**3D Slicer and MONAI project B1144257 資管三乙王奕晴**

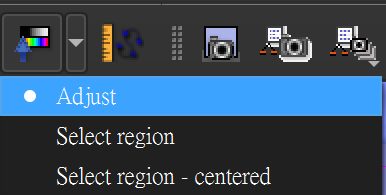
**Part one : *3D Slicer***

Introduction :

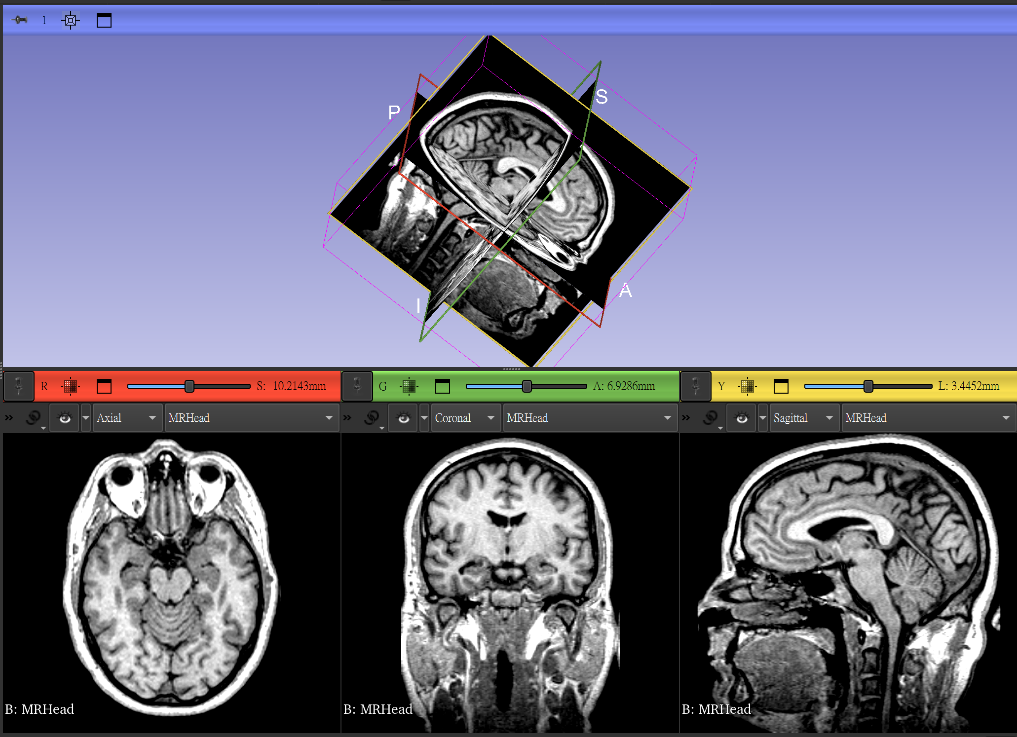
I use MR Head to complete my 3D Slicer project; this is the picture after I imported the image.

The red screen shows the axial of the head, and the green screen shows the coronal of the head, and the yellow screen shows the sagittal.

And I use Adjust to make the brain can see better.



The following picture shows the adjusted display screen



Problem and Result :

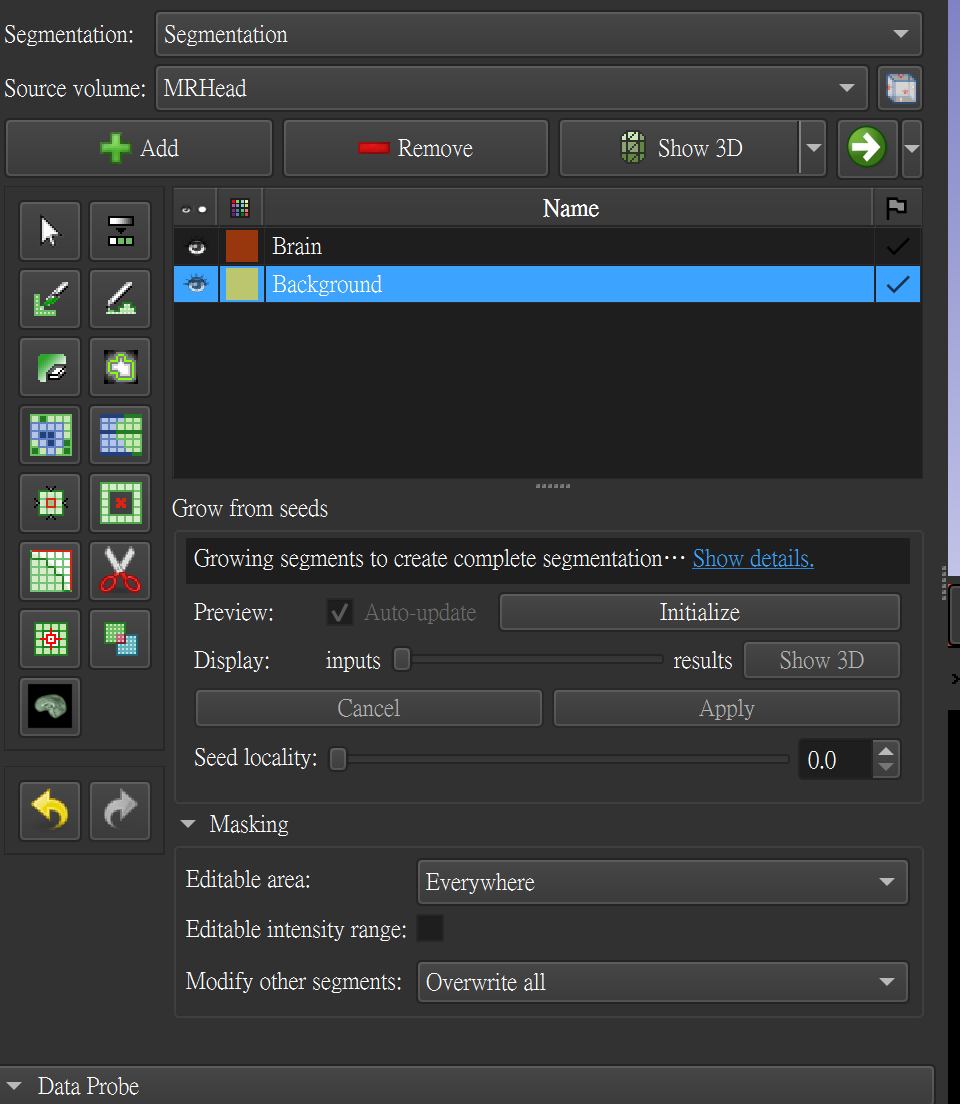
First, I’m not a medical student, so I’m not very familiar with medical knowledge, it makes me have many difficulties to encounter the process of completing the project.

Second, it was my first time using 3D Slicer, so I’m new to it. To complete this project, I asked chat GPT to fix the problem when I use 3D Slicer, I also search online for information when marking the brain.

