RadioTherapy

Generated by Doxygen 1.13.2

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 GUI Namespace Reference

Classes

- class MainWindow

Variables

- app = QApplication(sys.argv)
- window = MainWindow()

5.1.1 Variable Documentation

5.1.1.1 app

```
GUI.app = QApplication(sys.argv)
```

5.1.1.2 window

```
GUI.window = MainWindow()
```

5.2 training Namespace Reference

Functions

- select_channel (image, channel=0)
- KL_loss (z_mu, z_sigma)

Variables

- force
- directory = os.environ.get("MONAI_DATA_DIRECTORY")
- str root_dir = "/Users/giannigagliardi/Documents/Git/RadioTherapy/data"
- int batch_size = 1
- int channel = 0
- train_transforms
- train_ds
- train_loader
- device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- autoencoder
- discriminator
- l1_loss = L1Loss()
- adv_loss = PatchAdversarialLoss(criterion="least_squares")
- loss_perceptual
- float adv_weight = 0.01
- float perceptual_weight = 0.001
- int kl_weight = 1e-6
- optimizer_g = torch.optim.Adam(params=autoencoder.parameters(), lr=1e-4)
- optimizer_d = torch.optim.Adam(params=discriminator.parameters(), lr=1e-4)
- int n_epochs = 10
- int autoencoder_warm_up_n_epochs = 5
- int val_interval = 10
- list epoch_recon_loss_list = [ ]
- list epoch_gen_loss_list = [ ]
- list epoch_disc_loss_list = [ ]
- list val_recon_epoch_loss_list = [ ]
- list intermediary_images = [ ]
- int n_example_images = 4
- int epoch_loss = 0
- int gen_epoch_loss = 0
- int disc_epoch_loss = 0
- progress_bar = tqdm(enumerate(train_loader), total=len(train_loader), ncols=110)
- images = batch["image"].to(device)
- set_to_none
- reconstruction
- z_mu
- z_sigma
- kl_loss = KL_loss(z_mu, z_sigma)
- recons_loss = l1_loss(reconstruction.float(), images.float())
- p_loss = loss_perceptual(reconstruction.float(), images.float())
- int loss_g = recons_loss + kl_weight ∗ kl_loss + perceptual_weight ∗ p_loss
- logits_fake = discriminator(reconstruction.contiguous().float())[-1]
- generator_loss
- loss_d_fake
- logits_real = discriminator(images.contiguous().detach())[-1]
- loss_d_real
- tuple discriminator_loss = (loss_d_fake + loss_d_real) ∗ 0.5
- float loss_d = adv_weight ∗ discriminator_loss
- fontsize
- prop
- color
- linewidth
- label

- int idx = 0
- img = reconstruction[idx, channel].detach().cpu().numpy()
- fig
- axs
- nrows
- ncols
- ax = axs[0]
- cmap
- unet
- scheduler
- enabled
- first_batch = first(train_loader)
- z = autoencoder.encode_stage_2_inputs(first_batch["image"].to(device))
- int scale_factor = 1 / torch.std(z)
- inferer = LatentDiffusionInferer(scheduler, scale_factor=scale_factor)
- optimizer_diff = torch.optim.Adam(params=unet.parameters(), lr=1e-4)
- list epoch_loss_list = [ ]
- scaler = GradScaler()
- device_type
- noise = torch.randn_like(z).to(device)
- timesteps
- noise_pred
- loss = F.mse_loss(noise_pred.float(), noise.float())
- num_inference_steps
- synthetic_images

5.2.1 Function Documentation

5.2.1.1 KL_loss()

```
training.KL_loss (
              z_mu,
              z_sigma)
```

5.2.1.2 select_channel()

```
training.select_channel (
              image,
              channel = 0)
```

5.2.2 Variable Documentation

5.2.2.1 adv_loss

```
training.adv_loss = PatchAdversarialLoss(criterion="least_squares")
```

5.2.2.2 adv_weight

```
float training.adv_weight = 0.01
```

5.2.2.3 autoencoder

```
training.autoencoder
```

Initial value:

```
=  AutoencoderKL(
      spatial_dims=3,
      in_channels=1,
      out_channels=1,
      num_channels=(32, 32, 32),
      latent_channels=2,
      num_res_blocks=1,
      norm_num_groups=8,
      attention_levels=(False, False, True),
   )
```

5.2.2.4 autoencoder_warm_up_n_epochs

```
int training.autoencoder_warm_up_n_epochs = 5
```

5.2.2.5 ax

```
training.ax = axs[0]
```

5.2.2.6 axs

```
training.axs
```

5.2.2.7 batch_size

```
int training.batch_size = 1
```

5.2.2.8 channel

```
int training.channel = 0
```

5.2.2.9 cmap

```
training.cmap
```

5.2.2.10 color

```
training.color
```

5.2.2.11 device

```
training.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

5.2.2.12 device_type

