

Lab5: Image segmentation (MONAI)

3099704 AI for Digital Health (2025/2)



Outline

Objective

Material

MONAI: What is it? Built-in models?

Lab 5.0: YOLOv8s (Bonus!)

Dataset: kvasir dataset (2017)

Lab 5.1: UNet (2D segmentation)

Dataset: Skin Cancer MNIST (HAM10000)

Lab 5.2: UNet (3D segmentation)

Dataset: Brain Tumor Segmentation 2020

Dataset (BraTS2020)

3D Slicer

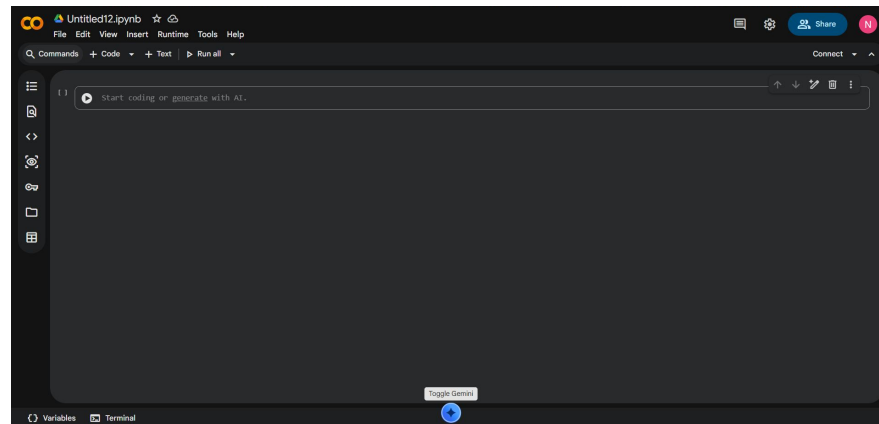
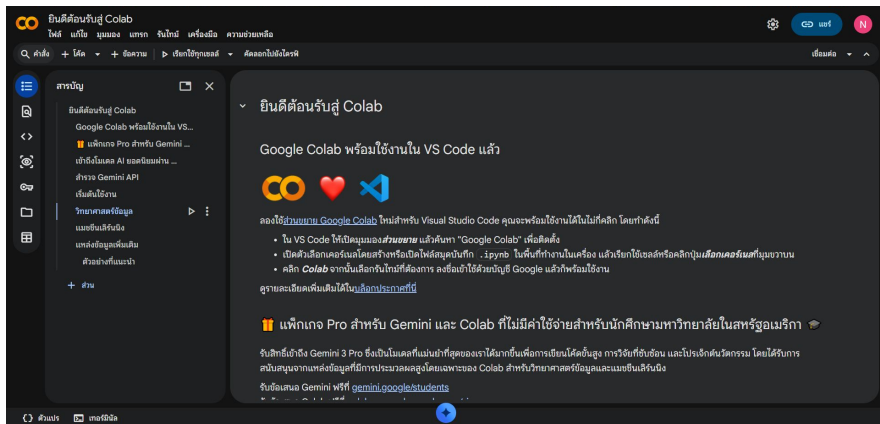
Objective

- Create segmentation model
- Use the **MONAI library** to build deep learning model (UNet)



Material

- With **Google Colab**, you don't need to install any software. All you need is a Google account, and you can start using it right away. Simply visit: <https://colab.research.google.com/> or select NEW NOTEBOOK to start a new file.



MONAI: What is it? Compare to Pytorch?

MONAI is an open-source medical imaging AI framework initiated by NVIDIA in 2020. PyTorch is a general-purpose framework for deep learning, while **MONAI** is built on PyTorch but adds specialized functions for medical imaging tasks, such as

- **Data Loading:** CacheDataset supports big data and caching
- **Transforms:** monai.transforms supports 3D medical volume
- **Networks:** build-in models, such as UNet and SegResNet
- **Inferers:** sliding_window_inference enable inference 3D volumes by 2D models



MONAI: Built-in models



What's New Highlights **API Reference** Installation Guide Precision and Accelerating More ▾



Q Search the docs ... Ctrl + K

Section Navigation

Applications
Auto3dseg
Federated Learning
Model Bundle
Transforms
Loss functions
Network architectures
Metrics
Optimizers
Data
Engines
Inference methods
Event handlers
Visualizations
Utilities

Nets

AHNet

```
class monai.networks.nets.AHNet(layers=(3, 4, 6, 3), spatial_dims=3,
in_channels=1, out_channels=1, psp_block_num=4, upsample_mode='transpose',
pretrained=False, progress=True)
```

[\[source\]](#)

AHNet based on [Anisotropic Hybrid Network](#). Adapted from [Isqshr's official code](#). Except from the original network that supports 3D inputs, this implementation also supports 2D inputs. According to the [tests for deconvolutions](#), using "transpose" rather than linear interpolations is faster. Therefore, this implementation sets "transpose" as the default upsampling method.

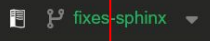
To meet the requirements of the structure, the input size for each spatial dimension (except the last one) should be: divisible by $2^{**}(\text{psp_block_num} + 3)$ and no less than 32 in "transpose" mode, and should be divisible by 32 and no less than $2^{**}(\text{psp_block_num} + 3)$ in other upsample modes. In addition, the input size for the last spatial dimension should be divisible by 32, and at least one spatial size should be no less than 64.

Parameters:

- layers** (`tuple`) – number of residual blocks for 4 layers of the network (layer1...layer4). Defaults to (3, 4, 6, 3).
- spatial_dims** (`int`) – spatial dimension of the input data. Defaults to 3.
- in_channels** (`int`) – number of input channels for the network. Default to 1.
- out_channels** (`int`) – number of output channels for the network. Defaults to 1.

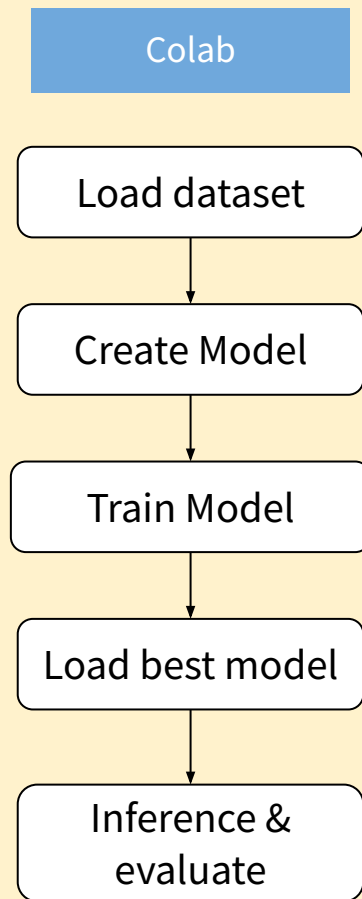
Nets

AHNet
AHNet
DenseNet
DenseNet121
DenseNet169
DenseNet201
DenseNet264
EfficientNet
BlockArgs
EfficientNetBN
EfficientNetBNFeatures
SegResNet
SegResNetDS
SegResNetVAE
ResNet
SENet
SENet154
SEResNet50
SEResNet101
SEResNet152
SEResNext50
SEResNext101
HighResNet
DynUNet
UNet



Lab5.0: YOLOv8s (Bonus!)

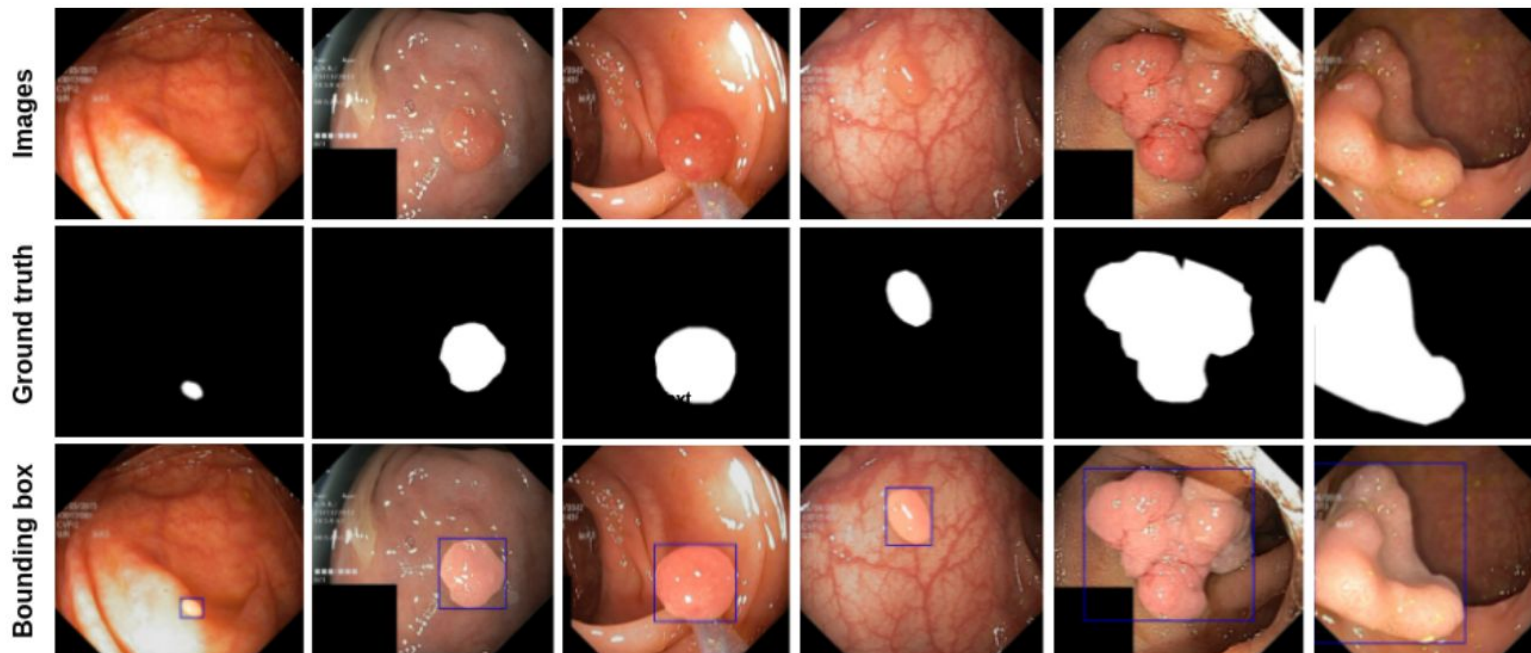
In this lab, you will create a polyp segmentation model (**YOLOv8s**) and evaluate its performance using **Ultralytics library** in Google Colab.