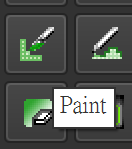
The last but not least, I successfully marked the brain area and calculated its volume, now I can finish the 3D Slicer project from my own self; to complete this project, it makes me better to use this tool, and also It also improved my English skills because of speech in English.

Process description:

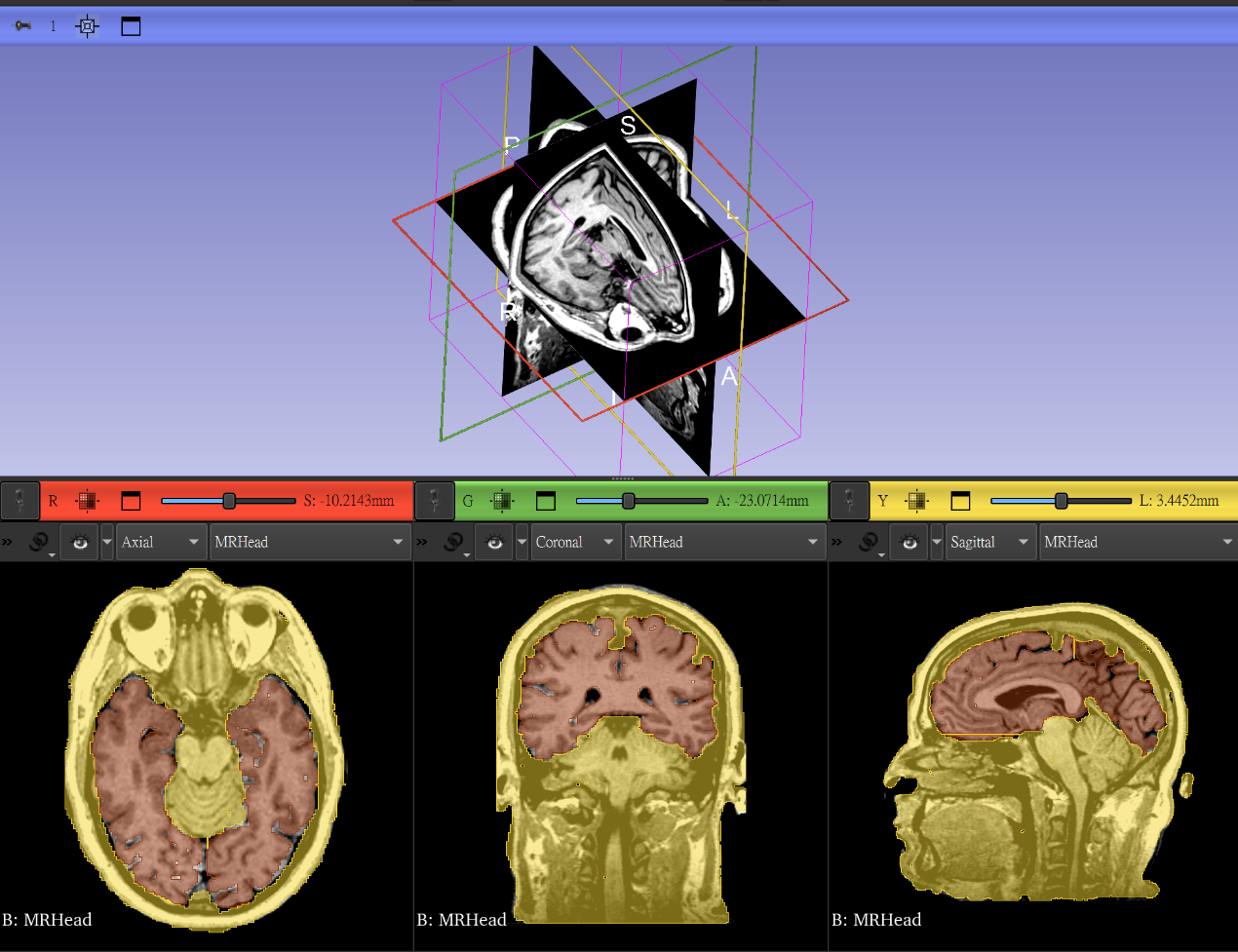
I used two segmentation, brain and background; to distinguish the main things I want to mark from other irrelevant parts.



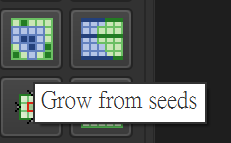
Use Paint to mark the brain side and the background side.



The red side is the brain, and the yellow side is the background.

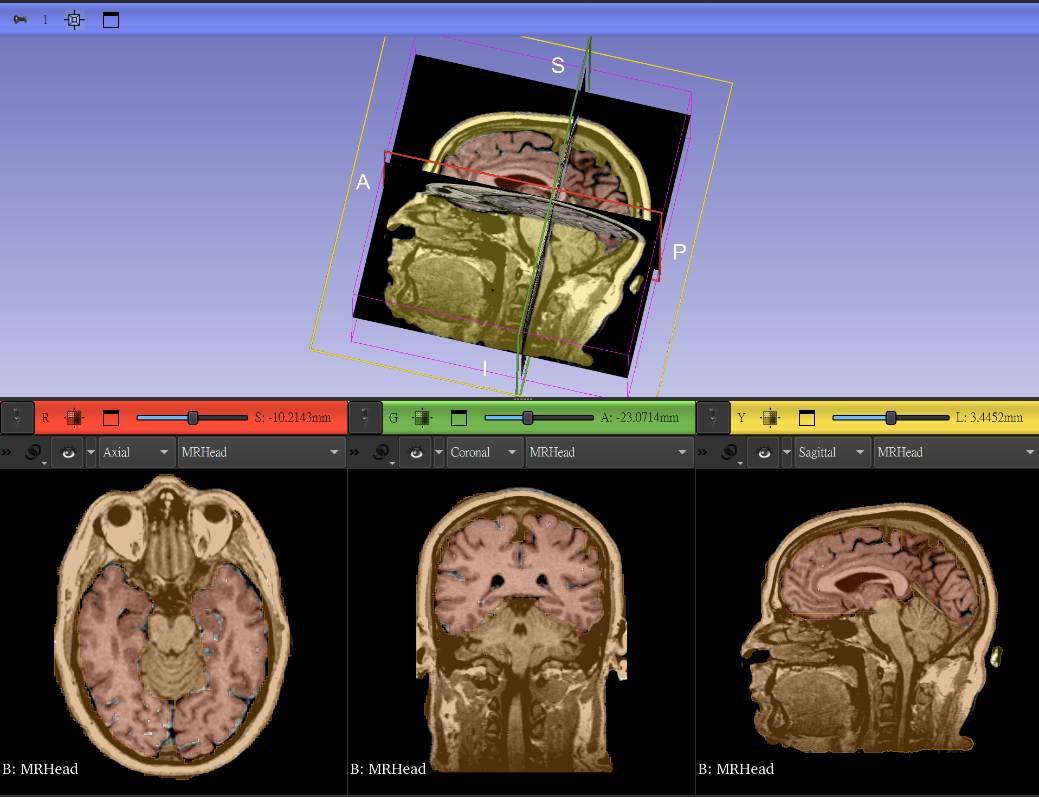


So why do I use two segmentations to mark the brain? It is because of next step I’ll use the tool named Grow from seeds, so what is grow from seeds? It is a semi-automatic segmentation tool in the segment Editor module of 3D Slicer. It is suitable for quickly separating the foreground (the structure to be marked) from the background (other surrounding tissues).



