
index_dice = np.argmax(dice_coef)
dice_highest = val_dice[index_dice]
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]

Epochs = [i+1 for i in range(len(tr_acc))]

acc_label = f'best epoch= {str(index_acc + 1)}'
iou_label = f'best epoch= {str(index_iou + 1)}'
dice_label = f'best epoch= {str(index_dice + 1)}'
loss_label = f'best epoch= {str(index_loss + 1)}'

# Plot training history
plt.figure(figsize= (20, 20))
plt.style.use('fivethirtyeight')

# Training Accuracy
plt.subplot(2, 2, 1)
plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Training IoU
plt.subplot(2, 2, 2)
plt.plot(Epochs, tr_iou, 'r', label= 'Training IoU')
plt.plot(Epochs, val_iou, 'g', label= 'Validation IoU')
plt.scatter(index_iou + 1 , iou_highest, s= 150, c= 'blue', label= iou_label)
plt.title('Training and Validation IoU Coefficient')
plt.xlabel('Epochs')
plt.ylabel('IoU')
plt.legend()

# Training Dice
plt.subplot(2, 2, 3)
plt.plot(Epochs, tr_dice, 'r', label= 'Training Dice')
plt.plot(Epochs, val_dice, 'g', label= 'Validation Dice')
plt.scatter(index_dice + 1 , dice_highest, s= 150, c= 'blue', label= dice_label)
plt.title('Training and Validation Dice Coefficient')
plt.xlabel('Epochs')
plt.ylabel('Dice')
plt.legend()

# Training Loss
plt.subplot(2, 2, 4)
plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout
plt.show()
```

Model Structure

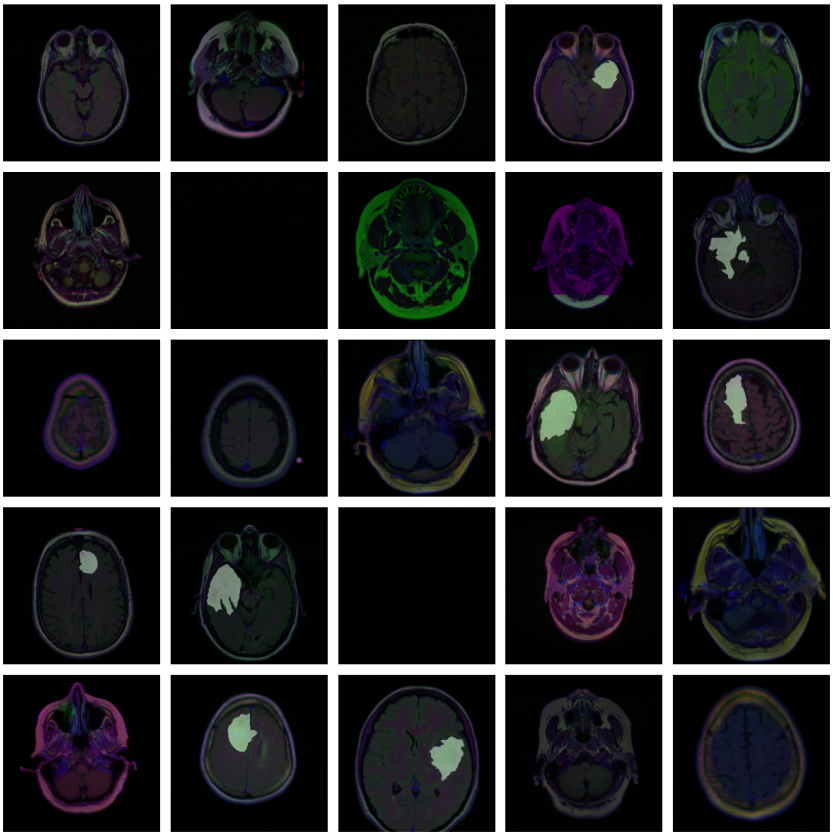
Start reading data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data_dir = '/content/drive/MyDrive/brats_datatset_/lgg-mri-segmentation/brats_3m'
```

```
df = create_df(data_dir)
train_df, valid_df, test_df = split_df(df)

tr_aug_dict = dict(rotation_range=0.2,
                           width_shift_range=0.05,
                           height_shift_range=0.05,
                           shear_range=0.05,
                           zoom_range=0.05,
                           horizontal_flip=True,
                           fill_mode='nearest')

train_gen = create_gens(train_df, aug_dict=tr_aug_dict)
valid_gen = create_gens(valid_df, aug_dict={})
test_gen = create_gens(test_df, aug_dict={})

show_images(list(train_df['images_paths']), list(train_df['masks_paths']))
```