```
training.device_type
```

5.2.2.13 directory

```
training.directory = os.environ.get("MONAI_DATA_DIRECTORY")
```

5.2.2.14 disc_epoch_loss

```
int training.disc_epoch_loss = 0
```

5.2.2.15 discriminator

```
training.discriminator
```

Initial value:

```
00001 =  PatchDiscriminator(
00002         spatial_dims=3, num_layers_d=3, num_channels=32, in_channels=1, out_channels=1
00003     )
```

5.2.2.16 discriminator_loss

```
tuple training.discriminator_loss = (loss_d_fake + loss_d_real) * 0.5
```

5.2.2.17 enabled

```
training.enabled
```

5.2.2.18 epoch_disc_loss_list

```
training.epoch_disc_loss_list = []
```

5.2.2.19 epoch_gen_loss_list

```
training.epoch_gen_loss_list = []
```

5.2.2.20 epoch_loss

```
int training.epoch_loss = 0
```

5.2.2.21 epoch_loss_list

```
training.epoch_loss_list = []
```

5.2.2.22 epoch_recon_loss_list

```
list training.epoch_recon_loss_list = []
```

5.2.2.23 fig

```
training.fig
```

5.2.2.24 first_batch

```
training.first_batch = first(train_loader)
```

5.2.2.25 fontsize

```
training.fontsize
```

5.2.2.26 force

```
training.force
```

5.2.2.27 gen_epoch_loss

```
int training.gen_epoch_loss = 0
```

5.2.2.28 generator_loss

```
training.generator_loss
```

Initial value:

```
=  adv_loss(
                    logits_fake, target_is_real=True, for_discriminator=False
                )
```

5.2.2.29 idx

```
int training.idx = 0
```

5.2.2.30 images

```
training.images = batch["image"].to(device)
```

5.2.2.31 img

```
training.img = reconstruction[idx, channel].detach().cpu().numpy()
```

5.2.2.32 inferer

```
training.inferer = LatentDiffusionInferer(scheduler, scale_factor=scale_factor)
```

5.2.2.33 intermediary_images

```
list training.intermediary_images = []
```

5.2.2.34 kl_loss

```
training.kl_loss = KL_loss(z_mu, z_sigma)
```

5.2.2.35 kl_weight

```
int training.kl_weight = 1e-6
```

5.2.2.36 l1_loss

```
training.l1_loss = L1Loss()
```

5.2.2.37 label

```
training.label
```

5.2.2.38 linewidth

```
training.linewidth
```

5.2.2.39 logits_fake

```
training.logits_fake = discriminator(reconstruction.contiguous().float())[-1]
```

5.2.2.40 logits_real

```
training.logits_real = discriminator(images.contiguous().detach())[-1]
```

5.2.2.41 loss

```
training.loss = F.mse_loss(noise_pred.float(), noise.float())
```

5.2.2.42 loss_d

```
float training.loss_d = adv_weight * discriminator_loss
```

5.2.2.43 loss_d_fake

```
training.loss_d_fake
```

Initial value:

```
=  adv_loss(
                    logits_fake, target_is_real=False, for_discriminator=True
                )
```

5.2.2.44 loss_d_real

```
training.loss_d_real
```

Initial value:

```
=  adv_loss(
                    logits_real, target_is_real=True, for_discriminator=True
                )
```

5.2.2.45 loss_g

```
int training.loss_g = recons_loss + kl_weight * kl_loss + perceptual_weight * p_loss
```

5.2.2.46 loss_perceptual

```
training.loss_perceptual
```

Initial value:

```
=  PerceptualLoss(
        spatial_dims=3, network_type="squeeze", is_fake_3d=True, fake_3d_ratio=0.2
    )
```

5.2.2.47 n_epochs

```
int training.n_epochs = 10
```

5.2.2.48 n_example_images

```
int training.n_example_images = 4
```

5.2.2.49 ncols

```
training.ncols
```

5.2.2.50 noise

```
training.noise = torch.randn_like(z).to(device)
```

5.2.2.51 noise_pred

```
training.noise_pred
```

Initial value:

```
=  inferer(
                 inputs=images,
                 autoencoder_model=autoencoder,
                 diffusion_model=unet,
                 noise=noise,
                 timesteps=timesteps,
             )
```

5.2.2.52 nrows

```
training.nrows
```

5.2.2.53 num_inference_steps

```
training.num_inference_steps
```

5.2.2.54 optimizer_d

```
training.optimizer_d = torch.optim.Adam(params=discriminator.parameters(), lr=1e-4)
```

5.2.2.55 optimizer_diff