Its principle is similar to the watershed algorithm, which automatically divides the entire volume into corresponding areas based on the speed points (Seeds) with different labels that you “plant” on the image in advance.

I used two segmentations, brain and background, as I said, to distinguish between the main content I wanted to label and other irrelevant parts. Since the grow from seeds tool requires at least two segments, brain and background. By “planting” the seed points of the foreground (the structure to be retained) and background in the image, the algorithm automatically spreads them and quickly completes the segmentation.



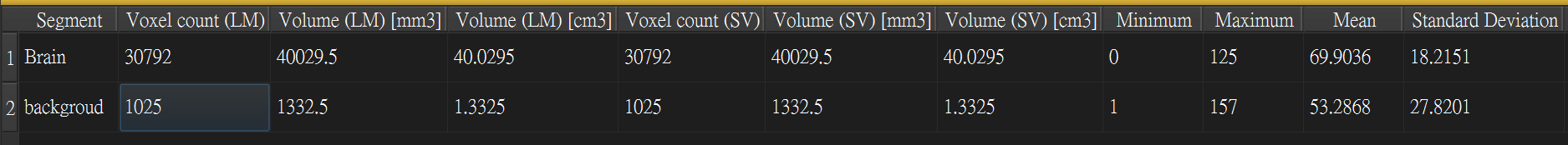
Next is the volume measurement part. I used Quantification to calculate it. The easiest way to calculate how big and how weight the brain is to look at volume (LM) or volume (CS), which means “how many milliliters.

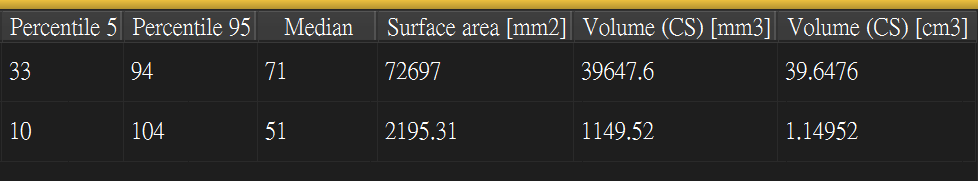


Label-Map Volume (LM) = Volume calculated by "number of blocks x block size". LM is calculated to be approximately 40.03 cm³. Closed-Surface Volume (CS) = Volume calculated from the 3D hull mesh model. CS is calculated to be approximately 39.65 cm³. Both are around 40 mL, with a difference of less than 1%.

The number is about 40 mL, which means the size of the space occupied by the brain parenchyma we segmented out.

LM and CS each have their own advantages and disadvantages, the former is fast, the latter is smooth, just choose according to your needs.





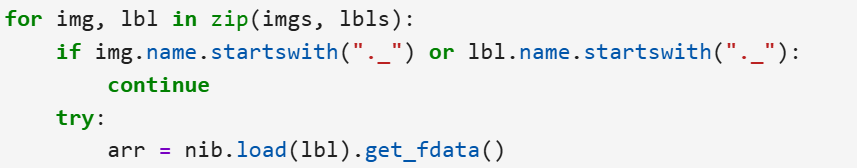
**Part two : *MONAI model***

Introduction :

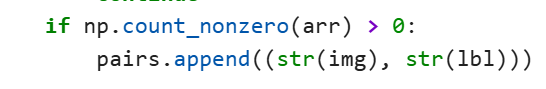
My MONAI model task is my spleen segmentation task. The dataset I used is [Task09\_Spleen from the Medical Segmentation Decathlon (MSD)], which consists of abdominal MRI scans (3D) of multiple patients, each with a corresponding spleen label. This model can be given a 3D MRI volume image, and it can output a segmentation map of the same size, accurately marking the spleen from the abdominal image.

Problem and Result :

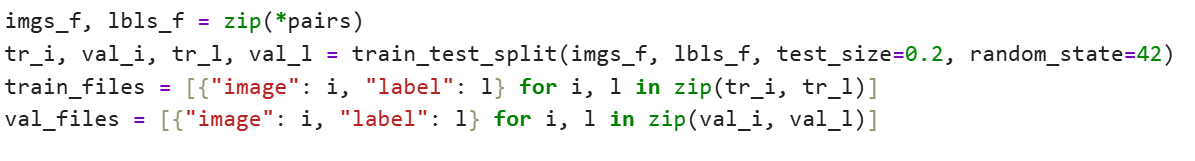
Start from data preprocessing, First I download the picture data, and then start processing my pictures, I download the image data and then start to process my images. The first step is to deal with the junk files due to the difference in file processing between Mac and Windows.



In the second step, I only kept the data with the label.



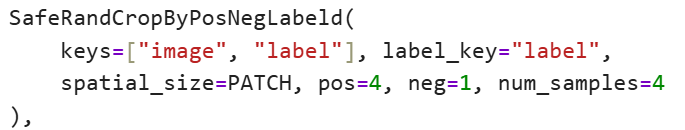
The third step is to classify the data. 80% of the data is classified into the training model data set, and 20% of the data is classified into the validation data set. The data is then packaged into a dictionary for use by MONAI DataLoader.



In the fourth step, we come to the Transform area (data preprocessing).



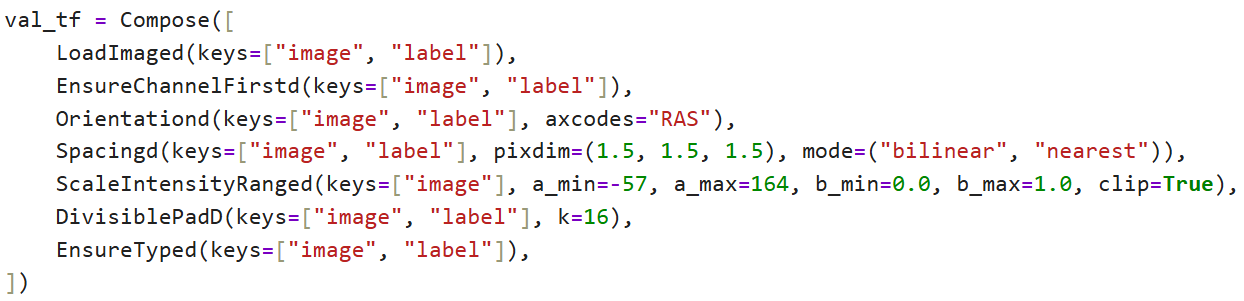
* LoadImaged(keys=["image", "label"]), ***# Read NIfTI file***
* EnsureChannelFirstd(keys=["image", "label"]), ***# becomes (C, D, H, W)***
* Orientationd(keys=["image", "label"], axcodes="RAS"), ***# Unify the direction (Right-Anterior-Superior)***
* Spacingd(keys=["image", "label"], pixdim=(1.5, 1.5, 1.5), mode=("bilinear", "nearest")), ***# Scale to uniform voxel***
* ScaleIntensityRanged(keys=["image"], a\_min=-57, a\_max=164, b\_min=0.0, b\_max=1.0, clip=True), ***# Image intensity normalization***



