## Introduction

In this notebook you will download and preprocess the data for the liver and liver tumor segmentation:
The data is provided by the medical segmentation decathlon (http://medicaldecathlon.com/)
(Data License: https://creativecommons.org/licenses/by-sa/4.0/)

You can directly download the original cardiac MRIs and segmentation maps from:
https://drive.google.com/file/d/1wEB2I6S6tQBVEPxir8cA5kFB8gTQadYY/view?usp=sharing

As this dataset has over 26GB we provide a resampled version of it. The new scans are of shape (256x256xZ), where Z is varying and reduce the size of the dataset to 2.5GB

It is directly included in this directory

```
%matplotlib notebook
from pathlib import Path
import nibabel as nib
import matplotlib.pyplot as plt
import numpy as np
from celluloid import Camera
from IPython.display import HTML
```

```
pip install celluloid
```

```
Requirement already satisfied: celluloid in /usr/local/lib/python3.10/dist-packages (0.2.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from celluloid) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->celluloid) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->celluloic
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

## Inspection:

Let's inspect some sample data

We do not need to preprocess this dataset as the necessary steps are directly performed by torchio during training

```
root = Path("/content/drive/MyDrive/08-3D-Liver-Tumor-Segmentation/Task03_Liver_rs/imagesTr")
label = Path("/content/drive/MyDrive/08-3D-Liver-Tumor-Segmentation/Task03_Liver_rs/labelsTr")
```

We start with a helper function which automatically replaces "imagesTr" with "labelsTr" in the filepaths so that we can easily switch between CT images and label masks

```
def change_img_to_label_path(path):
    """
    Replaces imagesTr with labelsTr
    """
    parts = list(path.parts)  # get all directories within the path
    parts[parts.index("imagesTr")] = "labelsTr"  # Replace imagesTr with labelsTr
    return Path(*parts)  # Combine list back into a Path object
```

```
# sample_path = list(root.glob("liver*"))[0]  # Choose a subject
# sample_path_label = change_img_to_label_path(sample_path)
sample_path = list(root.glob("liver*"))[0]  # Choose a subject
sample_path_label = change_img_to_label_path(sample_path)
```

Load NIfTI and extract image data

```
data = nib.load(sample_path)
label = nib.load(sample_path_label)

ct = data.get_fdata()
mask = label.get_fdata().astype(int)  # Class labels should not be handled as float64
```

```
nib.aff2axcodes(data.affine)
```

```
     ('R', 'A', 'S')
```

```
fig = plt.figure()
camera = Camera(fig)  # Create the camera object from celluloid

for i in range(ct.shape[2]):  # Axial view
    plt.imshow(ct[:,:,i], cmap="bone")
    mask_ = np.ma.masked_where(mask[:,:,i]==0, mask[:,:,i])
    plt.imshow(mask_, alpha=0.5)
    # plt.axis("off")
    camera.snap()  # Store the current slice
plt.tight_layout()
animation = camera.animate()  # Create the animation
```

```
HTML(animation.to_html5_video())
```

0:00 / 0:28

```
import torch
```

```
class DoubleConv(torch.nn.Module):
    """
    Helper Class which implements the intermediate Convolutions
    """
    def __init__(self, in_channels, out_channels):

        super().__init__()
        self.step = torch.nn.Sequential(torch.nn.Conv3d(in_channels, out_channels, 3, padding=1),
                                        torch.nn.ReLU(),
                                        torch.nn.Conv3d(out_channels, out_channels, 3, padding=1),
                                        torch.nn.ReLU())

    def forward(self, X):
        return self.step(X)
```

```
class UNet(torch.nn.Module):
    """
    This class implements a UNet for the Segmentation
    We use 3 down- and 3 UpConvolutions and two Convolutions in each step
    """

    def __init__(self):
        """Sets up the U-Net Structure
        """
        super().__init__()


        ############# DOWN #####################
        self.layer1 = DoubleConv(1, 32)
        self.layer2 = DoubleConv(32, 64)
        self.layer3 = DoubleConv(64, 128)
        self.layer4 = DoubleConv(128, 256)