Unet Model

```
model = unet()
model.compile(Adamax(learning_rate= 0.001), loss= dice_loss, metrics= ['accuracy', iou_coef, dice_coef])

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
```

```
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 256, 256, 64  1792        ['input_1[0][0]']
                                )

 activation (Activation)        (None, 256, 256, 64  0           ['conv2d[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 256, 256, 64  36928       ['activation[0][0]']
                                )

 batch_normalization (BatchNorm  (None, 256, 256, 64  256        ['conv2d_1[0][0]']
 alization)                     )

 activation_1 (Activation)      (None, 256, 256, 64  0           ['batch_normalization[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 128, 128, 64  0           ['activation_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 128, 128, 12  73856       ['max_pooling2d[0][0]']
                                8)

 activation_2 (Activation)      (None, 128, 128, 12  0           ['conv2d_2[0][0]']
                                8)

 conv2d_3 (Conv2D)              (None, 128, 128, 12  147584      ['activation_2[0][0]']
                                8)

 batch_normalization_1 (BatchNo  (None, 128, 128, 12  512        ['conv2d_3[0][0]']
 rmalization)                   8)

 activation_3 (Activation)      (None, 128, 128, 12  0           ['batch_normalization_1[0][0]']
                                8)

 max_pooling2d_1 (MaxPooling2D)  (None, 64, 64, 128)  0          ['activation_3[0][0]']

 conv2d_4 (Conv2D)              (None, 64, 64, 256)  295168      ['max_pooling2d_1[0][0]']

 activation_4 (Activation)      (None, 64, 64, 256)  0           ['conv2d_4[0][0]']

 conv2d_5 (Conv2D)              (None, 64, 64, 256)  590080      ['activation_4[0][0]']

 batch_normalization_2 (BatchNo  (None, 64, 64, 256)  1024       ['conv2d_5[0][0]']
 rmalization)

 activation_5 (Activation)      (None, 64, 64, 256)  0           ['batch_normalization_2[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 32, 32, 256)  0          ['activation_5[0][0]']

 conv2d_6 (Conv2D)              (None, 32, 32, 512)  1180160     ['max_pooling2d_2[0][0]']
```