***# This is a random positive and negative sample crop***

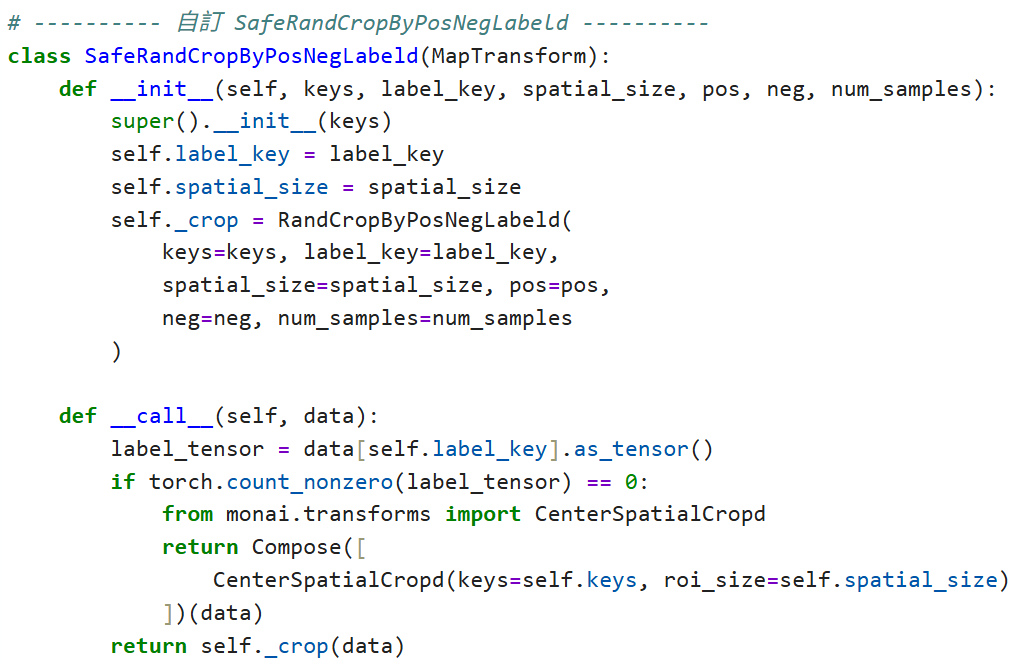
* EnsureTyped(keys=["image", "label"]), ***# Convert to tensor format***

In the fourth step, we come to the val\_tf (verification data) part.



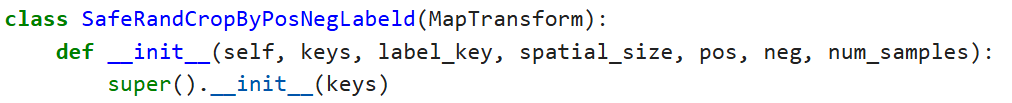
This part is similar to training, except that DivisiblePadD is used instead of SafeRandCropByPosNegLabeld to avoid cropping affecting verification.

Next, let me introduce my custom SafeRandCropByPosNegLabeld — safe foreground cropping tool

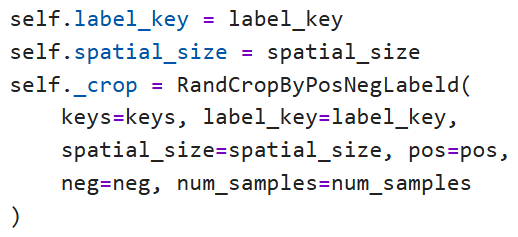


The reason for using this tool is that when the data has a foreground (spleen), use RandCropByPosNegLabeld to randomly crop.

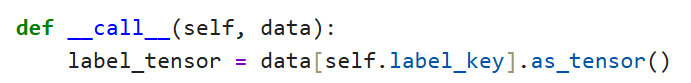
If the data is all background (label=0), use "center cropping" CenterSpatialCropd instead to avoid cropping patches that are all background, which will result in no learning.



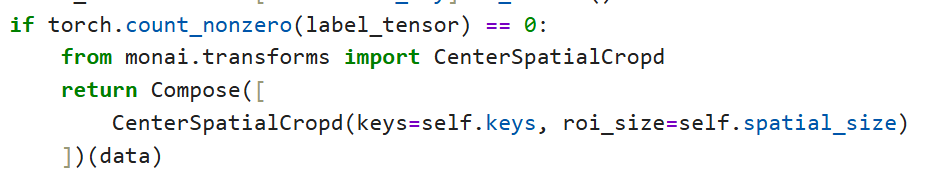
* Inherit MapTransform so that it can be used by Compose.
* keys → fields to be operated, e.g. ["image", "label"]
* label\_key → the label key, usually "label"
* spatial\_size → the size of the cropped patch (e.g., (96, 96, 96))
* pos/neg/num\_samples → Control the ratio of positive samples/negative samples and the number of patches



* Create an internal RandCropByPosNegLabeld instance, which is MONAI's native tool for cropping patches of positive and negative samples.
* Save as self.\_crop for subsequent calls.

****

* \_\_call\_\_: Allows this class to be called like a function ().
* label\_tensor: extract labels from data and convert them into tensor format.

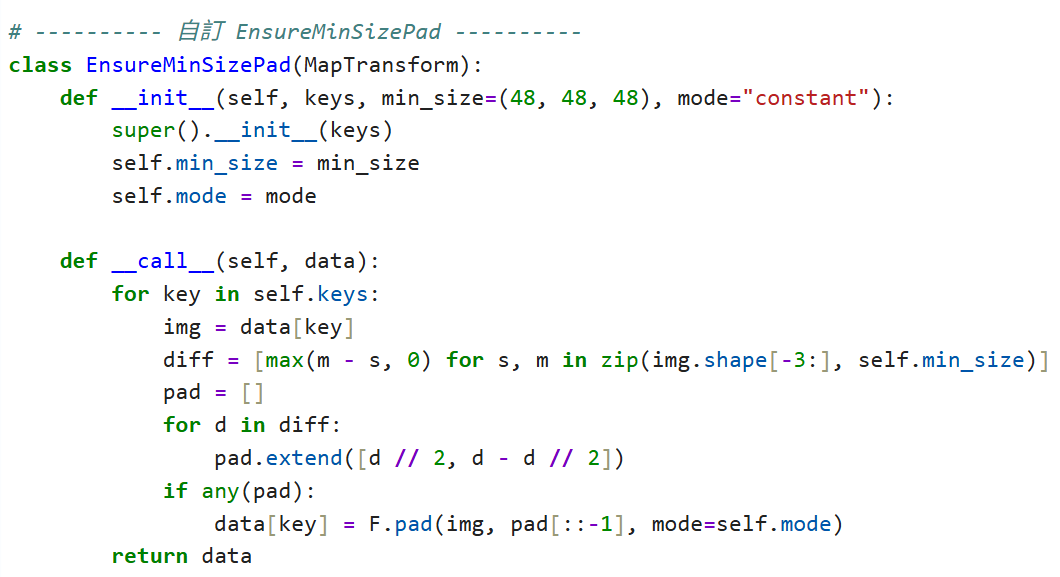
****

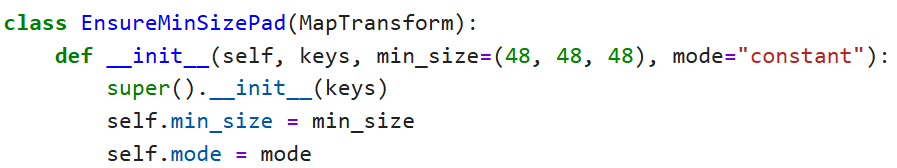
If this label has no foreground at all (all 0):