        #########################################

        ############## UP #######################
        self.layer5 = DoubleConv(256 + 128, 128)
        self.layer6 = DoubleConv(128+64, 64)
        self.layer7 = DoubleConv(64+32, 32)
        self.layer8 = torch.nn.Conv3d(32, 3, 1)  # Output: 3 values -> background, liver, tumor
        #########################################

        self.maxpool = torch.nn.MaxPool3d(2)

    def forward(self, x):

        ####### DownConv 1#########
        x1 = self.layer1(x)
        x1m = self.maxpool(x1)
        ###########################

        ####### DownConv 2#########
        x2 = self.layer2(x1m)
        x2m = self.maxpool(x2)
        ###########################

        ####### DownConv 3#########
        x3 = self.layer3(x2m)
        x3m = self.maxpool(x3)
        ###########################

        ##### Intermediate Layer ##
        x4 = self.layer4(x3m)
        ###########################

        ####### UpCONV 1#########
        x5 = torch.nn.Upsample(scale_factor=2, mode="trilinear")(x4)  # Upsample with a factor of 2
        x5 = torch.cat([x5, x3], dim=1)  # Skip-Connection
        x5 = self.layer5(x5)
        ###########################

        ####### UpCONV 2#########
        x6 = torch.nn.Upsample(scale_factor=2, mode="trilinear")(x5)
        x6 = torch.cat([x6, x2], dim=1)  # Skip-Connection
        x6 = self.layer6(x6)
        ###########################

        ####### UpCONV 3#########
        x7 = torch.nn.Upsample(scale_factor=2, mode="trilinear")(x6)
        x7 = torch.cat([x7, x1], dim=1)
        x7 = self.layer7(x7)
        ###########################

        ####### Predicted segmentation#########
        ret = self.layer8(x7)
        return ret
```

```
model = UNet()
```

```
random_input = torch.randn(1, 1, 128, 128, 128)
```

```
with torch.no_grad():
    output = model(random_input)
assert output.shape == torch.Size([1, 3, 128, 128, 128])
```

```
!pip install torch torchvision torchio pytorch-lightning
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu118)
Requirement already satisfied: torchio in /usr/local/lib/python3.10/dist-packages (0.19.3)
Requirement already satisfied: pytorch-lightning in /usr/local/lib/python3.10/dist-packages (2.1.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: Deprecated in /usr/local/lib/python3.10/dist-packages (from torchio) (1.2.14)
Requirement already satisfied: SimpleITK!=2.0.*,!=2.1.1.1 in /usr/local/lib/python3.10/dist-packages (from torchio) (2.3.1)
Requirement already satisfied: humanize in /usr/local/lib/python3.10/dist-packages (from torchio) (4.7.0)
Requirement already satisfied: nibabel in /usr/local/lib/python3.10/dist-packages (from torchio) (4.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from torchio) (1.11.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from torchio) (4.66.1)
Requirement already satisfied: typer[all] in /usr/local/lib/python3.10/dist-packages (from torchio) (0.9.0)
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (6.0.1)
Requirement already satisfied: torchmetrics>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (1.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (23.2)
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (0.10
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.10/dist-packages (from fsspec->torch) (3.8.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->pytorch-light
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/dist-packages (from Deprecated->torchio) (1.14.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2023.7.22
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.10/dist-packages (from typer[all]->torchio) (8.1.7)
Requirement already satisfied: colorama<0.5.0,>=0.4.3 in /usr/local/lib/python3.10/dist-packages (from typer[all]->torchio) (0.4.6)
Requirement already satisfied: shellingham<2.0.0,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]->torchio) (1.5
Requirement already satisfied: rich<14.0.0,>=10.11.0 in /usr/local/lib/python3.10/dist-packages (from typer[all]->torchio) (13.7.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec->to
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fssp
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec->t
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer[a
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=10.11.0->typer
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14.0.0,>=10.1
```

```
import importlib
import sys
from pathlib import Path

import torchio as tio
import torch
import pytorch_lightning as pl
from pytorch_lightning.callbacks import ModelCheckpoint
from pytorch_lightning.loggers import TensorBoardLogger
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
# Add the directory containing the file to sys.path
sys.path.append('/content/drive/MyDrive/08-3D-Liver-Tumor-Segmentation')
```

