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

We didnt used Monai's Dataloader class, because we wanted to use the data in the HPC. The dataset implementation is availabel in the `task4/brats_segmentations/dataloader.py`

`file name: ../task4/brats_segmentation/dataloader.py`

```
class RobustZScoreNormalization(MapTransform):
    """
    Apply robust Z-score normalization to the image volumes.
    """
    def __call__(self, data):
        d = dict(data)
        for key in self.key_iterator(d):
            mask = d[key] > 0  # Only normalize within the brain mask

            lower = np.percentile(d[key][mask], 0.2)
            upper = np.percentile(d[key][mask], 99.8)

            d[key][mask & (d[key] < lower)] = lower
            d[key][mask & (d[key] > upper)] = upper
```

```
            y = d[key][mask]
            d[key] -= y.mean()
            d[key] /= y.std()

        return d


# Custom transform to remap labels
class RemapLabels(MapTransform):
    """
    Remap labels to sequential class indices.
    Original labels: 0, 1, 2, 4
    Remapped labels: 0, 1, 2, 3
    """
    def __call__(self, data):
        d = dict(data)
        label = d["label"]
        label[label == 4] = 3  # Map class 4 to 3
        d["label"] = label.long()
        return d


# Preprocessing and augmentation
def get_transforms(roi_size, augment=True):
    """
    Generate transformation pipelines for training, validation, and testing.

    Args:
        roi_size (tuple): Size of the region of interest for cropping.
        augment (bool): Whether to apply augmentations (True for training).

    Returns:
        Compose: Transformation pipeline.
    """
    transforms_list = [
        # Load and standardize image and label data
        LoadImaged(keys=["t1", "t1ce", "t2", "flair", "label"]),
        EnsureChannelFirstd(keys=["t1", "t1ce", "t2", "flair", "label"]),
        ConcatItemsd(keys=["t1", "t1ce", "t2", "flair"], name="image"),
        Orientationd(keys=["image", "label"], axcodes="RAS"),
        Spacingd(keys=["image", "label"], pixdim=(1.0, 1.0, 1.0), mode=("bilinear", "nearest"
        RobustZScoreNormalization(keys=["image"]),  # Apply robust Z-score normalization
```

```
        RemapLabels(keys=["label"]),  # Add label remapping
    ]

    if augment:
        # Augmentations for training data
        transforms_list.extend([
            RandSpatialCropSamplesd(
                keys=["image", "label"],
                roi_size=roi_size,
                random_center=True,
                random_size=False,
                num_samples=2  # Multiple crops per image
            ),
            RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=0),
            RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=1),
            RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=2),
            RandRotate90d(keys=["image", "label"], prob=0.5, max_k=3),
            RandGaussianNoised(keys=["image"], prob=0.2, mean=0.0, std=0.1),
            RandScaleIntensityd(keys=["image"], factors=0.1, prob=1.0),
            RandShiftIntensityd(keys=["image"], offsets=0.1, prob=1.0),
        ])

    # Add tensor conversion
    transforms_list.append(ToTensord(keys=["image", "label"]))

    return Compose(transforms_list)


# DataLoader setup
def get_dataloaders(split_dir, roi_size, batch_size, num_workers=4):
    """
    Create data loaders for training, validation, and testing.

    Args:
        split_dir (str): Path to the directory containing split JSON files.
        roi_size (tuple): Size of the region of interest for cropping.
        batch_size (int): Batch size for data loaders.
        num_workers (int): Number of workers for data loading.

    Returns:
        tuple: Training, validation, and test DataLoaders.
    """
```

```
    # Generate transforms for train and validation/test
    train_transform = get_transforms(roi_size, augment=True)
    val_transform = get_transforms(roi_size, augment=False)

    # Load dataset splits
    with open(f"{split_dir}/train.txt", "r") as f:
        train_files = json.load(f)
    with open(f"{split_dir}/validation.txt", "r") as f:
        val_files = json.load(f)
    with open(f"{split_dir}/test.txt", "r") as f:
        test_files = json.load(f)

    # Create datasets
    train_ds = Dataset(data=train_files, transform=train_transform)
    val_ds = Dataset(data=val_files, transform=val_transform)
    test_ds = Dataset(data=test_files, transform=val_transform)

    # Create data loaders
    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=num_
    val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False, num_workers=num_wo
    test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False, num_workers=num_

    return train_loader, val_loader, test_loader


if __name__ == "__main__":
    # Example usage
    split_dir = "./splits/split1"
    roi_size = (128, 128, 128)
    batch_size = 2
    num_workers = 4

    train_loader, val_loader, test_loader = get_dataloaders(split_dir, roi_size, batch_size,

    # Inspect one batch from the training DataLoader
    print("Testing the training DataLoader...")
    for batch in train_loader:
        print(f"Image shape: {batch['image'].shape}")
        print(f"Label shape: {batch['label'].shape}")
        print(f"Label unique values: {torch.unique(batch['label'])}")
        break
```