* Use CenterSpatialCropd instead.
* Make sure there is at least one relatively meaningful patch, rather than just randomly cutting out an area that is all background.
* return self.\_crop(data) # Otherwise (label has foreground), execute RandCropByPosNegLabeld to randomly crop

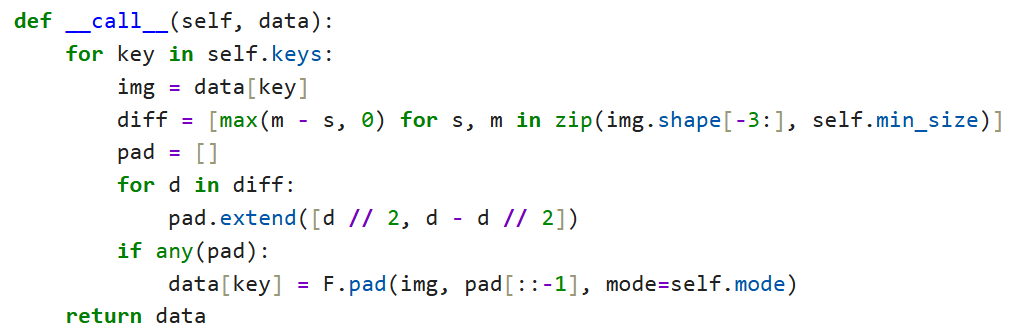
Summary (SafeRandCropByPosNegLabeld); if the processed photo is judged to be. If it is promising, use RandCropByPosNegLabeld. If the processed photo is judged to have no foreground, use CenterSpatialCropd. This is done to solve the problem that "all that is cut out is the background" which prevents Dice from being improved.

Next, let me introduce my custom EnsureMinSizePad — Patch tool to ensure minimum size. The reason for using this tool is that some images are too small to be cropped into a patch of the specified size → resulting in an error. This tool will automatically fill it up to a large enough size for you.





* def \_\_init\_\_(self, keys, min\_size=(48, 48, 48), mode="constant"): #min\_size → minimum size requirement, default is (48, 48, 48).
* mode → padding mode, "constant" means padding with 0.



* Each key (usually "image" and "label") is processed.
* Compare the current img size to see if it is smaller than min\_size.
* diff → Calculate how many cells need to be filled (if they are less than zero, do not fill them).
* To achieve symmetrical padding, divide the diff equally between the left and right sides.
* For example, diff = [4, 2, 6] → pad = [2,2,1,1,3,3], each dimension is allocated front and back.
* If padding is required, use torch.nn.functional.pad() to complete the patch.
* pad[::-1] → Because the format of F.pad is to write the last dimension first, the order needs to be reversed.

The two pieces of code I customized are "remedial tools" whose main purpose is to solve the two major problems of image features being too small and being unable to learn due to being cut off by the background.

Why did I want to use these two tools? That's because when I first trained the model, I found that the Dice of the model did not change, but the loss kept decreasing. I began to explore why this happened. During the thinking process, I guessed that it might be because the model only learned to judge the background, but did not learn about the marks spleen. So I used these two classes to try to solve the problem.

Next, to verify whether the model has the real labeled spleen, I used the following code to grab a few pictures for verification.



* Read data from train\_loader (training data iterator).
* enumerate(): Returns both batch index (idx) and data batch (batch).

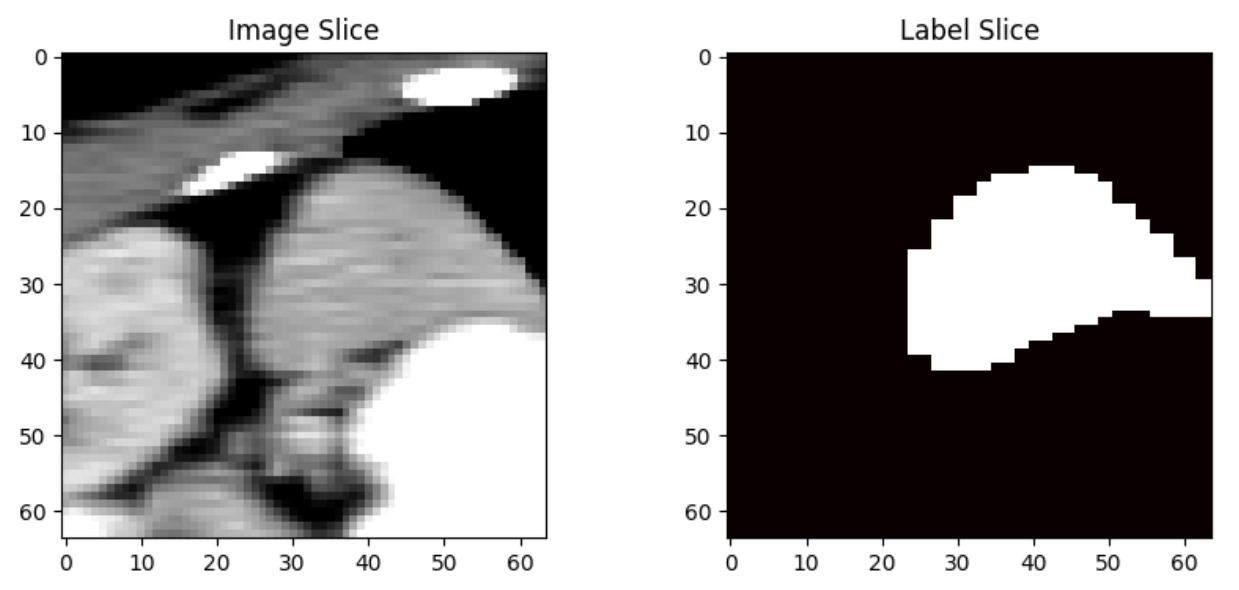
Each batch is a dictionary containing:

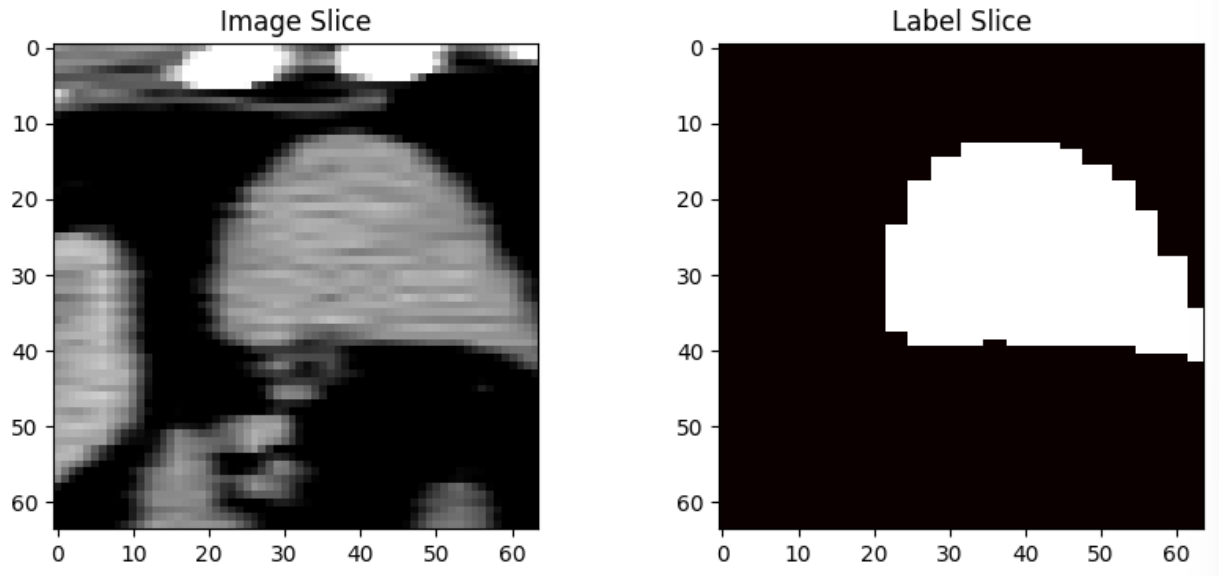
* batch["image"] → image (Tensor).
* batch["label"] → label (Segmentation mask).
* Take the first sample ([0], because batch size=1 or more),
* Remove single channel ([0], usually MRI is single channel grayscale).
* Use .cpu() to move Tensor to CPU (back from GPU), and then convert to .numpy() to facilitate subsequent plt drawing.
* mid → index of the middle slice (depth axis/slice axis of a 3D volume).

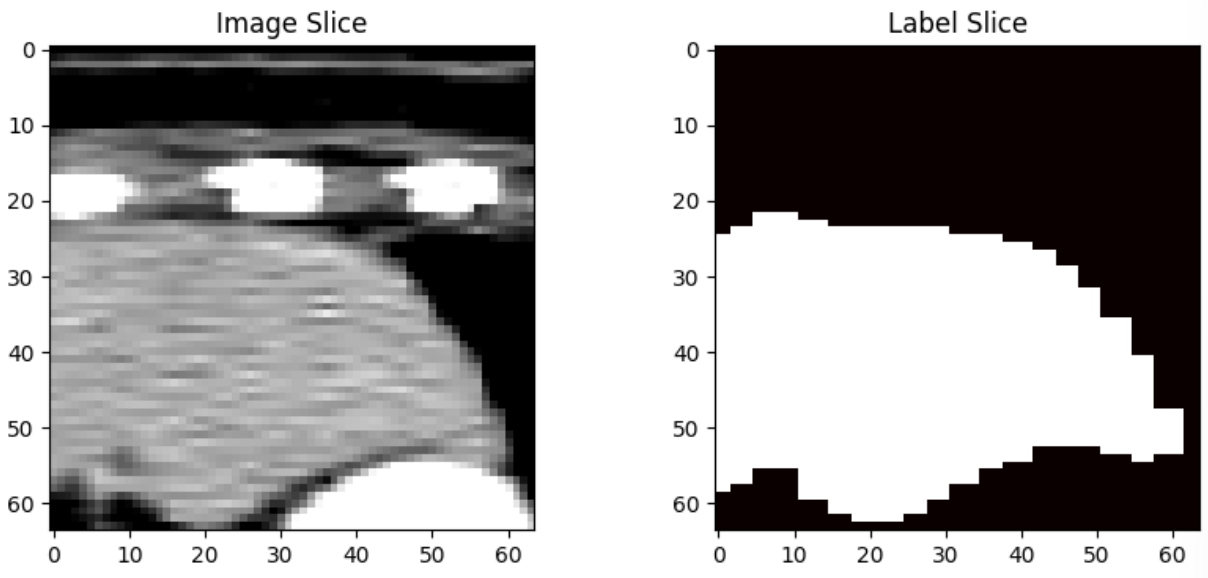
For example, if MRI is (96, 96, 96), mid is 48. This way you can see the images and labels in the middle layer, which are usually more representative.

* Create a new figure and set its size to 10 width × 4 height.
* subplot(1,2,1) → 1 row and 2 columns in total, this is the 1st subplot.
* imshow(img[mid], cmap="gray") → Draw the MRI image in grayscale.
* title() → Write "Image Slice" as the title.
* cmap="hot" → Labels are usually binary (0=background, 1=spleen), use hot color scheme to make the spleen more prominent.
* plt.title("Label Slice") #The title is "Label Slice".
* plt.show() # Display this set of images & labels. A pair of images is displayed each time a loop is run.
* if idx >= 2: break # Only the first 3 batches (index 0, 1, 2) are shown. Will not run the entire data set to avoid too many images.

The following are some pictures that are run out. As shown in the pictures, it can be clearly seen that there are marks spleen.







After confirming that the images were correctly labeled, we entered the model training phase. After searching online, we found that many people recommended using UNet as the model architecture for medical image processing.

Model introduction: UNet for Medical Image Segmentation. Let’s start with the core concepts of the UNet structure.

Encoder (downsampling path) :

* Just like the CNN classifier, it continuously performs convolution and pooling to learn high-level semantic features.

Decoder (upsampling path) :

Upsample the features extracted by the Encoder back to the original image size.

Skip Connections :

* "Directly pass" the low-level features of the Encoder to the Decoder (solving the problem of feature loss).

Suitable for segmentation :

* It can output a segmentation map of the same size as the input, which is suitable for pixel-level image segmentation.

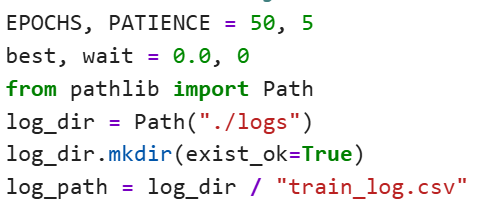
using 3D UNet :

* Process 3D medical images (MRI, CT) instead of single 2D images.

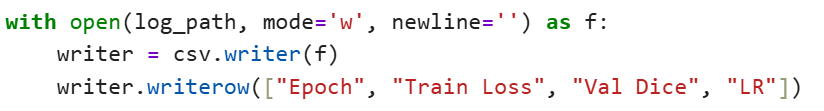
The following is a brief summary of my model settings :

* Use MONAI's built-in 3D UNet.
* in\_channels=1 (single-channel grayscale MRI image).
* out\_channels=2 (binary classification: background, spleen).
* Use DiceLoss as the loss function.
* Use Adam optimizer and ReduceLROnPlateau to adjust the learning rate.
* The core goal is to improve the Dice score, which means that the model has learned to “accurately segment the spleen.”