```
#Data Set creation
def change_img_to_label_path(path):
    """
    Replace data with mask to get the masks
    """
    parts = list(path.parts)
    parts[parts.index("imagesTr")] = "labelsTr"
    return Path(*parts)
```

```
path = root
subjects_paths = list(path.glob("liver_*"))
subjects = []

for subject_path in subjects_paths:
    label_path = change_img_to_label_path(subject_path)
    subject = tio.Subject({"CT":tio.ScalarImage(subject_path), "Label":tio.LabelMap(label_path)})
    subjects.append(subject)
```

```
print(subjects)
```

```
[Subject(Keys: ('CT', 'Label'); images: 2), Subject(Keys: ('CT', 'Label'); images: 2), Subject(Keys: ('CT', 'Label'); images: 2), Su
```

```
for subject in subjects:
    assert subject["CT"].orientation == ("R", "A", "S")
```

```
process = tio.Compose([
            tio.CropOrPad((256, 256, 200)),
            tio.RescaleIntensity((-1, 1))
            ])
```

```
augmentation = tio.RandomAffine(scales=(0.9, 1.1), degrees=(-10, 10))
```

```
val_transform = process
train_transform = tio.Compose([process, augmentation])
```

```
train_dataset = tio.SubjectsDataset(subjects[:105], transform=train_transform)
val_dataset = tio.SubjectsDataset(subjects[105:], transform=val_transform)

sampler = tio.data.LabelSampler(patch_size=96, label_name="Label", label_probabilities={0:0.2, 1:0.3, 2:0.5})
#sampler = tio.data.UniformSampler(patch_size=96)
```

```
# Todo: Adapt max_length and num_workers to your hardware

train_patches_queue = tio.Queue(
     train_dataset,
     max_length=40,
     samples_per_volume=5,
     sampler=sampler,
     num_workers=4,
    )

val_patches_queue = tio.Queue(
     val_dataset,
     max_length=40,
     samples_per_volume=5,
     sampler=sampler,
     num_workers=4,
    )
```

```
# TODO, adapt batch size according to your hardware
batch_size = 2

train_loader = torch.utils.data.DataLoader(train_patches_queue, batch_size=batch_size, num_workers=0)
val_loader = torch.utils.data.DataLoader(val_patches_queue, batch_size=batch_size, num_workers=0)
```

```
class Segmenter(pl.LightningModule):
    def __init__(self):
        super().__init__()

        self.model = UNet()

        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-4)
        self.loss_fn = torch.nn.CrossEntropyLoss()

    def forward(self, data):
        pred = self.model(data)
        return pred

    def training_step(self, batch, batch_idx):
        # You can obtain the raw volume arrays by accessing the data attribute of the subject
        img = batch["CT"]["data"]
        mask = batch["Label"]["data"][:,0]  # Remove single channel as CrossEntropyLoss expects NxHxW
        mask = mask.long()

        pred = self(img)
        loss = self.loss_fn(pred, mask)

        # Logs
        self.log("Train Loss", loss)
        if batch_idx % 50 == 0:
            self.log_images(img.cpu(), pred.cpu(), mask.cpu(), "Train")
        return loss


    def validation_step(self, batch, batch_idx):
        # You can obtain the raw volume arrays by accessing the data attribute of the subject
        img = batch["CT"]["data"]
        mask = batch["Label"]["data"][:,0]  # Remove single channel as CrossEntropyLoss expects NxHxW
        mask = mask.long()

        pred = self(img)
        loss = self.loss_fn(pred, mask)

        # Logs
        self.log("Val Loss", loss)
        self.log_images(img.cpu(), pred.cpu(), mask.cpu(), "Val")

        return loss


    def log_images(self, img, pred, mask, name):

        results = []
        pred = torch.argmax(pred, 1) # Take the output with the highest value
        axial_slice = 50  # Always plot slice 50 of the 96 slices

        fig, axis = plt.subplots(1, 2)
        axis[0].imshow(img[0][0][:,:,axial_slice], cmap="bone")
        mask_ = np.ma.masked_where(mask[0][:,:,axial_slice]==0, mask[0][:,:,axial_slice])
        axis[0].imshow(mask_, alpha=0.6)
        axis[0].set_title("Ground Truth")

        axis[1].imshow(img[0][0][:,:,axial_slice], cmap="bone")
        mask_ = np.ma.masked_where(pred[0][:,:,axial_slice]==0, pred[0][:,:,axial_slice])
        axis[1].imshow(mask_, alpha=0.6, cmap="autumn")
        axis[1].set_title("Pred")

        self.logger.experiment.add_figure(f"{name} Prediction vs Label", fig, self.global_step)



    def configure_optimizers(self):
        #Caution! You always need to return a list here (just pack your optimizer into one :))
        return [self.optimizer]