Dataset: kvasir dataset (2017)

- **Kvasir dataset** consists of 4,000 annotated images, including 8 classes showing anatomical landmarks, pathological findings, or endoscopic procedures in the GI tract.
- The dataset consists of the images with different resolutions, from 720x576 up to 1920x1072 pixels in JPG format, and documents in JSON format.
- The dataset was released in 2017 by the **Simula Research Laboratory, Norway**.
- To simplify the experiment, we selected only **500 images** containing polyps and prepared the dataset in a format compatible with YOLO training.

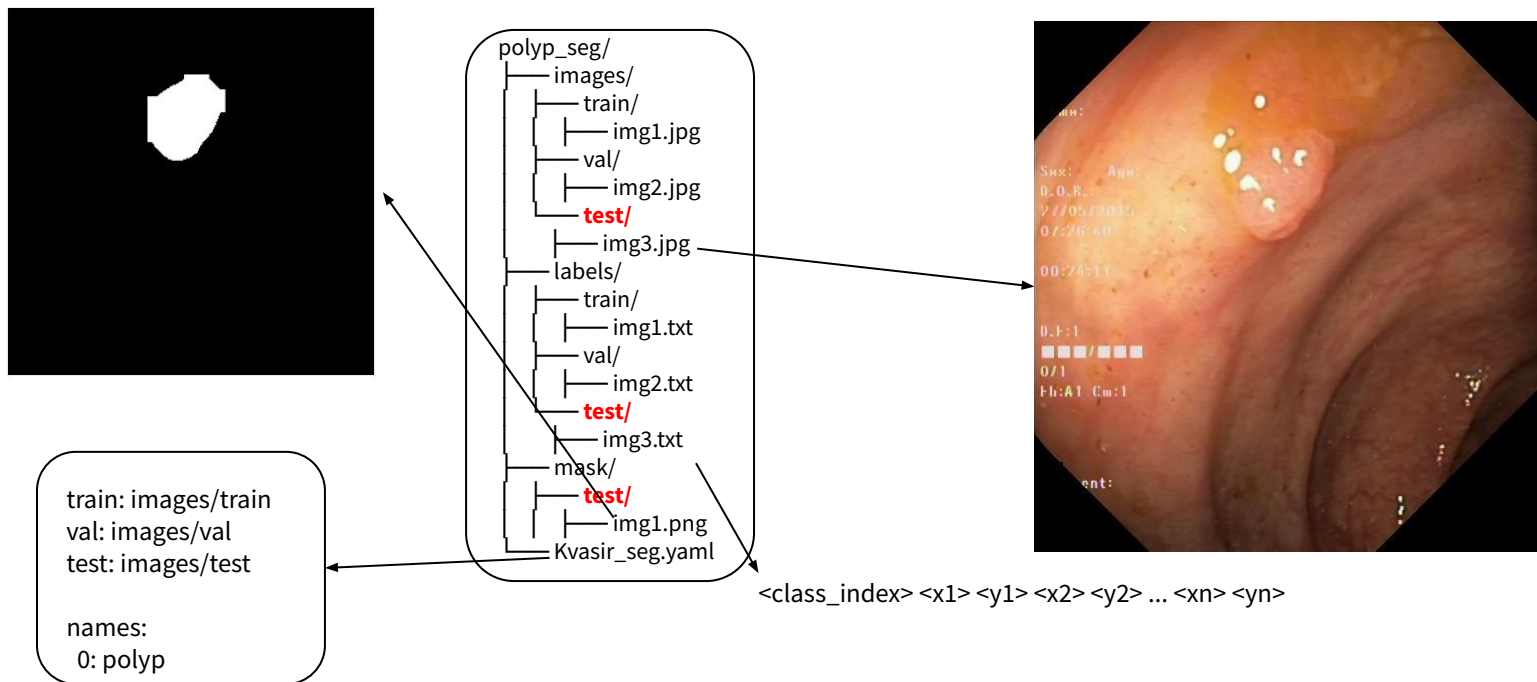
Dataset: kvasir dataset (2017)



The figure shows the example images, bounding box, and mask from Kvasir-SEG. The white mask shows the area covered by the polyp region, and the background regions contain non-polyp tissue pixels.

Lab5.0: YOLOv8s (Bonus!)

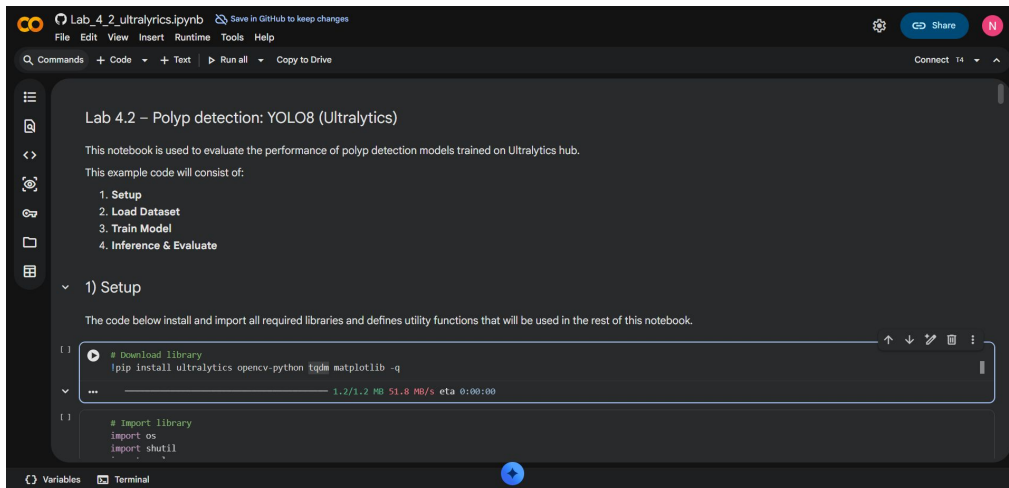
- Dataset format



Lab5.0: YOLOv8s (Bonus!): 4 steps

Run [Lab 5 0 ultralyrics.ipynb](#) (in colab)