```
training.optimizer_diff = torch.optim.Adam(params=unet.parameters(), lr=1e-4)
```

5.2.2.56 optimizer_g

```
training.optimizer_g = torch.optim.Adam(params=autoencoder.parameters(), lr=1e-4)
```

5.2.2.57 p_loss

```
training.p_loss = loss_perceptual(reconstruction.float(), images.float())
```

5.2.2.58 perceptual_weight

```
float training.perceptual_weight = 0.001
```

5.2.2.59 progress_bar

```
training.progress_bar = tqdm(enumerate(train_loader), total=len(train_loader), ncols=110)
```

5.2.2.60 prop

```
training.prop
```

5.2.2.61 recons_loss

```
training.recons_loss = l1_loss(reconstruction.float(), images.float())
```

5.2.2.62 reconstruction

```
training.reconstruction
```

5.2.2.63 root_dir

```
str training.root_dir = "/Users/giannigagliardi/Documents/Git/RadioTherapy/data"
```

5.2.2.64 scale_factor

```
int training.scale_factor = 1 / torch.std(z)
```

5.2.2.65 scaler

```
training.scaler = GradScaler()
```

5.2.2.66 scheduler

```
training.scheduler
```

Initial value:

```
00001 =  DDPMScheduler(
00002         num_train_timesteps=1000,
00003         schedule="scaled_linear_beta",
00004         beta_start=0.0015,
00005         beta_end=0.0195,
00006     )
```

5.2.2.67 set_to_none

```
training.set_to_none
```

5.2.2.68 synthetic_images

```
training.synthetic_images
```

Initial value:

```
=  inferer.sample(
        input_noise=noise,
        autoencoder_model=autoencoder,
        diffusion_model=unet,
        scheduler=scheduler,
    )
```

5.2.2.69 timesteps

```
training.timesteps
```

Initial value:

```
=  torch.randint(
                    0,
                    inferer.scheduler.num_train_timesteps,
                    (images.shape[0],),
                    device=images.device,
                ).long()
```

5.2.2.70 train_ds

```
training.train_ds
```

Initial value:

```
=  CustomDataset(
        root_dir=root_dir,
        section="training",  # validation
        cache_rate=0.0,  # you may need a few Gb of RAM... Set to 0 otherwise
        num_workers=0,  # Set download to True  if the dataset hasnt been downloaded yet
        transform=train_transforms,
        download=False,
        seed=0,
    )
```

5.2.2.71 train_loader

```
training.train_loader
```

Initial value:

```
=  DataLoader(
        train_ds,
        batch_size=batch_size,
        shuffle=True,
        num_workers=0,
        persistent_workers=False,
    )
```

5.2.2.72 train_transforms

```
training.train_transforms
```

Initial value:

```
=  transforms.Compose(
        [
            transforms.LoadImaged(keys=["image"], reader=NibabelReader),
            transforms.EnsureChannelFirstd(keys=["image"]),
            transforms.Lambdad(keys="image", func=select_channel),
            transforms.EnsureChannelFirstd(keys=["image"], channel_dim="no_channel"),
            transforms.EnsureTyped(keys=["image"]),
            transforms.Orientationd(keys=["image"], axcodes="RAS"),
            transforms.Spacingd(keys=["image"], pixdim=(2.4, 2.4, 2.2), mode=("bilinear")),
            SpatialPadd(keys=["image"], spatial_size=(32, 32, 32), method='symmetric'),
            transforms.CenterSpatialCropd(keys=["image"], roi_size=(32, 32, 32)),
            transforms.ScaleIntensityRangePercentilesd(
                keys="image", lower=0, upper=99.5, b_min=0, b_max=1
            ),
        ]
    )
```

5.2.2.73 unet

```
training.unet
```

Initial value:

```
=  DiffusionModelUNet(
        spatial_dims=3,
        in_channels=2,
        out_channels=2,
        num_res_blocks=1,
        num_channels=(32, 64, 64),
        attention_levels=(False, True, True),
        num_head_channels=(0, 64, 64),
    )
```

5.2.2.74 val_interval

```
int training.val_interval = 10
```

5.2.2.75 val_recon_epoch_loss_list

```
list training.val_recon_epoch_loss_list = []
```

5.2.2.76 z

```
training.z = autoencoder.encode_stage_2_inputs(first_batch["image"].to(device))
```

5.2.2.77 z_mu