Model training

```
epochs = 10
batch_size = 40
callbacks = [ModelCheckpoint('unet.hdf5', verbose=0, save_best_only=True)]

history = model.fit(train_gen,
                    steps_per_epoch=len(train_df) / batch_size,
                    epochs=epochs,
                    verbose=1,
                    callbacks=callbacks,
                    validation_data = valid_gen,
                    validation_steps=len(valid_df) / batch_size)
```

```
Found 3143 validated image filenames.
Found 3143 validated image filenames.
Epoch 1/10
79/78 [==============================] - ETA: -23s - loss: -0.0993 - accuracy: 0.8953 - iou_coef: 0.0532 - dice_coef: 0.0998Found 3
Found 393 validated image filenames.
78/78 [==============================] - 4894s 62s/step - loss: -0.0993 - accuracy: 0.8953 - iou_coef: 0.0532 - dice_coef: 0.0998 -
Epoch 2/10
78/78 [==============================] - 170s 2s/step - loss: -0.2176 - accuracy: 0.9829 - iou_coef: 0.1243 - dice_coef: 0.2180 - v
Epoch 3/10
78/78 [==============================] - 173s 2s/step - loss: -0.4070 - accuracy: 0.9910 - iou_coef: 0.2617 - dice_coef: 0.4076 - v
Epoch 4/10
78/78 [==============================] - 168s 2s/step - loss: -0.5933 - accuracy: 0.9936 - iou_coef: 0.4273 - dice_coef: 0.5918 - v
Epoch 5/10
78/78 [==============================] - 170s 2s/step - loss: -0.6755 - accuracy: 0.9944 - iou_coef: 0.5161 - dice_coef: 0.6760 - v
Epoch 6/10
78/78 [==============================] - 173s 2s/step - loss: -0.6876 - accuracy: 0.9943 - iou_coef: 0.5311 - dice_coef: 0.6883 - v
Epoch 7/10
```

```
78/78 [==============================] - 170s 2s/step - loss: -0.7175 - accuracy: 0.9948 - iou_coef: 0.5667 - dice_coef: 0.7179 - v
Epoch 8/10
78/78 [==============================] - 170s 2s/step - loss: -0.7389 - accuracy: 0.9951 - iou_coef: 0.5899 - dice_coef: 0.7378 - v
Epoch 9/10
78/78 [==============================] - 174s 2s/step - loss: -0.7436 - accuracy: 0.9953 - iou_coef: 0.5965 - dice_coef: 0.7426 - v
Epoch 10/10
78/78 [==============================] - 169s 2s/step - loss: -0.7622 - accuracy: 0.9955 - iou_coef: 0.6199 - dice_coef: 0.7619 - v
```

```
plot_training(history)
```

Model Evaluation

```
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print("Train IoU: ", train_score[2])
print("Train Dice: ", train_score[3])
print('-' * 20)
```

```
print("Valid Loss: ", valid_score[0])
print("Valid Accuracy: ", valid_score[1])
print("Valid IoU: ", valid_score[2])
print("Valid Dice: ", valid_score[3])
print('-' * 20)

print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
print("Test IoU: ", test_score[2])
print("Test Dice: ", test_score[3])
```

```
131/131 [==============================] - 152s 1s/step - loss: -0.7042 - accuracy: 0.9952 - iou_coef: 0.5530 - dice_coef: 0.7041
131/131 [==============================] - 86s 659ms/step - loss: -0.6812 - accuracy: 0.9952 - iou_coef: 0.5242 - dice_coef: 0.6803
Found 393 validated image filenames.
Found 393 validated image filenames.
131/131 [==============================] - 574s 4s/step - loss: -0.7458 - accuracy: 0.9957 - iou_coef: 0.6020 - dice_coef: 0.7464
Train Loss:  -0.7041603326797485
Train Accuracy:  0.9952444434165955
Train IoU:  0.5529682040214539
Train Dice:  0.7040674686431885
--------------------
Valid Loss:  -0.6811785101890564
Valid Accuracy:  0.9952473640441895
Valid IoU:  0.5242473483085632
Valid Dice:  0.6803381443023682
--------------------
Test Loss:  -0.7457866072654724
Test Accuracy:  0.9957333207130432
Test IoU:  0.6019558310508728
Test Dice:  0.7463856935501099
```

Prediction

```
for _ in range(20):
    index = np.random.randint(1, len(test_df.index))
    img = cv2.imread(test_df['images_paths'].iloc[index])
    img = cv2.resize(img, (256, 256))
    img = img/255
    img = img[np.newaxis, :, :, : ]

    predicted_img = model.predict(img)

    plt.figure(figsize=(12, 12))

    plt.subplot(1, 3, 1)
    plt.imshow(np.squeeze(img))
    plt.axis('off')
    plt.title('Original Image')

    plt.subplot(1, 3, 2)
    plt.imshow(np.squeeze(cv2.imread(test_df['masks_paths'].iloc[index])))
    plt.axis('off')
    plt.title('Original Mask')

    plt.subplot(1, 3, 3)
    plt.imshow(np.squeeze(predicted_img) > 0.5 )
    plt.title('Prediction')
    plt.axis('off')

    plt.show()
```