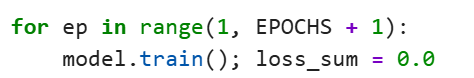
Let’s take a look at the architecture of the code for training the model.



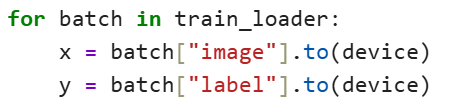
* EPOCHS = Maximum of 50 training reps.
* PATIENCE = 5 → If the validation set Dice does not improve for 5 consecutive times, stop early.
* Create a ./logs folder and save the training results as train\_log.csv.



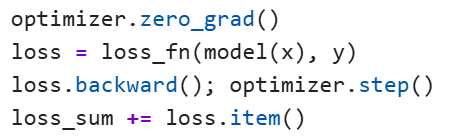
* Open the CSV file and write the table headers (Epoch, Training Loss, Validation Dice, Learning Rate).



* model.train() → Enable training mode (e.g. enable dropout, batchnorm).
* loss\_sum → records the total loss of each batch and is used to calculate the average loss.



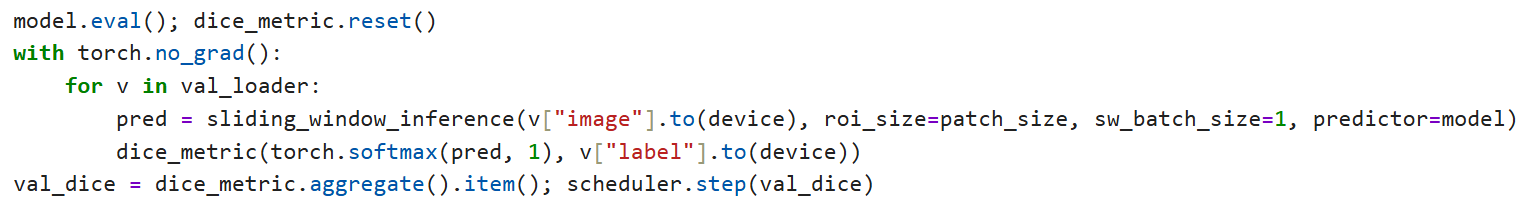
* Take each batch from train\_loader.
* x = imaging (MRI).
* y = label (spleen mask).



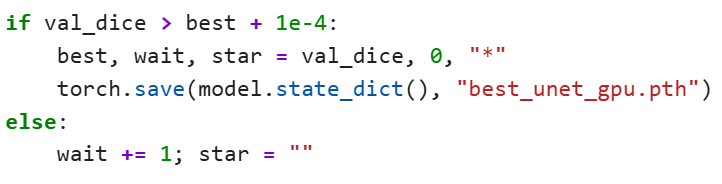
* zero\_grad() → Clears the gradient.
* loss = loss\_fn(model(x), y) → Calculate DiceLoss.
* backward() → Backward propagation.
* step() → Update weights.
* loss\_sum accumulates the loss of each batch.



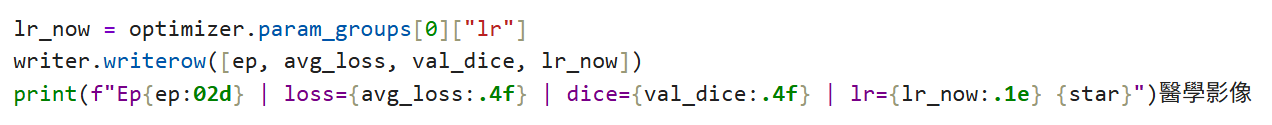
* The average of all batch losses.



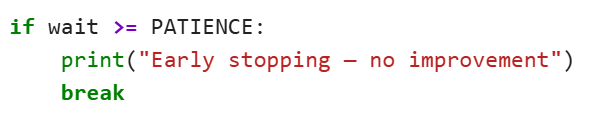
* model.eval() → The model switches to evaluation mode (turn off dropout, BN).
* sliding\_window\_inference → Process large 3D volumes using a sliding window approach.
* torch.softmax(pred, 1) → Convert predictions to probabilities.
* dice\_metric() → Calculate the Dice of each batch, and finally .aggregate() to get the average Dice.
* scheduler.step(val\_dice) → Adjust the learning rate according to Dice.



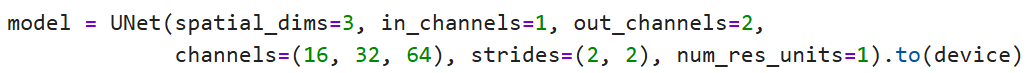
* If validation Dice improves, update best score & save model.
* Otherwise, wait times wait + 1.
* Star is used to mark epochs with progress.



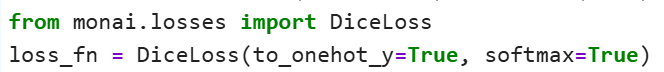
* Print results every epoch & write to CSV.
* Contains epoch number, average training loss, validation Dice, and learning rate.



* If the validation Dice does not improve for 5 consecutive times, stop training.

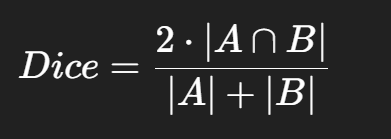


* spatial\_dims=3 → 3D UNet, suitable for processing 3D medical images (such as MRI, CT).
* in\_channels=1 → Input image has 1 channel (monochrome MRI).
* out\_channels=2 → The output segmentation result is binary classification (background=0, spleen=1).
* channels=(16, 32, 64) → The number of feature maps in each layer of the network. From shallow to deep layers, the number of channels in each convolution layer increases, and more abstract features are learned.
* strides=(2, 2) → Encoder Downsampling factor for each layer (usually each layer is reduced by 1/2).
* num\_res\_units=1 → Number of Residual Blocks used in each layer.



* DiceLoss → A loss function specially designed for image segmentation, which is particularly effective for unbalanced data (such as a lot of background and very little foreground).
* to\_onehot\_y=True → Automatically convert Ground Truth to One-Hot encoding (necessary because DiceLoss is calculated for multiple categories).
* softmax=True → Perform softmax on the model output to convert it into a probability distribution, and then calculate Dice.

DiceLoss principle introduction :



→ A measure of how well the prediction overlaps with the true label, with 1 representing perfect overlap and 0 representing a complete error.

Adam : optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

* Adam → A commonly used adaptive learning rate optimizer.
* lr=1e-3 → The initial learning rate is set to 0.001.