- 1) Setup
- 2) Load Data
- 3) Train model
- 4) Inference & Evaluate



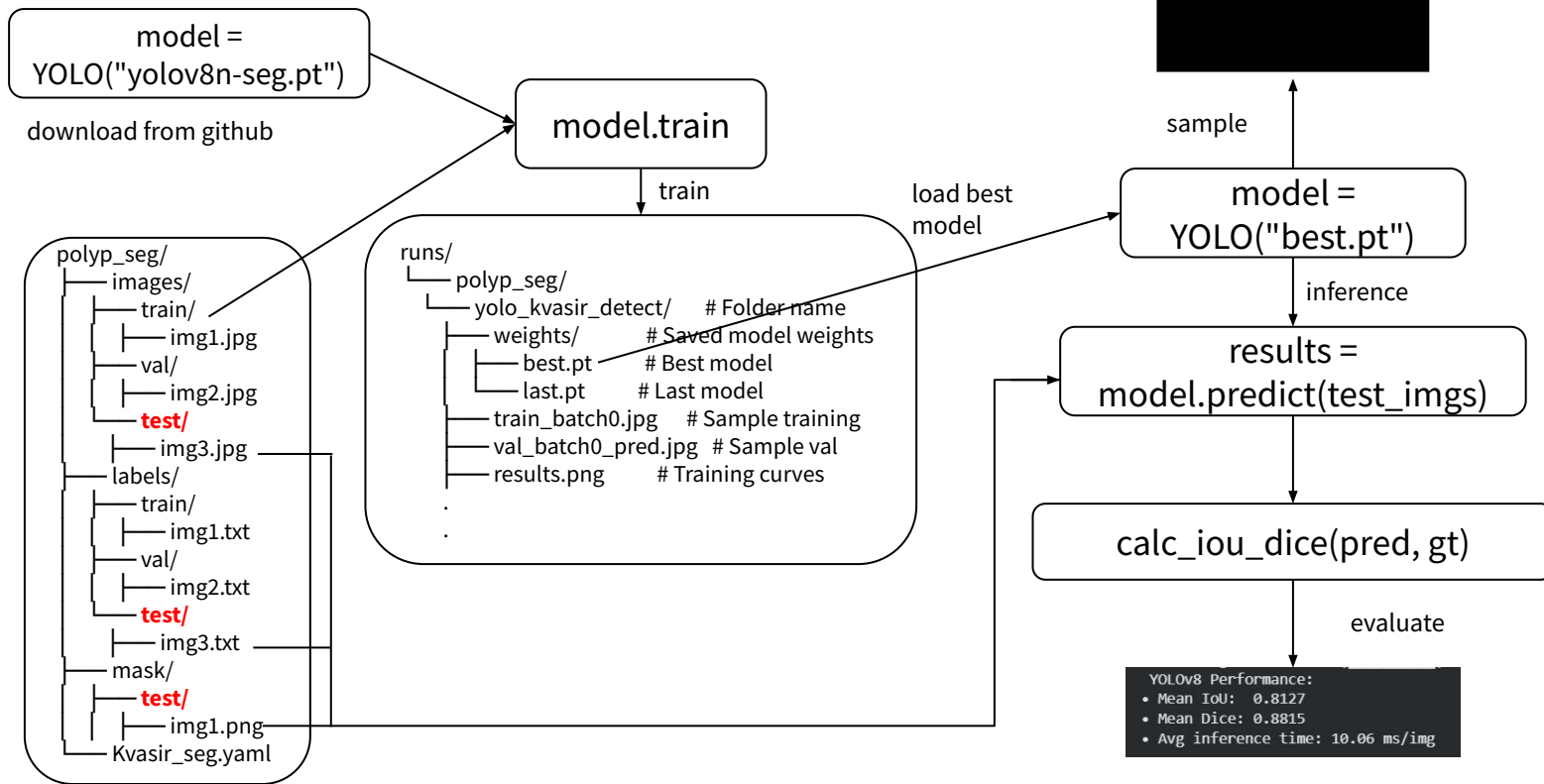
The screenshot shows a Google Colab notebook interface. The title bar indicates the file is 'Lab_4_2_ultralyrics.ipynb' and it has been saved to GitHub. The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu, there are tabs for Commands, Code, Text, Run all, and Copy to Drive. The notebook content is titled 'Lab 4.2 - Polyp detection: YOLO8 (Ultralytics)' and includes a description of the notebook's purpose and a list of steps: 1. Setup, 2. Load Dataset, 3. Train Model, and 4. Inference & Evaluate. The first step, '1) Setup', is expanded, showing a code cell with the following content:

```
[1] # download library
    !pip install ultralytics opencv-python tqdm matplotlib -q
    ...
[2] # Import library
    import os
    import shutil
```

The code cell shows a progress bar for the first command, indicating a download of 1.2/1.2 MB at 51.8 MB/s with an estimated time of 0:00:00. The bottom of the notebook shows tabs for Variables and Terminal.

Lab5.0: YOLOv8s (Bonus!): Overview

Overview of [Lab 5 0 ultralytics.ipynb](#)

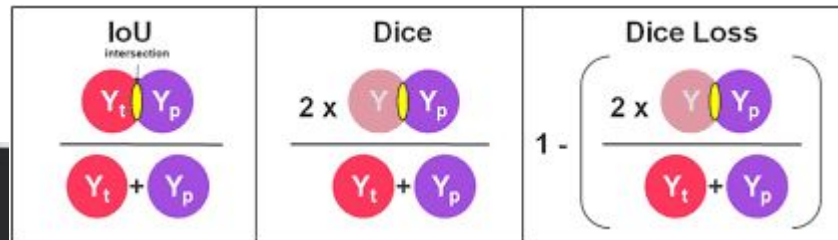


Lab5.0: YOLOv8s (Bonus!): Results

Results may vary between runs due to random seed initialization and hyperparameter tuning; however, the overall performance should be similar to the results shown on this page.

YOLOv8 Performance:

- Mean IoU: 0.8127
- Mean Dice: 0.8815
- Avg inference time: 10.06 ms/img



$$\text{IoU} = \frac{TP}{TP + FP + FN}$$

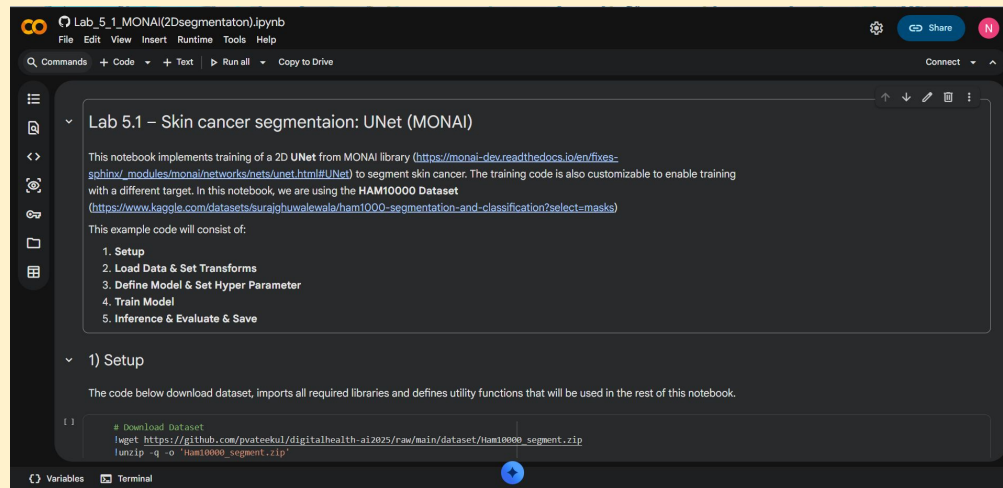
$$\text{Dice} = \frac{2TP}{2TP + FP + FN}$$

Lab5.1: UNet (2D segmentation)

In this lab, you will create and evaluate an skin cancer segmentation model (UNet) using the **MONAI library**. Code can be executed in [Lab 5 1 MONAI\(2Dsegmentaton\)](#) on Google Colab.

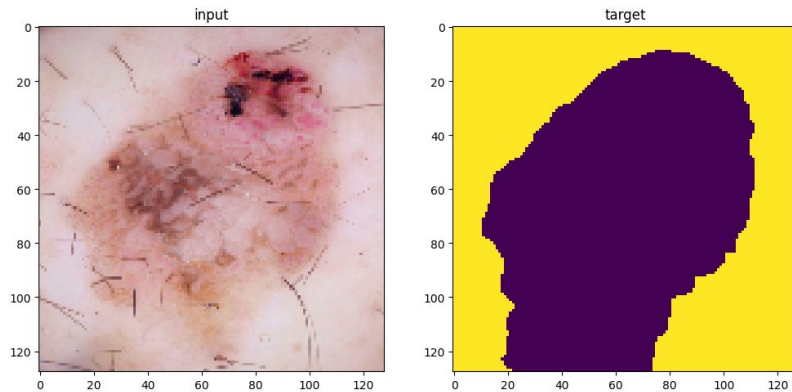
This notebook will consist of:

- 1) Setup
- 2) Load Data & Set Transforms
- 3) Define Model & Set Parameter
- 4) Train Model
- 5) Inference & Evaluate & Save