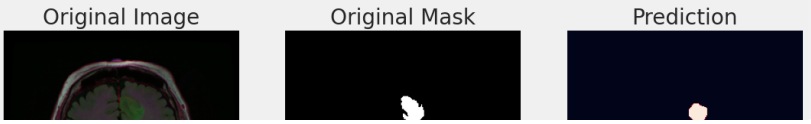
```
1/1 [==============================] - 2s 2s/step
```


```
1/1 [==============================] - 0s 22ms/step
```


```
1/1 [==============================] - 0s 31ms/step
```


```
1/1 [==============================] - 0s 21ms/step
```


```
1/1 [==============================] - 0s 23ms/step
```



Setup environment

```
!python -c "import monai" || pip install -q "monai-weekly[nibabel]"
!python -c "import matplotlib" || pip install -q matplotlib
%matplotlib inline
```

Setup imports

```
import os
import json
import shutil
import tempfile
import time

import matplotlib.pyplot as plt
import numpy as np
import nibabel as nib

from monai.losses import DiceLoss
from monai.inferers import sliding_window_inference
from monai import transforms
from monai.transforms import (
    AsDiscrete,
    Activations,
)

from monai.config import print_config
from monai.metrics import DiceMetric
from monai.utils.enums import MetricReduction
from monai.networks.nets import SwinUNETR
from monai import data
from monai.data import decollate_batch
from functools import partial

import torch


print_config()
```

```
MONAI version: 0.9.0rc3
Numpy version: 1.22.2
Pytorch version: 1.10.0a0+0aef44c
MONAI flags: HAS_EXT = False, USE_COMPILED = False
MONAI rev id: a20ff4ebc84776b29744fc7a3ba177f724442f05
MONAI __file__: /opt/conda/lib/python3.8/site-packages/monai/__init__.py

Optional dependencies:
Pytorch Ignite version: 0.4.8
Nibabel version: 3.1.1
scikit-image version: 0.19.1
Pillow version: 9.0.1
Tensorboard version: 2.6.0
gdown version: 4.2.1
TorchVision version: 0.11.0a0
tqdm version: 4.62.3
lmdb version: 1.2.1
psutil version: 5.8.0
pandas version: 1.4.1
einops version: 0.3.2
transformers version: 4.16.2
mlflow version: 1.23.1
pynrrd version: NOT INSTALLED or UNKNOWN VERSION.

For details about installing the optional dependencies, please visit:
    https://docs.monai.io/en/latest/installation.html#installing-the-recommended-dependencies
```

Setup data directory

```
directory = os.environ.get("MONAI_DATA_DIRECTORY")
root_dir = tempfile.mkdtemp() if directory is None else directory
print(root_dir)
```

Setup average meter, fold reader, checkpoint saver

```
class AverageMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = np.where(self.count > 0, self.sum / self.count, self.sum)


def datafold_read(datalist, basedir, fold=0, key="training"):
    with open(datalist) as f:
        json_data = json.load(f)

    json_data = json_data[key]

    for d in json_data:
        for k in d:
            if isinstance(d[k], list):
                d[k] = [os.path.join(basedir, iv) for iv in d[k]]
            elif isinstance(d[k], str):
                d[k] = os.path.join(basedir, d[k]) if len(d[k]) > 0 else d[k]

    tr = []
    val = []
    for d in json_data:
        if "fold" in d and d["fold"] == fold:
            val.append(d)
        else:
            tr.append(d)

    return tr, val


def save_checkpoint(model, epoch, filename="model.pt", best_acc=0, dir_add=root_dir):
    state_dict = model.state_dict()
    save_dict = {"epoch": epoch, "best_acc": best_acc, "state_dict": state_dict}
    filename = os.path.join(dir_add, filename)
    torch.save(save_dict, filename)
    print("Saving checkpoint", filename)
```