```
training.z_mu
```

5.2.2.78 z_sigma

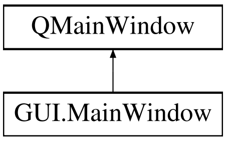
```
training.z_sigma
```

Chapter 6

Class Documentation

6.1 GUI.MainWindow Class Reference

Inheritance diagram for GUI.MainWindow:



Public Member Functions

- __init__ (self)
- open_file_dialog (self)

Public Attributes

- open_file_dialog = QPushButton("CT-Scan hochladen")
- close = QWidgetAction(self)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 __init__()

```
GUI.MainWindow.__init__ (
                self)
```

6.1.2 Member Function Documentation

6.1.2.1 open_file_dialog()

```
GUI.MainWindow.open_file_dialog (
                self)
```

6.1.3 Member Data Documentation

6.1.3.1 close

```
GUI.MainWindow.close = QWidgetAction(self)
```

6.1.3.2 open_file_dialog

```
GUI.MainWindow.open_file_dialog = QPushButton("CT-Scan hochladen")
```

The documentation for this class was generated from the following file:

- GUI.py

Chapter 7

File Documentation

7.1 GUI.py File Reference

Classes

- class GUI.MainWindow

Namespaces

- namespace GUI

Variables

- GUI.app = QApplication(sys.argv)
- GUI.window = MainWindow()

7.2 training.py File Reference

Namespaces

- namespace training

Functions

- training.select_channel (image, channel=0)
- training.KL_loss (z_mu, z_sigma)

Variables

- training.force
- training.directory = os.environ.get("MONAI_DATA_DIRECTORY")
- str training.root_dir = "/Users/giannigagliardi/Documents/Git/RadioTherapy/data"
- int training.batch_size = 1
- int training.channel = 0
- training.train_transforms
- training.train_ds
- training.train_loader
- training.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- training.autoencoder
- training.discriminator
- training.l1_loss = L1Loss()
- training.adv_loss = PatchAdversarialLoss(criterion="least_squares")
- training.loss_perceptual
- float training.adv_weight = 0.01
- float training.perceptual_weight = 0.001
- int training.kl_weight = 1e-6
- training.optimizer_g = torch.optim.Adam(params=autoencoder.parameters(), lr=1e-4)
- training.optimizer_d = torch.optim.Adam(params=discriminator.parameters(), lr=1e-4)
- int training.n_epochs = 10
- int training.autoencoder_warm_up_n_epochs = 5
- int training.val_interval = 10
- list training.epoch_recon_loss_list = [ ]
- list training.epoch_gen_loss_list = [ ]
- list training.epoch_disc_loss_list = [ ]
- list training.val_recon_epoch_loss_list = [ ]
- list training.intermediary_images = [ ]
- int training.n_example_images = 4
- int training.epoch_loss = 0
- int training.gen_epoch_loss = 0
- int training.disc_epoch_loss = 0
- training.progress_bar = tqdm(enumerate(train_loader), total=len(train_loader), ncols=110)
- training.images = batch["image"].to(device)
- training.set_to_none
- training.reconstruction
- training.z_mu
- training.z_sigma
- training.kl_loss = KL_loss(z_mu, z_sigma)
- training.recons_loss = l1_loss(reconstruction.float(), images.float())
- training.p_loss = loss_perceptual(reconstruction.float(), images.float())
- int training.loss_g = recons_loss + kl_weight ∗ kl_loss + perceptual_weight ∗ p_loss
- training.logits_fake = discriminator(reconstruction.contiguous().float())[-1]
- training.generator_loss
- training.loss_d_fake
- training.logits_real = discriminator(images.contiguous().detach())[-1]
- training.loss_d_real
- tuple training.discriminator_loss = (loss_d_fake + loss_d_real) ∗ 0.5
- float training.loss_d = adv_weight ∗ discriminator_loss
- training.fontsize
- training.prop
- training.color
- training.linewidth
- training.label

- int training.idx = 0
- training.img = reconstruction[idx, channel].detach().cpu().numpy()
- training.fig
- training.axs
- training.nrows
- training.ncols
- training.ax = axs[0]
- training.cmap
- training.unet
- training.scheduler
- training.enabled
- training.first_batch = first(train_loader)
- training.z = autoencoder.encode_stage_2_inputs(first_batch["image"].to(device))
- int training.scale_factor = 1 / torch.std(z)
- training.inferer = LatentDiffusionInferer(scheduler, scale_factor=scale_factor)
- training.optimizer_diff = torch.optim.Adam(params=unet.parameters(), lr=1e-4)
- list training.epoch_loss_list = [ ]
- training.scaler = GradScaler()
- training.device_type
- training.noise = torch.randn_like(z).to(device)
- training.timesteps
- training.noise_pred
- training.loss = F.mse_loss(noise_pred.float(), noise.float())
- training.num_inference_steps
- training.synthetic_images

Index