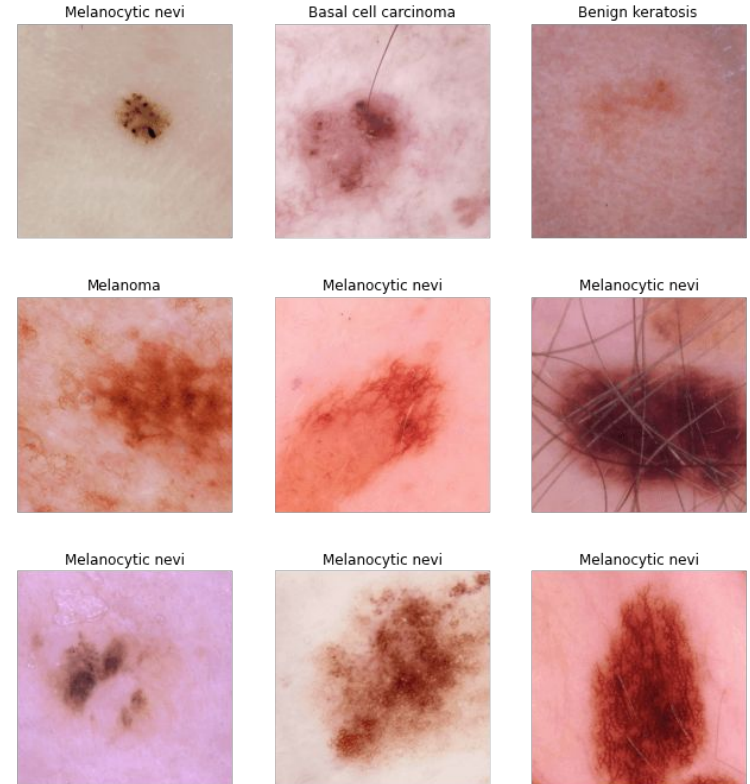
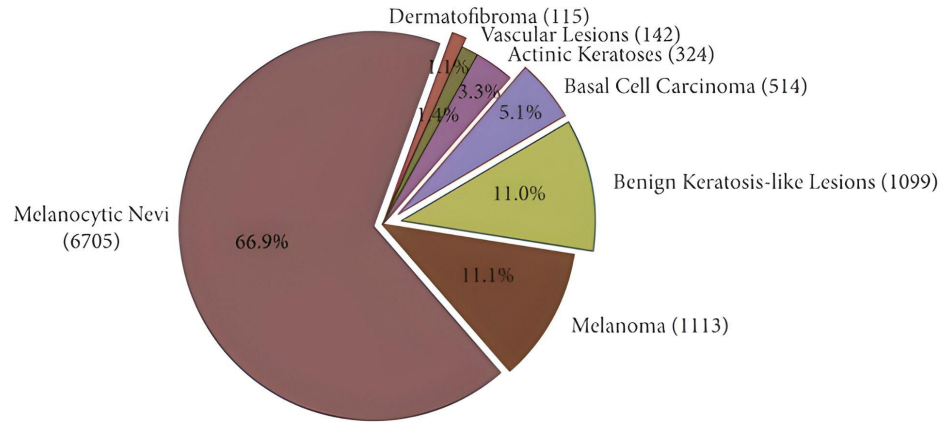


Dataset: Skin Cancer MNIST (HAM10000)

- The dataset consists of 10,015 images with 10,013 labeled objects belonging to **7 skin cancer classes**.
- The data contains image in JPG format, masks in PNG format and documents in JSON format
- The dataset was released in 2018 by the Medical University of Vienna and the University of Queensland.
- To simplify the experiment, we selected only **200 cases** containing images and masks.



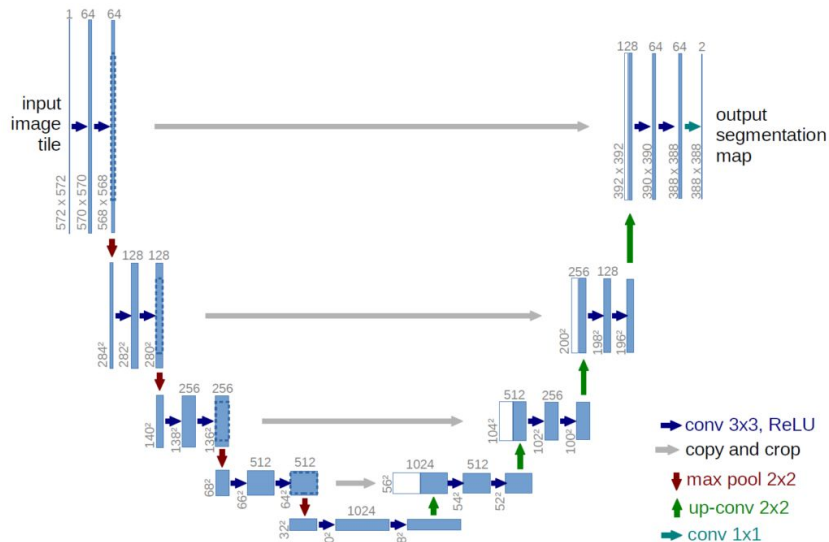
Dataset: Skin Cancer MNIST (HAM10000)



Model: UNet (Ronneberger O, 2015)

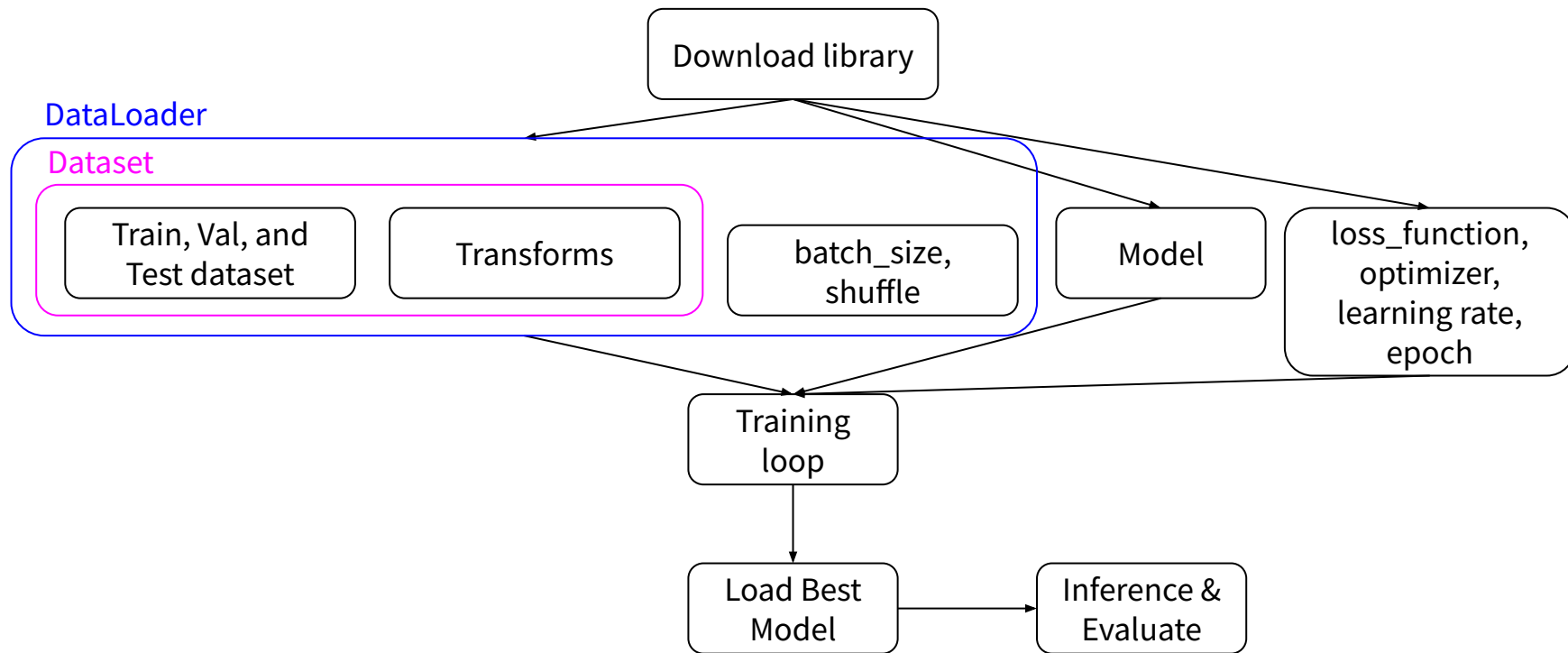
In this lab, we chose to use Unet, which has the following key architectural innovations:

- The left side (**encoder**) progressively downsamples the image to capture context.
- The right side (**decoder**) upsamples to recover spatial resolution for pixel-level predictions.
- **Skip connections** are directly connected feature maps from the encoder to the decoder at corresponding levels.



Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)



Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Train, Val, and
Test datasets

```
# Dataset
train_images = sorted(glob.glob(os.path.join("/content/images", "*.jpg")))
train_labels = sorted(glob.glob(os.path.join("/content/masks", "*.png")))
data_dicts = [{"image": image_name, "label": label_name} for image_name, label_name in zip(train_images, train_labels)]
train_files, val_files, test_files = data_dicts[:120], data_dicts[120:160], data_dicts[160:]
```

```
train_files[:5]

[{'image': '/content/images/ISIC_0024306.jpg',
 'label': '/content/masks/ISIC_0024306_segmentation.png'},
 {'image': '/content/images/ISIC_0024307.jpg',
 'label': '/content/masks/ISIC_0024307_segmentation.png'},
 {'image': '/content/images/ISIC_0024308.jpg',
 'label': '/content/masks/ISIC_0024308_segmentation.png'},
 {'image': '/content/images/ISIC_0024309.jpg',
 'label': '/content/masks/ISIC_0024309_segmentation.png'},
 {'image': '/content/images/ISIC_0024310.jpg',
 'label': '/content/masks/ISIC_0024310_segmentation.png'}]
```

Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Transforms
(pre)

```
# Transforms (pre&post proceeding)
roi_size = (128, 128)
train_transforms = Compose(
    [ # 1) Load data path -> tensor
      LoadImaged(keys=["image", "label"], ensure_channel_first=True, image_only=True, dtype=torch.float),
      # 2) scale intensity [0-255] -> [0-1]
      ScaleIntensityRanged(keys=["image", "label"], a_min=0, a_max=255, b_min=0.0, b_max=1.0, clip=True),
      # 3) Resize tensor [HxW] -> [128x128]
      Resized(keys=["image", "label"], spatial_size=roi_size, mode=["bilinear", "nearest-exact"]),
      # 4) Augment (Random flip in each epoch.)
      RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=0),
    ]
)
val_transforms = Compose(
    [
      # 1) Load data path -> tensor
      LoadImaged(keys=["image", "label"], ensure_channel_first=True, image_only=True, dtype=torch.float),
      # 2) scale intensity [0-255] -> [0-1]
      ScaleIntensityRanged(keys=["image", "label"], a_min=0, a_max=255, b_min=0.0, b_max=1.0, clip=True),
      # 3) Resize tensor [HxW] -> [128x128]
      Resized(keys=["image", "label"], spatial_size=roi_size, mode=["bilinear", "nearest-exact"]),
    ]
)
test_transforms = Compose(
    [
      # 1) Load data path -> tensor
      LoadImaged(keys=["image", "label"], ensure_channel_first=True, image_only=True, dtype=torch.float),
      # 2) scale intensity [0-255] -> [0-1]
      ScaleIntensityRanged(keys=["image", "label"], a_min=0, a_max=255, b_min=0.0, b_max=1.0, clip=True),
      # 3) Resize tensor [HxW] -> [128x128]
      Resized(keys=["image", "label"], spatial_size=roi_size, mode=["bilinear", "nearest-exact"]),
    ]
)
```

Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Transforms
(post)

```
post_transforms = Compose(  
    [  
        Invertd(  
            keys="pred",  
            transform=test_transforms,  
            orig_keys="image",  
            meta_keys="pred_meta_dict",  
            orig_meta_keys="image_meta_dict",  
            meta_key_postfix="meta_dict",  
            nearest_interp=False,  
            to_tensor=True,  
        ),  
        AsDiscreted(keys="pred", argmax=True, to_onehot=2),  
    ]  
)
```

Lab5.1: UNet (2D segmentation)

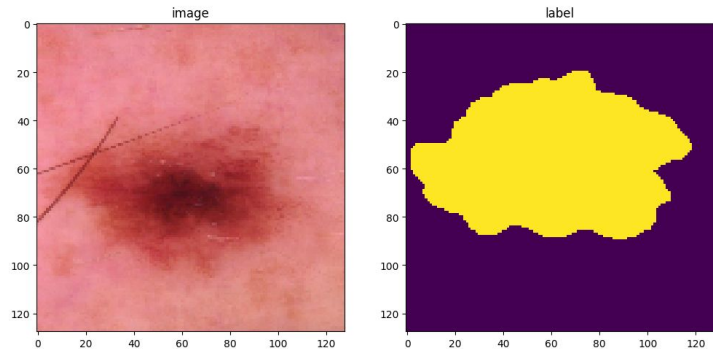
Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Dataset

Train, Val, and
Test dataset

Transforms

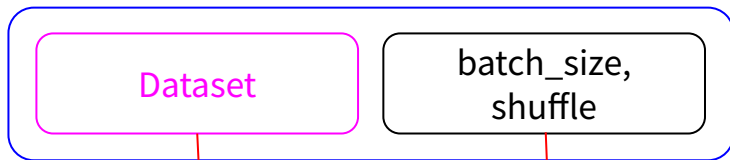
```
# Dataset
train_ds = CacheDataset(data=train_files, transform=train_transforms, cache_rate=1.0, num_workers=20)
val_ds = CacheDataset(data=val_files, transform=val_transforms, cache_rate=1.0, num_workers=20)
#Use 20 processes to prepare the data and 100% of the dataset is cached.
```



Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

DataLoader



```
# DataLoader
train_loader = DataLoader(train_ds, batch_size=4, shuffle=True, num_workers=20)
val_loader = DataLoader(val_ds, batch_size=1, num_workers=20)
```

Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Model

loss_function,
optimizer,
learning rate,
epoch

```
device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
model = UNet(spatial_dims=2, in_channels=3, out_channels=2,
             channels=(16, 32, 64, 128), strides=(2, 2, 2)).to(device)

learning_rate = 0.0001 # @param {type:"slider", min:1e-4, max:1e-3, step:1e-4}
loss_function = DiceCELoss(to_onehot_y=True, softmax=True)
optimizer = torch.optim.Adam(model.parameters(), learning_rate)
dice_metric = DiceMetric(include_background=False, reduction="mean")

max_epochs = 50 # @param {type:"slider", min:5, max:100, step:1}
val_interval = 2 # @param {type:"slider", min:1, max:10, step:1}
```


Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Training
loop

```
for epoch in range(max_epochs):
    print("-" * 10)
    print(f"epoch {epoch + 1}/{max_epochs}")
    model.train()

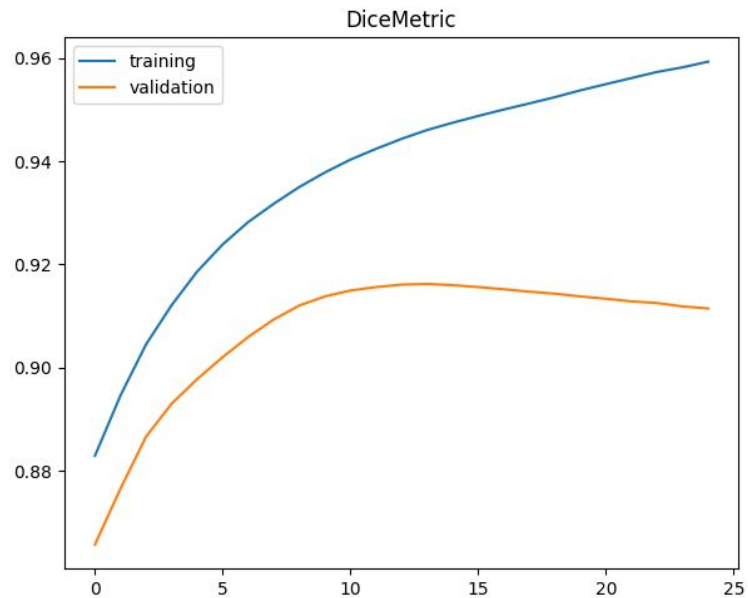
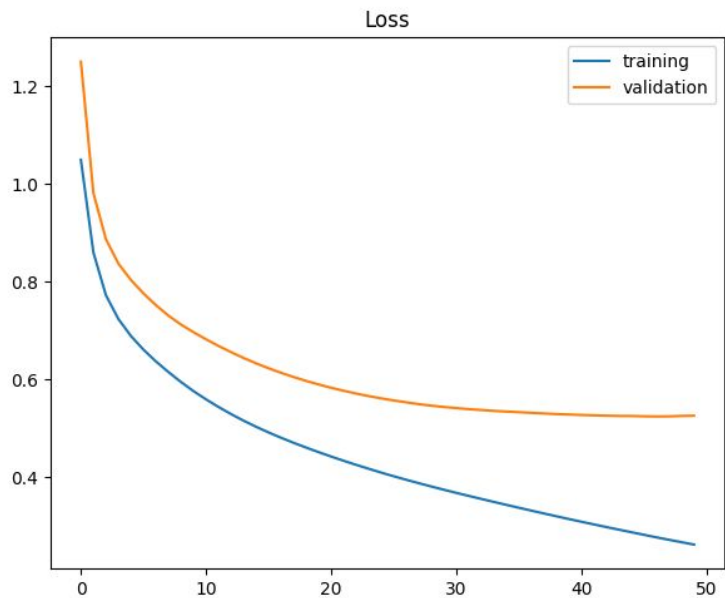
    epoch_loss = 0
    step = 0
    for i, batch_data in enumerate(train_loader):
        step += 1
        inputs, labels = (batch_data["image"].to(device), batch_data["label"].to(device))

        optimizer.zero_grad()
        outputs = model(inputs)           # predict outputs
        loss = loss_function(outputs, labels) # calculate loss
        loss.backward()                   # backpropagation
        optimizer.step()
        optimizer.zero_grad()
```

Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Training
loop



Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Load Best
Model

Evaluate

Inference

```
# Example & Save (.png)
# Load best model
model.load_state_dict(torch.load(os.path.join("/content/UNet_best_metric_model.pth")))
model.eval()

DSC_all = []
with torch.no_grad():
    for i, batch in enumerate(test_loader):
        img = batch["image"].to(device)
        label = batch["label"].to(device)

        output = model(img)
        pred = torch.argmax(output, dim=1, keepdim=True)

        dice_metric = DiceMetric(include_background=False, reduction="mean")

        dice_metric(y_pred=pred[0], y=label[0].to(device))
        DSC_all.append(dice_metric.aggregate().item())

        if i % 5 == 0:
            visualize_rgb_with_dice(
                image=img[0],
                gt=label[0],
                pred=pred[0],
                dice_value=dice_metric.aggregate().item()
            )
        dice_metric.reset()
```