Setup dataloader

```
def get_loader(batch_size, data_dir, json_list, fold, roi):
    data_dir = data_dir
    datalist_json = json_list
    train_files, validation_files = datafold_read(datalist=datalist_json, basedir=data_dir, fold=fold)
    train_transform = transforms.Compose(
        [
            transforms.LoadImaged(keys=["image", "label"]),
            transforms.ConvertToMultiChannelBasedOnBratsClassesd(keys="label"),
            transforms.CropForegroundd(
                keys=["image", "label"],
                source_key="image",
                k_divisible=[roi[0], roi[1], roi[2]],
            ),
            transforms.RandSpatialCropd(
                keys=["image", "label"],
                roi_size=[roi[0], roi[1], roi[2]],
                random_size=False,
            ),
            transforms.RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=0),
            transforms.RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=1),
            transforms.RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=2),
            transforms.NormalizeIntensityd(keys="image", nonzero=True, channel_wise=True),
            transforms.RandScaleIntensityd(keys="image", factors=0.1, prob=1.0),
            transforms.RandShiftIntensityd(keys="image", offsets=0.1, prob=1.0),
        ]
    )
    val_transform = transforms.Compose(
        [
```

```
        transforms.LoadImaged(keys=["image", "label"]),
        transforms.ConvertToMultiChannelBasedOnBratsClassesd(keys="label"),
        transforms.NormalizeIntensityd(keys="image", nonzero=True, channel_wise=True),
    ]
)

train_ds = data.Dataset(data=train_files, transform=train_transform)

train_loader = data.DataLoader(
    train_ds,
    batch_size=batch_size,
    shuffle=True,
    num_workers=8,
    pin_memory=True,
)
val_ds = data.Dataset(data=validation_files, transform=val_transform)
val_loader = data.DataLoader(
    val_ds,
    batch_size=1,
    shuffle=False,
    num_workers=8,
    pin_memory=True,
)

return train_loader, val_loader
```

Set dataset root directory and hyper-parameters

```
from google.colab import drive
drive.mount('/content/drive')
```

```
data_dir = '/content/drive/MyDrive/RSNA_ASNR_MICCAI_BraTS2021_TrainingData_16July2021/TrainingData'
json_list = "/content/brats21_folds.json"
roi = (128, 128, 128)
batch_size = 2
sw_batch_size = 4
fold = 1
infer_overlap = 0.5
max_epochs = 100
val_every = 10
train_loader, val_loader = get_loader(batch_size, data_dir, json_list, fold, roi)
```

Check data shape and visualize

```
img_add = os.path.join(data_dir, "BraTS2021_00002/BraTS2021_00002_flair.nii.gz")
label_add = os.path.join(data_dir, "BraTS2021_00002/BraTS2021_00002_seg.nii.gz")
img = nib.load(img_add).get_fdata()
label = nib.load(label_add).get_fdata()
print(f"image shape: {img.shape}, label shape: {label.shape}")
plt.figure("image", (18, 6))
plt.subplot(1, 2, 1)
plt.title("image")
plt.imshow(img[:, :, 78], cmap="gray")
plt.subplot(1, 2, 2)
plt.title("label")
plt.imshow(label[:, :, 78])
plt.show()
```

```
image shape: (240, 240, 155), label shape: (240, 240, 155)
```

Create Swin UNETR model

```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = SwinUNETR(
    img_size=roi,
    in_channels=4,
    out_channels=3,
    feature_size=48,
    drop_rate=0.0,
    attn_drop_rate=0.0,
    dropout_path_rate=0.0,
    use_checkpoint=True,
).to(device)
```

