SIM Denoising Pipeline

Generated by Doxygen 1.9.1

Chapter 1

Namespace Index

1.1 Packages

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 analyse Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- default
- int
- action
- args = parser.parse_args()
- output_dir = pathlib.Path(args.output_dir)
- parents
- True
- exist_ok
- tuple device
- ckpt
- model
- gt_dir = pathlib.Path(args.gt_dir)
- raw_dir = pathlib.Path(args.raw_dir)
- model_1_dir = pathlib.Path(args.model_1_dir)
- gt_files = sorted(list(gt_dir.glob(args.glob_str)))
- raw_files = sorted(list(raw_dir.glob(args.glob_str)))
- model_1_files = sorted(list(model_1_dir.glob(args.glob_str)))
- model_2_dir = pathlib.Path(args.model_2_dir)
- model_2_files = sorted(list(model_2_dir.glob(args.glob_str)))
- N = len(gt_files)
- psnr = PSNR(data_range=65536, device=device)
- ssim
- df
- gt = reshape_to_bcwh(tifffile.imread(gt_files[i]))
- raw = reshape_to_bcwh(tifffile.imread(raw_files[i]))
- model_1 = reshape_to_bcwh(tifffile.imread(model_1_files[i]))
- model_2 = reshape_to_bcwh(tifffile.imread(model_2_files[i]))
- rng = np.random.default_rng(seed=31052024)
- img_idx = list(range(N))
- list gt_samples = [np.squeeze(tifffile.imread(gt_files[i])) for i in img_idx]
- list raw_samples = [np.squeeze(tifffile.imread(raw_files[i])) for i in img_idx]
- list model_1_samples
- list model_2_samples
- cmap

5.1.1 Variable Documentation

5.1.1.1 action

```
analyse.action
```

5.1.1.2 args

```
analyse.args = parser.parse_args()
```

5.1.1.3 ckpt

```
analyse.ckpt
```

5.1.1.4 cmap

```
analyse.cmap
```

5.1.1.5 default

```
analyse.default
```

5.1.1.6 device

```
tuple analyse.device
```

Initial value:

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

5.1.1.7 df

```
analyse.df
```

Initial value:

```
=  pd.DataFrame(
        columns=[
            "file",
            "psnr_raw",
            "psnr_model_1",
            "psnr_model_2",
            "ssim_raw",
            "ssim_model_1",
            "ssim_model_2",
        ]
    )
```

5.1.1.8 exist_ok

```
analyse.exist_ok
```

5.1.1.9 gt

```
analyse.gt = reshape_to_bcwh(tifffile.imread(gt_files[i]))
```

5.1.1.10 gt_dir

```
analyse.gt_dir = pathlib.Path(args.gt_dir)
```

5.1.1.11 gt_files

```
analyse.gt_files = sorted(list(gt_dir.glob(args.glob_str)))
```

5.1.1.12 gt_samples

```
list analyse.gt_samples = [np.squeeze(tifffile.imread(gt_files[i])) for i in img_idx]
```

5.1.1.13 img_idx

```
analyse.img_idx = list(range(N))
```

5.1.1.14 int

```
analyse.int
```

5.1.1.15 model

```
analyse.model
```

5.1.1.16 model_1

```
analyse.model_1 = reshape_to_bcwh(tifffile.imread(model_1_files[i]))
```

5.1.1.17 model_1_dir

```
analyse.model_1_dir = pathlib.Path(args.model_1_dir)
```

5.1.1.18 model_1_files

```
analyse.model_1_files = sorted(list(model_1_dir.glob(args.glob_str)))
```

5.1.1.19 model_1_samples

```
list analyse.model_1_samples
```

Initial value:

```
=  [
         np.squeeze(tifffile.imread(model_1_files[i])) for i in img_idx
     ]
```

5.1.1.20 model_2

```
analyse.model_2 = reshape_to_bcwh(tifffile.imread(model_2_files[i]))
```

5.1.1.21 model_2_dir

```
analyse.model_2_dir = pathlib.Path(args.model_2_dir)
```

5.1.1.22 model_2_files

```
list analyse.model_2_files = sorted(list(model_2_dir.glob(args.glob_str)))
```

5.1.1.23 model_2_samples

```
analyse.model_2_samples
```

Initial value:

```
= [
            np.squeeze(tifffile.imread(model_2_files[i])) for i in img_idx
        ]
```

5.1.1.24 N

```
analyse.N = len(gt_files)
```

5.1.1.25 output_dir

```
analyse.output_dir = pathlib.Path(args.output_dir)
```

5.1.1.26 parents

```
analyse.parents
```

5.1.1.27 parser

```
analyse.parser = argparse.ArgumentParser()
```

5.1.1.28 psnr

```
analyse.psnr = PSNR(data_range=65536, device=device)
```

5.1.1.29 raw

```
analyse.raw = reshape_to_bcwh(tifffile.imread(raw_files[i]))
```