Lab5.1: UNet (2D segmentation)

Overview of [Lab 5 1 MONAI\(2Dsegmentaton\)](#)

Load Best
Model

Evaluate

Inference

```
batch["pred"] = pred
batch = [post_transforms(i) for i in decollate_batch(batch)]

# Create the output directory if it doesn't exist
output_dir = "output/UNet"
os.makedirs(output_dir, exist_ok=True)

# Create a tensor (e.g., a simple example tensor)
tensor = torch.argmax(batch[0]["pred"], dim=0).detach().cpu()

# Metadata for saving the NIfTI file
metadata = {
    "filename_or_obj": os.path.join(output_dir, "example.nii.gz")
}

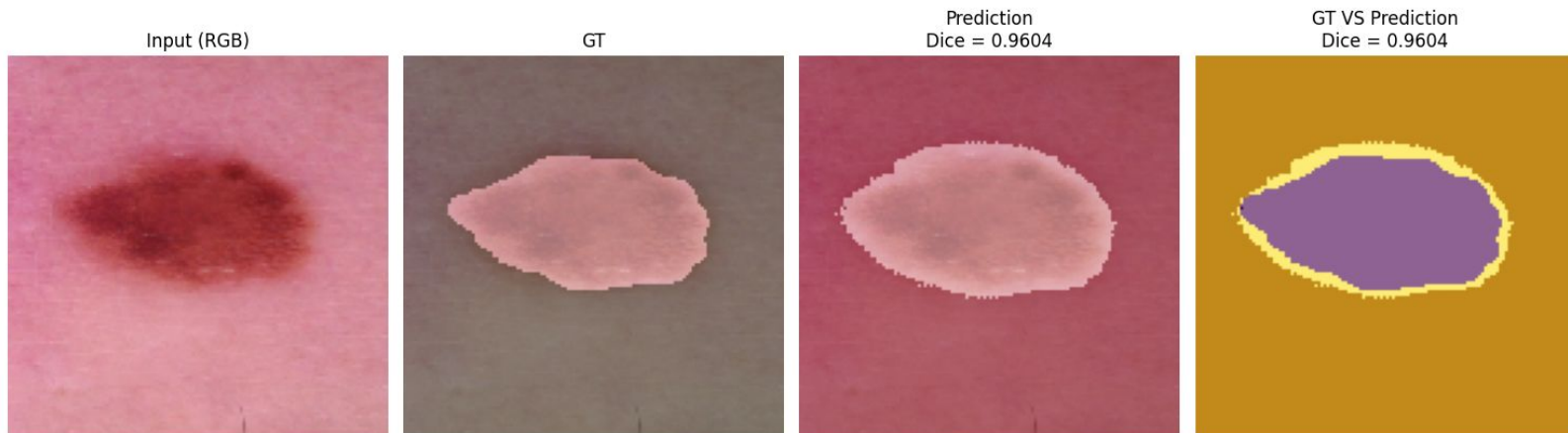
# Initialize the SaveImage transform
saver = SaveImage(output_dir=output_dir, output_postfix="_UNet", output_ext=".png", resample=False)

# Save the tensor as a NIfTI file
saver(tensor, meta_data=metadata)

print(f"NIfTI file saved in {output_dir}")
```

Lab5.1: UNet (2D segmentation)

Results may vary between runs due to random seed initialization and hyperparameter tuning; however, the overall performance should be similar to the results shown on this page.



Mean Dice: 0.7297

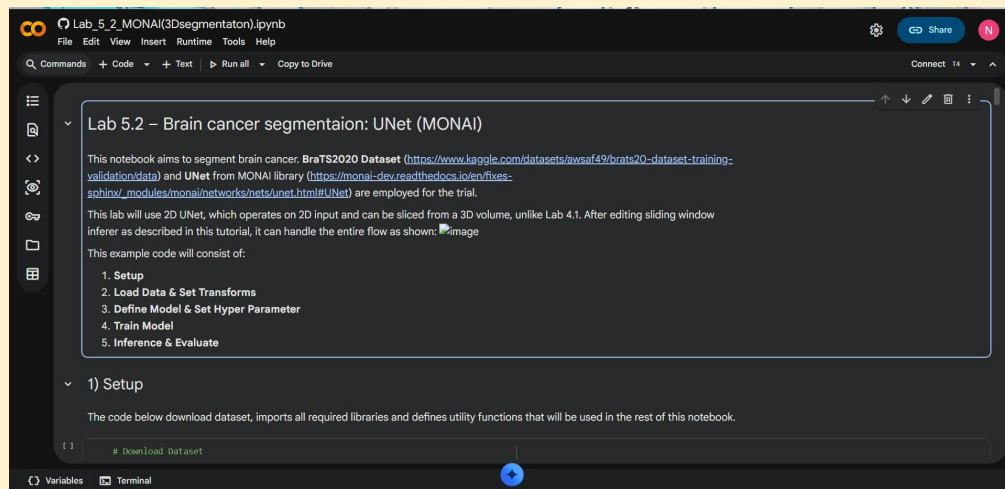
Lab5.2: UNet (3D segmentation)

In this lab, you will create and evaluate a brain tumor segmentation model (UNet). Unlike Lab 5.1, this lab focuses on a 3D segmentation task, with the main differences in the DataLoader and inference process. All code can be executed in

[Lab 5 2 MONAI\(3Dsegmentaton\).](#)

This notebook will consist of:

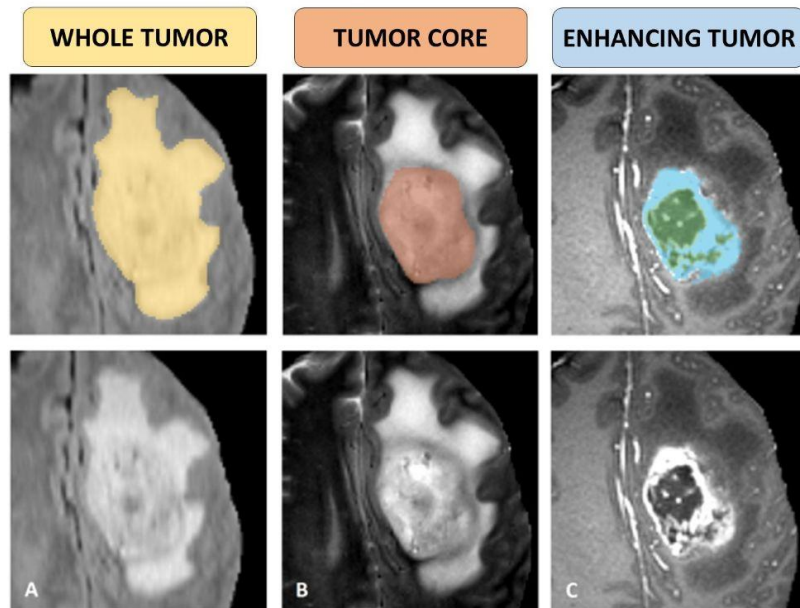
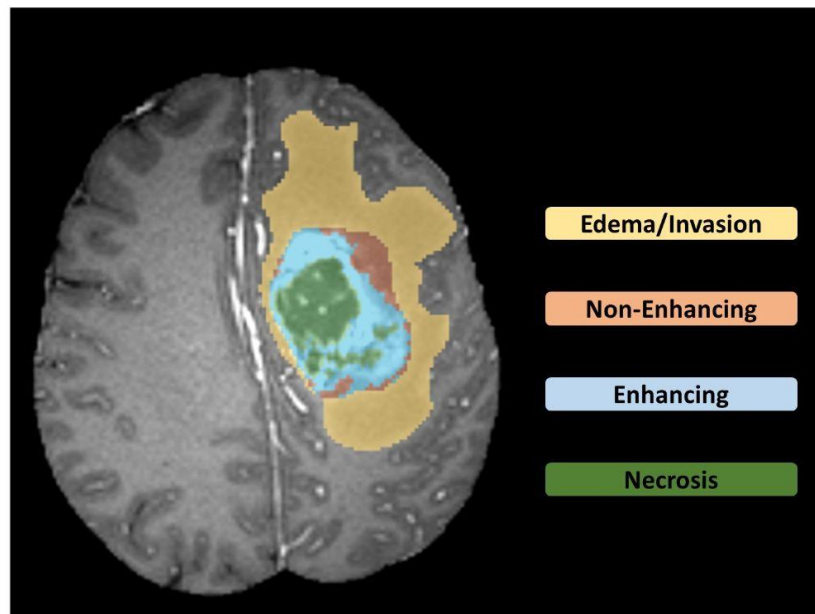
- 1) Setup
- 2) Load Data & Set Transforms
- 3) Define Model & Set Parameter
- 4) Train Model
- 5) Inference & Evaluate & Save