Optimizer and loss function

```
torch.backends.cudnn.benchmark = True
dice_loss = DiceLoss(to_onehot_y=False, sigmoid=True)
post_sigmoid = Activations(sigmoid=True)
post_pred = AsDiscrete(argmax=False, threshold=0.5)
dice_acc = DiceMetric(include_background=True, reduction=MetricReduction.MEAN_BATCH, get_not_nans=True)
model_inferer = partial(
    sliding_window_inference,
    roi_size=[roi[0], roi[1], roi[2]],
    sw_batch_size=sw_batch_size,
    predictor=model,
    overlap=infer_overlap,
)

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-5)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=max_epochs)
```

Define Train and Validation Epoch

```
def train_epoch(model, loader, optimizer, epoch, loss_func):
    model.train()
    start_time = time.time()
    run_loss = AverageMeter()
    for idx, batch_data in enumerate(loader):
        data, target = batch_data["image"].to(device), batch_data["label"].to(device)
        logits = model(data)
        loss = loss_func(logits, target)
        loss.backward()
        optimizer.step()
        run_loss.update(loss.item(), n=batch_size)
        print(
            "Epoch {}/{} {}/{}".format(epoch, max_epochs, idx, len(loader)),
            "loss: {:.4f}".format(run_loss.avg),
            "time {:.2f}s".format(time.time() - start_time),
        )
        start_time = time.time()
    return run_loss.avg


def val_epoch(
    model,
    loader,
    epoch,
    acc_func,
    model_inferer=None,
    post_sigmoid=None,
    post_pred=None,
):
    model.eval()
    start_time = time.time()
    run_acc = AverageMeter()

    with torch.no_grad():
        for idx, batch_data in enumerate(loader):
```

```
            data, target = batch_data["image"].to(device), batch_data["label"].to(device)
            logits = model_inferer(data)
            val_labels_list = decollate_batch(target)
            val_outputs_list = decollate_batch(logits)
            val_output_convert = [post_pred(post_sigmoid(val_pred_tensor)) for val_pred_tensor in val_outputs_list]
            acc_func.reset()
            acc_func(y_pred=val_output_convert, y=val_labels_list)
            acc, not_nans = acc_func.aggregate()
            run_acc.update(acc.cpu().numpy(), n=not_nans.cpu().numpy())
            dice_tc = run_acc.avg[0]
            dice_wt = run_acc.avg[1]
            dice_et = run_acc.avg[2]
            print(
                "Val {}/{} {}/{}".format(epoch, max_epochs, idx, len(loader)),
                ", dice_tc:",
                dice_tc,
                ", dice_wt:",
                dice_wt,
                ", dice_et:",
                dice_et,
                ", time {:.2f}s".format(time.time() - start_time),
            )
            start_time = time.time()

    return run_acc.avg
```

Define Trainer

```
def trainer(
    model,
    train_loader,
    val_loader,
    optimizer,
    loss_func,
    acc_func,
    scheduler,
    model_inferer=None,
    start_epoch=0,
    post_sigmoid=None,
    post_pred=None,
):
    val_acc_max = 0.0
    dices_tc = []
    dices_wt = []
    dices_et = []
    dices_avg = []
    loss_epochs = []
    trains_epoch = []
    for epoch in range(start_epoch, max_epochs):
        print(time.ctime(), "Epoch:", epoch)
        epoch_time = time.time()
        train_loss = train_epoch(
            model,
            train_loader,
            optimizer,
            epoch=epoch,
            loss_func=loss_func,
        )
        print(
            "Final training  {}/{}".format(epoch, max_epochs - 1),
            "loss: {:.4f}".format(train_loss),
            "time {:.2f}s".format(time.time() - epoch_time),
        )

        if (epoch + 1) % val_every == 0 or epoch == 0:
            loss_epochs.append(train_loss)
            trains_epoch.append(int(epoch))
            epoch_time = time.time()
            val_acc = val_epoch(
                model,
                val_loader,
                epoch=epoch,
                acc_func=acc_func,
                model_inferer=model_inferer,
                post_sigmoid=post_sigmoid,
                post_pred=post_pred,
            )
            dice_tc = val_acc[0]
            dice_wt = val_acc[1]
```

```
            dice_et = val_acc[2]
            val_avg_acc = np.mean(val_acc)
            print(
                "Final validation stats {}/{}".format(epoch, max_epochs - 1),
                ", dice_tc:",
                dice_tc,
                ", dice_wt:",
                dice_wt,
                ", dice_et:",
                dice_et,
                ", Dice_Avg:",
                val_avg_acc,
                ", time {:.2f}s".format(time.time() - epoch_time),
            )
            dices_tc.append(dice_tc)
            dices_wt.append(dice_wt)
            dices_et.append(dice_et)
            dices_avg.append(val_avg_acc)
            if val_avg_acc > val_acc_max:
                print("new best ({:.6f} --> {:.6f}). ".format(val_acc_max, val_avg_acc))
                val_acc_max = val_avg_acc
                save_checkpoint(
                    model,
                    epoch,
                    best_acc=val_acc_max,
                )
            scheduler.step()
    print("Training Finished !, Best Accuracy: ", val_acc_max)
    return (
        val_acc_max,
        dices_tc,
        dices_wt,
        dices_et,
        dices_avg,
        loss_epochs,
        trains_epoch,
    )
```

