Task 4

Task 4

Data Handling and Preprocessing (10 Points)

a. You can focus for now on loading the T1-weighted images and the matching labels.

We used the `LoadImaged` transform from MONAI to load multiple modalities (`t1`, `t1ce`, `t2`, `flair`) and corresponding labels. The images are concatenated into a single tensor using the `ConcatItemsd` transform, and the labels are remapped using `RemapLabels` to ensure the labels are correctly handled.

b. Create a dataloader for the data using PyTorch's Dataloader (or Monai's Dataloader class).

The dataloader is created using the `get_dataloader` function. This function: - Reads the split files (e.g., `train.txt`, `validation.txt`, `test.txt`) containing dataset paths. - Applies a transformation pipeline (`get_transforms`) that preprocesses and augments the data. - Uses MONAI's `CacheDataset` for efficient loading and preprocessing. - Wraps the dataset in PyTorch's `DataLoader` to iterate over batches during training and validation.

c. Create suitable augmentations for the task to solve. Please note: If you apply transformations to the input data, you should think about if you need to apply any transformation to the label of the image as well.

Augmentations are applied in `get_transforms`: - `RandFlipd` flips the images and labels along different axes. - `RandSpatialCropd` crops the images and labels to the specified `roi_size`. The transformations account for labels by applying the same operations to both images and labels.

the code applies several augmentations in the `get_transforms` function: - **Spatial augmentations**: - `RandFlipd` flips the image and label along all three spatial axes with a probability of 0.5. - `RandSpatialCropd` crops both the images and labels to a predefined `roi_size`. - **Normalization**: - `NormalizeIntensityd` normalizes the image intensities channel-wise.

2. Data Splitting (5 Points)

a. Think about an appropriate split of the dataset into train, validation and test split. Briefly explain why it is good scientific practice to have separate Training, Validation and Test dataset?

Separating these datasets prevents overfitting and ensures that: - **Training dataset** is used for learning. - **Validation dataset** is used for tuning hyperparameters and evaluating performance during training. - **Test dataset** is used for final unbiased evaluation of the model's performance.

b. Please define the patient ids for each split in a separate file, like test.txt or validation.yml

Patient IDs are loaded from JSON files (`train.txt`, `validation.txt`, `test.txt`) in the `split_dir`. This ensures reproducibility and consistency in data splits.

3. Model Selection and Training (20 Points)

a. Implement a training pipeline to train the U-Net model of MONAI (or your own implementation if you want). Make use of the dataloader you created.

The training pipeline is implemented in the `train()` function: - The MONAI `UNet` is initialized with: - `spatial_dims=3` (to handle 3D data). - The number of input channels matches the modalities (`t1`, `t1ce`, `t2`, `flair`), and the number of output channels corresponds to the number of segmentation classes. - Customizable encoder-decoder channels and strides. - The pipeline uses the dataloaders (`train_loader` and `val_loader`) generated from the `get_dataloaders()` function, which provides training and validation datasets with proper transformations and augmentations. - The optimizer is `Adam`, and the loss function is defined as the `DiceLoss`.

b. At the end of each epoch, you should run a validation of the models performance.

Yes, the pipeline includes a validation step at the end of each epoch: - The validation data is processed through the model using `sliding_window_inference` to handle 3D volumes. - The Dice metric is calculated using the MONAI `DiceMetric` class to assess the overlap between predictions and ground truth. - Validation metrics, including the Dice score for each class and the mean Dice score, are logged for monitoring performance.

c. As we do not want to waste unnecessary energie and time, please implement an early stopping condition, that stops the training when the performance of the model does not improve for a few epochs on the validation set.

Yes, early stopping is implemented: - The `train()` function monitors the mean Dice score on the validation set. - If the Dice score does not improve for a defined number of epochs (default: `10`, controlled by `config.early_stop_limit`), training is stopped early to save time and computational resources.

d. Which loss function do you think suitable to train your model for this segmentation task? The `DiceLoss` is used for training, as it is well-suited for segmentation tasks. This loss directly optimizes the Dice score, which is a common metric for medical image segmentation. It ensures accurate overlap between the predicted and ground truth masks.

e. Please save the weights of the model that you trained. You can either save the weights after each epoch seperatly or you can override the previously saved model with the new best performing one after each validation.

The model weights are saved during training: - After each epoch, if the validation mean Dice score improves, the current model's weights are saved as `best_model.pth` in the specified `config.save_path` directory. - Only the best-performing model is saved, ensuring the most optimal model is retained without using unnecessary storage for suboptimal models.

4. Testing and Performance Evaluation (15 Points)

a. Think about a suitable metric to evaluate the performance of your model on the test set. You can also use more than one metric to capture different aspects of the models performance. The suitable metric for evaluating the model's performance on the test set is the **Dice score**, as it measures the overlap between predicted segmentation masks and ground truth labels. This metric is commonly used in medical image segmentation tasks.

b. Briefly explain, which model should you evaluate? You get a trained model after each epoch? Should you evaluate all of them and pick the best performing one?

- The best-performing model is saved during training (`best_model.pth`) based on the highest mean Dice score on the validation set.
- This saved model should be evaluated on the test set, as it represents the model with the best generalization during training and validation.
- Evaluating all models from every epoch would be unnecessary and computationally expensive, as the best model has already been identified using the validation set.

c. Code a test pipeline to evaluate your model's performance.

5. Extension to New Modalities (5 Bonus Points)

a. Extent the model to include multiple modalities of your choice as input (T2, T1ce, …). Justify your choice. The modalities are combined into a single tensor using `ConcatItemsd` in the `get_transforms` function.

- Different MRI modalities capture complementary information about the tumor:
  - **T1** and **T1ce** highlight the tumor core and enhancing regions.
  - **T2** and **FLAIR** are useful for visualizing edema and non-enhancing tumor areas.
- Combining modalities allows the model to leverage richer information, leading to improved segmentation performance and better differentiation between tumor sub-regions.

b. Train the model with the same data split and asses the performance of the model using your test pipeline. The training pipeline is designed to support multiple modalities as input:

The dataloader loads all modalities (t1, t1ce, t2, flair), and the concatenated tensor is passed to the UNet model for training. Using the same data split, the performance of the model can be assessed by running the `test_model()` function on the test set after training. The test pipeline evaluates the model's performance using metrics such as the Dice score.

c. Why and how does the performance change now that you use multiple modalities compared to using a single T1-weighted image as input?

Each modality highlights different tumor characteristics. For example: - **T1ce** captures enhancing regions. - **T2** captures edema and fluid buildup. - **FLAIR** is effective for visualizing non-enhancing tumor areas. - Using multiple modalities provides the model with diverse and complementary information, enabling it to make more accurate segmentation predictions.