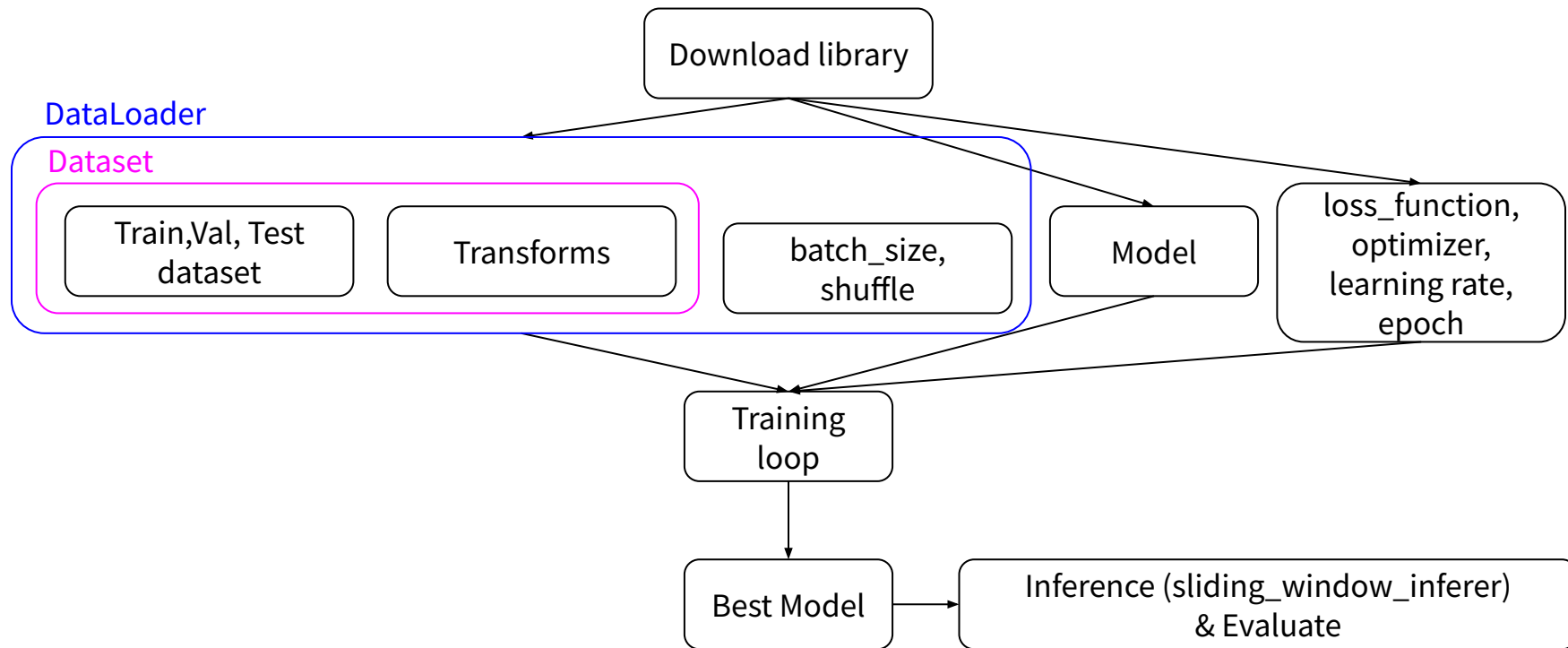
Dataset: Brain Tumor Segmentation 2020 Dataset (BraTS2020)

- BraTS 2020 is dataset for brain tumor segmentation, consisting of T1, T1ce, T2, FLAIR with tumors (4 classes), which contains image in nii.gz format
- BraTS 2020 includes **369 training cases**, with further validation/test subjects provided for evaluation.
- BraTS 2020 dataset was released by the Multimodal Brain Tumor Segmentation Challenge organizing team
- To simplify the experiment, we selected only **20 cases** containing FLAIR images and tumor masks.

Dataset: Brain Tumor Segmentation 2020 Dataset (BraTS2020)



Lab5.2: UNet (3D segmentation)



Lab5.2: UNet (3D segmentation)

Lab 5.2 is similar to Lab 5.1 in many cells; however, there are several key differences:

Transforms
(pre)

```
class ConvertToTumorChannel(MapTransform):
    def __call__(self, data):
        d = dict(data)
        for key in self.keys:
            d[key] = torch.isin(d[key], torch.tensor([1, 2, 3, 4], device=d[key].device)).float()
        return d

# Transforms (pre&post proceeding)
roi_size = (128, 128, 128)

train_transforms = Compose(
    [# 1) Load data path -> tensor
    LoadImaged(keys=["image", "label"], ensure_channel_first=True, image_only=True, dtype=torch.float),
    # 2) Orient the image to the standard coordinate system
    Orientationd(keys=["image", "label"], axcodes="RAS"),
    # 3) scale intensity [0-1023] -> [0-1]
    ScaleIntensityRanged(keys=["image"], a_min=0, a_max=1023, b_min=0.0, b_max=1.0, clip=True),
    # 4) merged all tumor classes into a single class to simplify
    ConvertToTumorChannel(keys="label"),
    # 5) Resize tensor [HxWxD] -> [128x128x128]
    Resized(keys=["image", "label"], spatial_size=roi_size, mode=["bilinear", "nearest-exact"]),
    # 6) Augment (Random flip in each epoch.)
    RandFlipd(keys=["image", "label"], prob=0.5, spatial_axis=0),
)
```

Lab5.2: UNet (3D segmentation)

DataLoader

Dataset

batch_size,
shuffle

Since the data used in this lab is 3D, but we want to use it in conjunction with a 2D model, MONAI has a specific function for this.

```
# DataLoader 3D -> 2D
train_ds = CacheDataset(data=train_files, transform=train_transforms, cache_rate=1.0, num_workers=4)
val_ds = CacheDataset(data=val_files, transform=val_transforms, cache_rate=1.0, num_workers=4)

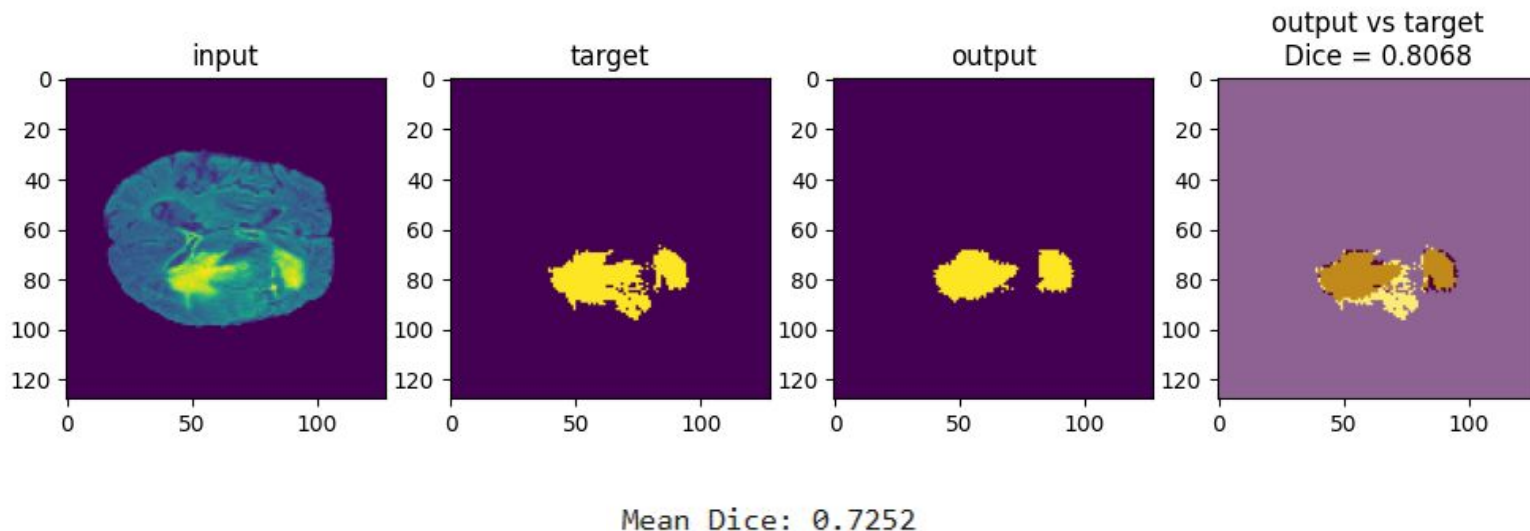
patch_func = monai.data.PatchIterd(keys=["image", "label"], patch_size=(None, None, 1), start_pos=(0, 0, 0))
patch_transform = Compose([
    SqueezeDimd(keys=["image", "label"], dim=-1), # squeeze the last dim
    Resized(keys=["image", "label"], spatial_size=[128, 128]),
])

train_patch_ds = monai.data.GridPatchDataset(data=train_ds, patch_iter=patch_func, transform=patch_transform, with_coordinates=False)
train_shuffle_ds = monai.data.ShuffleBuffer(train_patch_ds, buffer_size=30, seed=0)
train_loader = DataLoader(train_shuffle_ds, batch_size=4, num_workers=4, pin_memory=torch.cuda.is_available())

val_patch_ds = monai.data.GridPatchDataset(data=val_ds, patch_iter=patch_func, transform=patch_transform, with_coordinates=False)
val_shuffle_ds = monai.data.ShuffleBuffer(val_patch_ds, buffer_size=30, seed=0)
val_loader = DataLoader(val_shuffle_ds, batch_size=4, num_workers=4, pin_memory=torch.cuda.is_available())
```