Execute training

```
start_epoch = 0

(
    val_acc_max,
    dices_tc,
    dices_wt,
    dices_et,
    dices_avg,
    loss_epochs,
    trains_epoch,
) = trainer(
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    optimizer=optimizer,
    loss_func=dice_loss,
    acc_func=dice_acc,
    scheduler=scheduler,
    model_inferer=model_inferer,
    start_epoch=start_epoch,
    post_sigmoid=post_sigmoid,
    post_pred=post_pred,
)
```

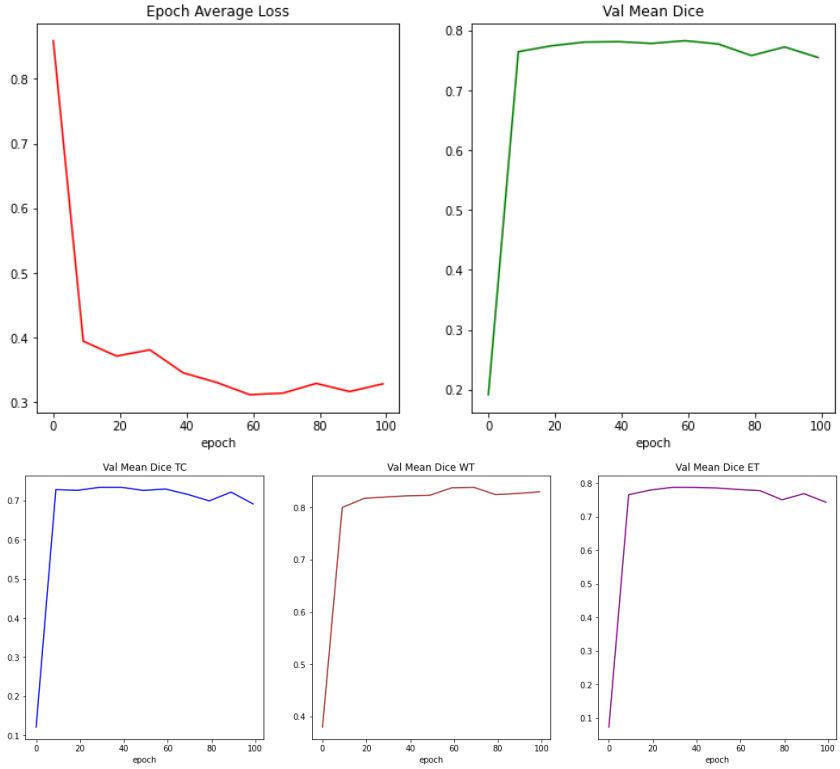
```
print(f"train completed, best average dice: {val_acc_max:.4f} ")
```

```
     train completed, best average dice: 0.7828
```

