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 ConvertToMultiChannelBasedOnBratsClassesd(MapTransform):
    def __call__(self, data):
        d = dict(data)
        for key in self.keys:
            if key != "label":
                continue
            result = []
            # Tumor Core (TC): Combine label 1 and
            tc = torch.logical_or(d[key] == 1, d[key] == 4)
            result.append(tc)

            # Whole Tumor (WT): Combine label 1, label 2, and label 4
            wt = torch.logical_or(
                torch.logical_or(d[key] == 2, d[key] == 4), d[key] == 1
            )
            result.append(wt)

            # Enhancing Tumor (ET): Only label 4
            et = d[key] == 4
            result.append(et)

            # Stack binary masks into multi-channel format
            d[key] = torch.stack(result, dim=0).float().squeeze(1)
        return d


def get_transforms(roi_size, augment=True):
    """
    Generate transforms for data preprocessing and augmentation.

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

    Returns:
        Compose: Transformation pipeline.
    """
    images_types = ["t1", "t1ce", "t2", "flair"]
    transforms = [
        LoadImaged(keys=images_types + ["label"]),
        EnsureChannelFirstd(keys=images_types + ["label"]),
        ConcatItemsd(keys=images_types, name="image"),
        DeleteItemsd(keys=images_types),
        Orientationd(keys=["image", "label"], axcodes="RAS"),
        Spacingd(
            keys=["image", "label"],
            pixdim=(1.0, 1.0, 1.0),
            mode=("bilinear", "nearest"),
        ),
        NormalizeIntensityd(keys="image", nonzero=True, channel_wise=True),
        # ConvertToMultiChannelBasedOnBratsClassesd(
        #     keys=["label"]
        # ),  # One-hot encode labels
        AsDiscreted(keys="label", to_onehot=5),
        RandSpatialCropd(keys=["image", "label"], roi_size=roi_size, random_size=False),
    ]
    if augment:
        transforms.extend(
            [
                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),
            ]
        )

    transforms.append(ToTensord(keys=["image", "label"]))
    return Compose(transforms)


def get_dataloader(
    split_dir: str | os.PathLike,
    name: Literal["train", "test", "validation"],
    roi_size: tuple,
    batch_size: int = 2,
    num_workers: int = 0,
    cache_rate: float = 0.5,
):
    """
    Create a PyTorch DataLoader for the BraTS dataset.

    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): Number of samples per batch.
        num_workers (int): Number of workers to use for

    Returns:
        DataLoader: PyTorch DataLoader.
    """
    if name not in ["train", "test", "validation"]:
        raise ValueError("name must be one of 'train', 'test', or 'validation'")
    split_dir = Path(split_dir)
    with open(split_dir / f"{name}.txt", "r") as f:
        files = json.load(f)
    transform = get_transforms(roi_size, augment=True if name == "train" else False)
    dataset = CacheDataset(
        data=files,
        transform=transform,
        cache_rate=0.1,
        num_workers=num_workers,
    )
    return DataLoader(
        dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers
    )


# DataLoader setup with CacheDataset
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.
    """
    return [
        get_dataloader(split_dir, name, roi_size, batch_size, num_workers)
        for name in ["train", "validation", "test"]
    ]


def make_images(images, segmentation, slice_index, label_color):
    imgs = images[..., slice_index]
    imgs = (imgs - imgs.min(axis=0)) / (imgs.max(axis=0) - imgs.min(axis=0) + 1e-5)
    seg = segmentation[..., slice_index]
    for img in imgs:

        img = np.stack([img, img, img], axis=-1)
        for index, color in enumerate(label_color, start=1):
            img[seg == index] = color

        yield img


# Visualization utility using Weights & Biases
```

```python
def visualize_samples(loader: DataLoader):
    """
    Visualize 2D slices of images and labels using Weights & Biases.

    Args:
        loader: DataLoader to fetch samples from.
    """
    print("------------")
    # label_color = [[0, 0, 1], [0, 1, 0], [1, 0, 0]]
    # label_text = ["Tumor Core", "Whole Tumor", "Enhancing"]
    label_color = [[0, 0, 1], [0, 1, 0], [1, 1, 0], [1, 0, 0]]
    label_text = [
        "Necrotic and Non-Enhancing Tumor",
        "Edema",
        "IMPOSSIBLE",
        "Enhancing Tumor",
    ]

    sample = next(iter(loader))

    # sample[imgage].shape = (batch_size, num_channels, H, W, D)
    images = sample["image"][0].numpy()  # First item in the batch

    segmentation = sample["label"]
    print(segmentation.shape)
    segmentation = torch.argmax(segmentation[0], dim=0).numpy()
    assert (
        images.shape[1:] == segmentation.shape
    ), f"Shape mismatch {images.shape=} {segmentation.shape=}"

    initial_slice = images.shape[-1] // 2

    # 3D visualization :')
    if images.shape[0] == 4:
        fig, axes = plt.subplots(2, 2, figsize=(10, 10))
        plt.subplots_adjust(bottom=0.2)
        axes = axes.flatten()
    else:
        fig, axes = plt.subplots(1, images.shape[0], figsize=(10, 10))
    plt.subplots_adjust(bottom=0.2)

    img_displays = []
    for ax, image, title in zip(
        axes,
        make_images(images, segmentation, initial_slice, label_color),
        ["T1", "T1CE", "T2", "FLAIR"],
    ):
        ax.set_title(title)
        ax.axis("off")
        img_displays.append(ax.imshow(image))

    ax_slider = plt.axes([0.2, 0.05, 0.65, 0.03])  # [left, bottom, width, height]
    slider = Slider(
        ax_slider, "Slice", 0, images.shape[-1] - 1, valinit=initial_slice, valstep=1
    )

    def update(_):
        slice_index = int(slider.val)
        for img_display, img in zip(
            img_displays, make_images(images, segmentation, slice_index, label_color)
        ):
            img_display.set_data(img)
            fig.suptitle(f"Slice {slice_index}")
        fig.canvas.draw_idle()

    cmap = ListedColormap(list(label_color))

    axes[-1].legend(
        handles=[Patch(color=cmap(i), label=text) for i, text in enumerate(label_text)],
        bbox_to_anchor=(1.05, 1),
        loc="upper left",
    )
    slider.on_changed(update)
    plt.show()


if __name__ == "__main__":
    # Example usage
    split_dir = "./splits/split3"
    # roi_size = (128, 128, 128)
    roi_size = (240, 240, 160)
    batch_size = 8
    num_workers = 1
    val_dataloader = get_dataloader(
        split_dir, "validation", roi_size, batch_size, num_workers
    )

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

    # # Visualize samples using Weights & Biases
    # print("Visualizing samples with W&B...")
    visualize_samples(val_dataloader)
```