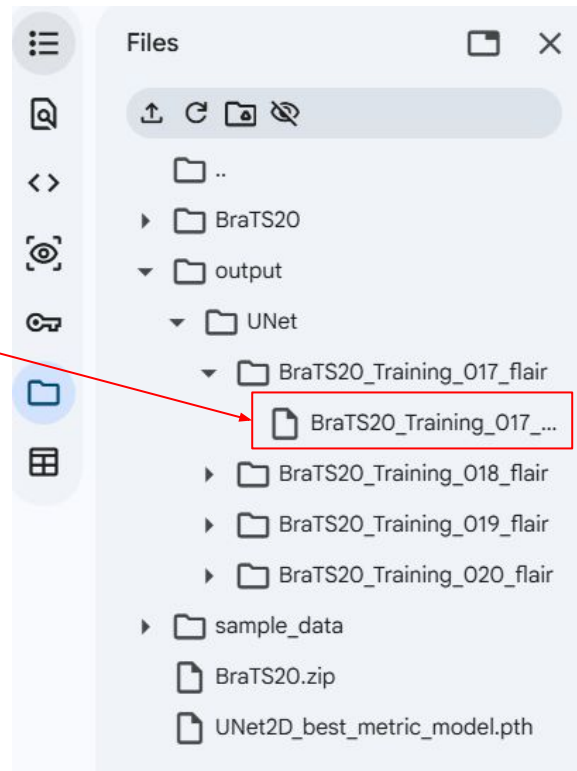
Lab5.2: UNet (3D segmentation)

Results may vary between runs due to random seed initialization and hyperparameter tuning; however, the overall performance should be similar to the results shown on this page.



Lab5.2: UNet (3D segmentation)

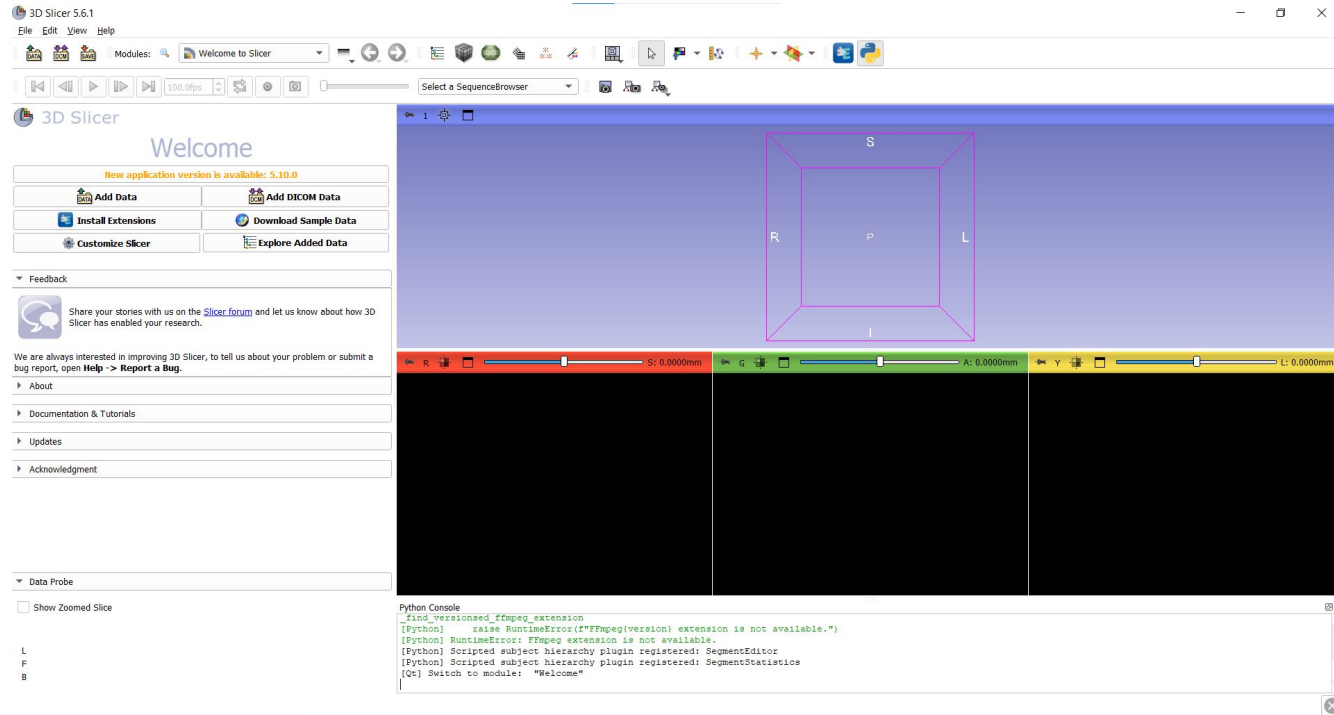
The outputs generated from the test dataset are saved as **.nii.gz** files. These files can be downloaded and opened for visualization in 3D Slicer.



3D Slicer

- **3D Slicer** is a free, open-source software platform for medical image analysis and visualization
- Supports **3D/4D imaging data** such as CT, MRI, and ultrasound
- Provides tools for **segmentation, registration, and 3D visualization**
- [Download](#)

3D Slicer (cont.)

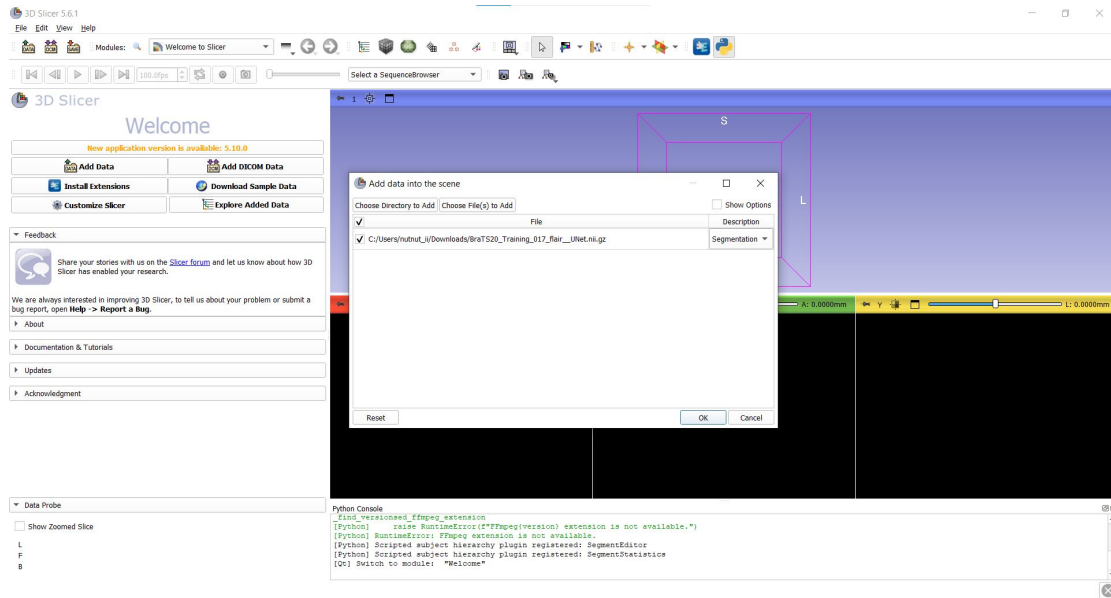


Example image of the 3D slicer program.

3D Slicer: Load a segmented data to the program

Steps to Load a Segmentation (.nii.gz) in 3D Slicer

1. Load the **.nii.gz** file using *File* → *Add Data*
2. In the **Add Data** window, set the file type to **Label Map**.
3. Click **OK** to load the segmentation.
4. Right-click the label map and select **Convert to Segmentation**.
5. The segmentation will appear in the **Segment Editor** and 3D view.



3D Slicer: Load a segmented data to the program

Steps to Load a Segmentation (.nii.gz) in 3D Slicer

1. Load the **.nii.gz** file using *File* → *Add Data*
2. In the **Add Data** window, set the file type to **Label Map**.
3. Click **OK** to load the segmentation.
4. Right-click the label map and select **Convert to Segmentation**.
5. The segmentation will appear in the **Segment Editor** and 3D view.