# Instanciate the model
model = Segmenter()


# Create the checkpoint callback
checkpoint_callback = ModelCheckpoint(
    monitor='Val Loss',
    save_top_k=10,
    mode='min')
```

```
import pytorch_lightning as pl
from pytorch_lightning.loggers import TensorBoardLogger

# Specify the number of GPUs you want to use
gpus = 1

# Use the 'gpu' accelerator if GPUs are available, otherwise use 'cpu'
accelerator = 'gpu' if gpus > 0 else 'cpu'

# Create the trainer
trainer = pl.Trainer(accelerator=accelerator,
                     logger=TensorBoardLogger(save_dir="./logs"),
                     log_every_n_steps=1,
                     callbacks=checkpoint_callback,
                     max_epochs=100)
```

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
```

```
import torch

# Check if GPU is available
print(torch.cuda.is_available())
```

```
True
```

```
# Train the model.
# This might take some hours depending on your GPU
trainer.fit(model, train_loader, val_loader)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name    | Type             | Params
---------------------------------------------
0 | model   | UNet             | 5.8 M
1 | loss_fn | CrossEntropyLoss | 0
---------------------------------------------
5.8 M     Trainable params
0         Non-trainable params
5.8 M     Total params
23.344    Total estimated model params size (MB)

/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'val_dataloader' does not ha
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/utilities/data.py:77: Trying to infer the `batch_size` from an ambiguous c
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/data_connector.py:441: The 'train_dataloader' does not
Epoch 0: 46%                                                        120/263 [01:07<01:20, 1.77it/s, v_num=4]
```

```
from IPython.display import HTML
from celluloid import Camera
```

```
model = Segmenter.load_from_checkpoint("/content/drive/MyDrive/08-3D-Liver-Tumor-Segmentation/weights/epoch=97-step=25773.ckpt")
model = model.eval()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device);
```

```
# Select a validation subject and extract the images and segmentation for evaluation
IDX = 4
mask = val_dataset[IDX]["Label"]["data"]
imgs = val_dataset[IDX]["CT"]["data"]

# GridSampler
grid_sampler = tio.inference.GridSampler(val_dataset[IDX], 96, (8, 8, 8))
```

```
# GridAggregator
aggregator = tio.inference.GridAggregator(grid_sampler)
```

```
# DataLoader for speed up
patch_loader = torch.utils.data.DataLoader(grid_sampler, batch_size=4)
```

```
# Prediction
with torch.no_grad():
    for patches_batch in patch_loader:
        input_tensor = patches_batch['CT']["data"].to(device)  # Get batch of patches
        locations = patches_batch[tio.LOCATION]  # Get locations of patches
```

```
# Extract the volume prediction
output_tensor = aggregator.get_output_tensor()


fig = plt.figure()
camera = Camera(fig)  # create the camera object from celluloid
pred = output_tensor.argmax(0)

for i in range(0, output_tensor.shape[3], 2):  # axial view
    plt.imshow(imgs[0,:,:,i], cmap="bone")
    mask_ = np.ma.masked_where(pred[:,:,i]==0, pred[:,:,i])
    label_mask = np.ma.masked_where(mask[0,:,:,i]==0, mask[0,:,:,i])
    plt.imshow(mask_, alpha=0.1, cmap="autumn")
    #plt.imshow(label_mask, alpha=0.5, cmap="jet")  # Uncomment if you want to see the label

    # plt.axis("off")
    camera.snap()  # Store the current slice
animation = camera.animate()  # create the animation
```

```
HTML(animation.to_html5_video())  # convert the animation to a video
```

0:00 / 0:20