Plot the loss and Dice metric

```
plt.figure("train", (12, 6))
plt.subplot(1, 2, 1)
plt.title("Epoch Average Loss")
plt.xlabel("epoch")
plt.plot(trains_epoch, loss_epochs, color="red")
plt.subplot(1, 2, 2)
plt.title("Val Mean Dice")
```

```
plt.xlabel("epoch")
plt.plot(trains_epoch, dices_avg, color="green")
plt.show()
plt.figure("train", (18, 6))
plt.subplot(1, 3, 1)
plt.title("Val Mean Dice TC")
plt.xlabel("epoch")
plt.plot(trains_epoch, dices_tc, color="blue")
plt.subplot(1, 3, 2)
plt.title("Val Mean Dice WT")
plt.xlabel("epoch")
plt.plot(trains_epoch, dices_wt, color="brown")
plt.subplot(1, 3, 3)
plt.title("Val Mean Dice ET")
plt.xlabel("epoch")
plt.plot(trains_epoch, dices_et, color="purple")
plt.show()
```



Create test set dataloader

```
case_num = "01619"

test_files = [
    {
        "image": [
            os.path.join(
                data_dir,
                "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_flair.nii.gz",
            ),
            os.path.join(
                data_dir,
                "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_t1ce.nii.gz",
            ),
            os.path.join(
                data_dir,
                "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_t1.nii.gz",
            ),
            os.path.join(
```

```
                data_dir,
                "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_t2.nii.gz",
            ),
        ],
        "label": os.path.join(
            data_dir,
            "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_seg.nii.gz",
        ),
    }
]

test_transform = transforms.Compose(
    [
        transforms.LoadImaged(keys=["image", "label"]),
        transforms.ConvertToMultiChannelBasedOnBratsClassesd(keys="label"),
        transforms.NormalizeIntensityd(keys="image", nonzero=True, channel_wise=True),
    ]
)

test_ds = data.Dataset(data=test_files, transform=test_transform)

test_loader = data.DataLoader(
    test_ds,
    batch_size=1,
    shuffle=False,
    num_workers=8,
    pin_memory=True,
)
```

Load the best saved checkpoint and perform inference

```
model.load_state_dict(torch.load(os.path.join(root_dir, "model.pt"))["state_dict"])
model.to(device)
model.eval()

model_inferer_test = partial(
    sliding_window_inference,
    roi_size=[roi[0], roi[1], roi[2]],
    sw_batch_size=1,
    predictor=model,
    overlap=0.6,
)


with torch.no_grad():
    for batch_data in test_loader:
        image = batch_data["image"].cuda()
        prob = torch.sigmoid(model_inferer_test(image))
        seg = prob[0].detach().cpu().numpy()
        seg = (seg > 0.5).astype(np.int8)
        seg_out = np.zeros((seg.shape[1], seg.shape[2], seg.shape[3]))
        seg_out[seg[1] == 1] = 2
        seg_out[seg[0] == 1] = 1
        seg_out[seg[2] == 1] = 4
```

Visualize segmentation output and compare with label

```
slice_num = 67
img_add = os.path.join(
    data_dir,
    "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_t1ce.nii.gz",
)
label_add = os.path.join(
    data_dir,
    "TrainingData/BraTS2021_" + case_num + "/BraTS2021_" + case_num + "_seg.nii.gz",
)
img = nib.load(img_add).get_fdata()
label = nib.load(label_add).get_fdata()
plt.figure("image", (18, 6))
plt.subplot(1, 3, 1)
plt.title("image")
plt.imshow(img[:, :, slice_num], cmap="gray")
plt.subplot(1, 3, 2)
plt.title("label")
plt.imshow(label[:, :, slice_num])
plt.subplot(1, 3, 3)
plt.title("segmentation")
```

```
plt.imshow(seg_out[:, :, slice_num])
plt.show()
```