ReduceLROnPlateau : scheduler = ReduceLROnPlateau(optimizer, mode="max", factor=0.5, patience=3)

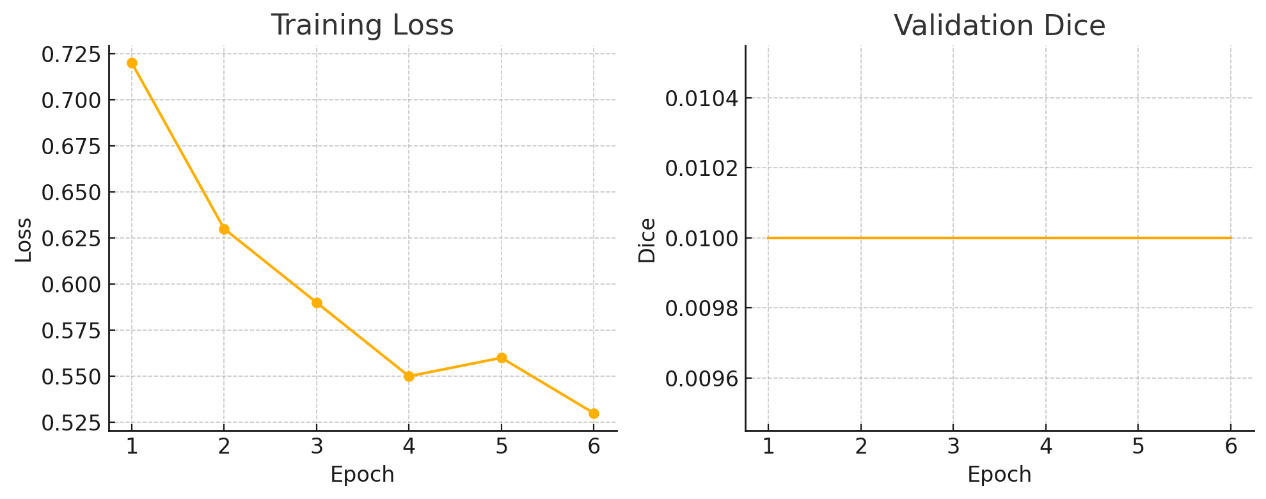
* ReduceLROnPlateau → Monitor a metric (here, verify the Dice score).
* mode="max" → We hope that the monitored Dice is as high as possible (the indicator should go up).
* factor=0.5 → If Dice does not make progress for a long time, the learning rate will be halved.
* patience=3 → If Dice does not improve for 3 times in a row, a learning rate reduction is triggered.

This is a dynamic learning rate adjustment strategy that helps the model converge more stably.

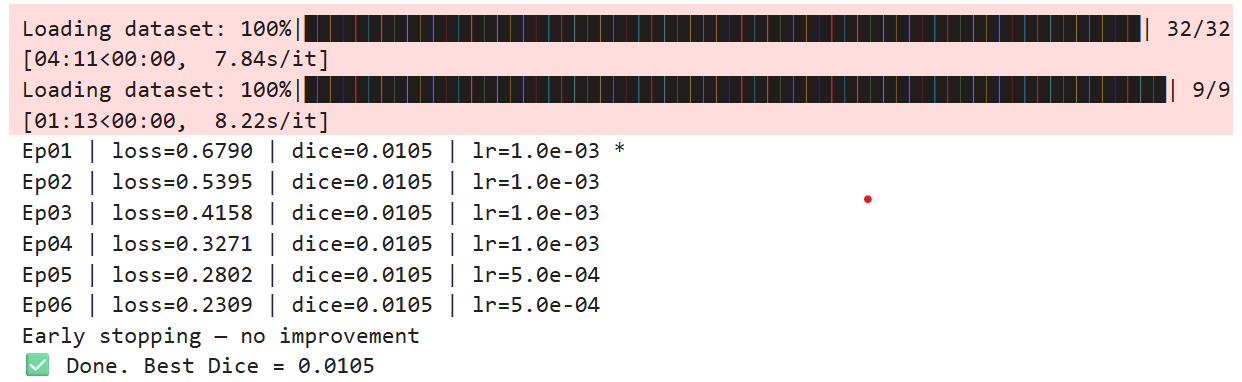
DiceMetric : dice\_metric = DiceMetric(include\_background=False, reduction="mean")

* DiceMetric → Tool for evaluating Dice scores.
* include\_background=False → Do not count Dice from the background category and only focus on the spleen.
* reduction="mean" → Average the Dice scores of multiple batches.

Result :



I originally thought that the model did not recognize the spleen so Dice did not decrease, but after I adjusted it, Dice still did not change. Later, I found a possible reason. It was predicted that the file name or file structure was different, which caused the model to fail to capture the marked and organized pictures. However, after further adjustments and improvements, I found that Dice still did not change. In the future, I will try to find more reasons why the model Dice has not changed.



Thoughts :

**1. Project Goals and Context Analysis**

In this project, I focused on "Lumbar Spine MRI Segmentation and Volume Measurement" as the core task, utilizing the open-source tools **3D Slicer** and **MONAI** for medical imaging processing.

Project objectives:

1. Segment, compile, and measure the volume of lumbar spine MRI images.
2. Implement a 3D U-Net segmentation model using MONAI for training and validation.
3. Troubleshoot GPU/CPU environment issues, data loading errors, and foreground segmentation tests.

**2. Discussion and Problem-Solving**

**(A) 3D Slicer Workflow**

* **DICOM Loading Issues**: Initially unable to load DICOM images properly; later resolved by adding format conversion information and opening .nii.gz files.
* **Grow from Seeds Usage**: Encountered segmentation failure due to skipping the initialize step before running.
* **LabelMap vs. ClosedSurface Volume Differences**: Explained the principles behind LM and CS volume calculations, aiding in result validation and interpretation.

**(B) MONAI Model Development and Training**

* **EnsureMinimumSized ImportError**: MONAI could not import EnsureMinimumSized; I created a custom function EnsureMinSizePad with equivalent functionality.
* **Custom Foreground Replacement in SafeRandCropByPosNegLabeld**: Frequently failed to find foreground patches; fallback modified to use CenterCrop.
* **Missing Foreground in Labels**: Used torch.unique(label) to inspect label value distributions, ensuring valid labels contain [0., 1.].

**(C) Model Training Results and Performance Analysis**

* **Dice Stuck at 0.0105**: Root causes included:
  + Patch cropping not finding foreground.
  + Validation labels padded with invalid formats (zeros + arbitrary values).
  + U-Net configuration too small (insufficient channels).
* **Debugging Techniques**:
  + Visualized image/label pairs using matplotlib.
  + Printed torch.unique(...) on the first batch.

**3. Hands-On Experience and Process**

* I added training log output train\_log.csv, summarizing Loss and Dice trends with pandas and matplotlib.
* External support like SafeRandCrop and Pad filler was the most valuable custom engineering adjustment.
* Through this project, I gained a complete understanding of the full workflow in a medical segmentation project: preprocessing, normalization, model training, and result analysis.

**4. Future Directions and Reflections**

* For potential experimental extensions, I could consider using nnUNet.
* If continuing with MONAI, I should redesign the pipeline to include modules like OneHotEncodeLabeld and CastToTyped(label) for proper label handling.
* After completing the assignment, I am now capable of independently building and managing a full 3D MRI segmentation pipeline.

**Final Conclusion** :

This project demonstrates my ability to tackle AI implementation and problem-solving in medical imaging, reaching the level of a standard research project. It also lays a solid foundation for further exploration in statistical analysis, federated learning, and diverse data testing.