**Full code and explanations of the MONAI segmentation pipeline**

**1. Loading image paths**

This section defines the paths to the various images used for training, validation and testing.

Files are sorted to match images and masks. If no file is found, an error is raised.

***image\_dir = "/media/biomech/FLORA/SEGMENTATION/Training Set/Original"  
seg\_dir = "/media/biomech/FLORA/SEGMENTATION/Training Set/Mask"  
output\_dir = "/media/biomech/FLORA/TEST COMPARISON/Output"  
test\_image\_dir = "/media/biomech/FLORA/TEST COMPARISON/Original"  
  
val\_image\_dir = "/media/biomech/FLORA/SEGMENTATION/Validation Set/Original"  
val\_seg\_dir = "/media/biomech/FLORA/SEGMENTATION/Validation Set/Mask"  
  
os.makedirs(output\_dir, exist\_ok=True)  
  
images = sorted(glob(os.path.join(image\_dir, "\*.tif")))  
segs = sorted(glob(os.path.join(seg\_dir, "\*.tif")))  
test\_images = sorted(glob(os.path.join(test\_image\_dir, "\*.tif")))  
val\_images = sorted(glob(os.path.join(val\_image\_dir, "\*.tif")))  
val\_segs = sorted(glob(os.path.join(val\_seg\_dir, "\*.tif")))  
  
if len(images) == 0 or len(segs) == 0:  
 raise ValueError("No images or segmentations found. Check your directories.")***

**2. Loading images and preparing files**

Utility functions for loading TIFF images as numpy arrays and adding an extra dimension.

val\_files, test\_files, val\_files\_eval lists are prepared for MONAI.

***def load\_and\_convert\_image(path):  
 with Image.open(path) as img:  
 img = np.array(img, dtype=np.float32)  
 img = np.expand\_dims(img, axis=0)  
 return img  
  
def threshold\_and\_scale(img):  
 return img  
  
val\_files = [{"img": threshold\_and\_scale(load\_and\_convert\_image(img)), "seg": load\_and\_convert\_image(seg)} for img, seg in zip(images, segs)]  
test\_files = [{"img": threshold\_and\_scale(load\_and\_convert\_image(img))} for img in test\_images]  
val\_files\_eval = [{"img": threshold\_and\_scale(load\_and\_convert\_image(img)), "seg": load\_and\_convert\_image(seg)} for img, seg in zip(val\_images, val\_segs)]***

**3. Transform of data**

Definition of data transformations for training, validation and testing. Images are converted to tensors, normalized and resized.

RandAffine is also used for data augmentation.

***train\_transforms = Compose([...])  
val\_transforms = Compose([...])  
test\_transforms = Compose([...])***

**4. Creation of Dataset and DataLoader**

Creation of MONAI Dataset and DataLoader objects for the various datasets. These objects handle batch processing and automatic pre-processing.

***val\_ds = Dataset(data=val\_files, transform=train\_transforms)  
val\_loader = DataLoader(val\_ds, batch\_size=8, num\_workers=4, collate\_fn=list\_data\_collate)  
  
test\_ds = Dataset(data=test\_files, transform=test\_transforms)  
test\_loader = DataLoader(test\_ds, batch\_size=8, num\_workers=4, collate\_fn=monai.data.list\_data\_collate,shuffle=False)  
  
val\_ds\_eval = Dataset(data=val\_files\_eval, transform=val\_transforms)  
val\_loader\_eval = DataLoader(val\_ds\_eval, batch\_size=8, num\_workers=4, collate\_fn=list\_data\_collate)***

**5. Definition of the model, loss et optimizer**

Definition of MONAI's SegResNet model, the loss function (DiceFocal + BCE), the Adam optimizer and the scheduler to modulate the learning rate.

***model = SegResNet(...).to(device)  
loss\_function = lambda pred, target: DiceFocalLoss(alpha=0.5,gamma=3)(pred, target) + nn.BCEWithLogitsLoss()(pred, target)  
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)  
scheduler = torch.optim.lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=50)***

**6. Training of the model and validation**

Model training with batch loss calculation. Visualize predictions with the `visualize\_prediction` function.

***for epoch in range(num\_epochs):  
 ...  
 for val\_batch in val\_loader\_eval:  
 val\_outputs = sliding\_window\_inference(val\_inputs, (512,512), 8, model)  
 ...***

**7. Test image inference and backup**

Inference performed on test images with `torch.no\_grad` (no gradient calculated).

Predictions are binarized with a threshold, converted to images and saved with the suffix `.mask.png`.

***with torch.no\_grad():  
 for i, test\_data in enumerate(test\_loader):  
 ...  
 Image.fromarray(binary\_output).save(os.path.join(output\_dir, new\_filename))***