5.1.1.30 raw_dir

```
analyse.raw_dir = pathlib.Path(args.raw_dir)
```

5.1.1.31 raw_files

```
analyse.raw_files = sorted(list(raw_dir.glob(args.glob_str)))
```

5.1.1.32 raw_samples

```
list analyse.raw_samples = [np.squeeze(tifffile.imread(raw_files[i])) for i in img_idx]
```

5.1.1.33 required

```
analyse.required
```

5.1.1.34 rng

```
analyse.rng = np.random.default_rng(seed=31052024)
```

5.1.1.35 ssim

```
analyse.ssim
```

Initial value:

```
=  SSIM(
        data_range=65536,
        kernel_size=(11, 11),
        sigma=(1.5, 1.5),
        k1=0.01,
        k2=0.03,
        gaussian=True,
        device=device,
    )
```

5.1.1.36 str

```
analyse.str
```

5.1.1.37 True

```
analyse.True
```

5.1.1.38 type

```
analyse.type
```

5.2 apply Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- choices
- default
- percentile
- action
- args = parser.parse_args()
- input_path = pathlib.Path(args.input)
- output_path = pathlib.Path(args.output)
- parents
- raw_files = sorted(input_path.glob("∗.tif"))
- data = itertools.zip_longest(raw_files, [ ])
- tuple device
- ckpt
- model
- RCAN_hyperparameters = ckpt["hyperparameters"]
- list overlap_shape
- raw = normalize(tifffile.imread(raw_file), args.p_min, args.p_max)
- restored
- output_file = output_path / ("pred_" + raw_file.name)
- imagej

5.2.1 Variable Documentation

5.2.1.1 action

```
apply.action
```

5.2.1.2 args

```
apply.args = parser.parse_args()
```

5.2.1.3 choices

```
apply.choices
```

5.2.1.4 ckpt

```
apply.ckpt
```

5.2.1.5 data

```
list apply.data = itertools.zip_longest(raw_files, [])
```

5.2.1.6 default

```
apply.default
```

5.2.1.7 device

```
tuple apply.device
```

Initial value:

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

5.2.1.8 imagej

```
apply.imagej
```

5.2.1.9 input_path

```
apply.input_path = pathlib.Path(args.input)
```

5.2.1.10 int

```
apply.int
```

5.2.1.11 model

```
apply.model
```

5.2.1.12 output_file

```
apply.output_file = output_path / ("pred_" + raw_file.name)
```

5.2.1.13 output_path

```
apply.output_path = pathlib.Path(args.output)
```

5.2.1.14 overlap_shape

```
apply.overlap_shape
```

Initial value:

```
=  [
         max(1, x // 8) if x > 2 else 0
         for x in RCAN_hyperparameters["input_shape"]
     ]
```

5.2.1.15 parents

```
apply.parents
```

5.2.1.16 parser

```
apply.parser = argparse.ArgumentParser()
```

5.2.1.17 percentile

```
apply.percentile
```

5.2.1.18 raw

```
apply.raw = normalize(tifffile.imread(raw_file), args.p_min, args.p_max)
```

5.2.1.19 raw_files

```
apply.raw_files = sorted(input_path.glob("*.tif"))
```

5.2.1.20 RCAN_hyperparameters

```
apply.RCAN_hyperparameters = ckpt["hyperparameters"]
```

5.2.1.21 required

```
apply.required
```

5.2.1.22 restored

```
apply.restored
```

Initial value:

```
=  apply(
        model,
        raw,
        RCAN_hyperparameters["input_shape"],
        RCAN_hyperparameters["input_shape"],
        RCAN_hyperparameters["num_input_channels"],
        RCAN_hyperparameters["num_output_channels"],
        batch_size=1,
        device=device,
         overlap_shape=overlap_shape,
         verbose=True,
    )
```

5.2.1.23 str

```
apply.str
```

5.2.1.24 type

```
apply.type
```

5.3 convert_omx_to_czxy Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- action
- args = parser.parse_args()
- input_dir = pathlib.Path(args.input)
- input_files = sorted(input_dir.rglob("∗.tif"))
- original = tifffile.imread(input_file)
- converted
- imagej

5.3.1 Variable Documentation

5.3.1.1 action

```
convert_omx_to_czxy.action
```

5.3.1.2 args

```
convert_omx_to_czxy.args = parser.parse_args()
```

5.3.1.3 converted

```
convert_omx_to_czxy.converted
```

Initial value:

```
=  conv_omx_to_czxy(
             original, args.num_phases, args.num_angles
        )
```

5.3.1.4 imagej

```
convert_omx_to_czxy.imagej
```

5.3.1.5 input_dir

```
convert_omx_to_czxy.input_dir = pathlib.Path(args.input)
```

5.3.1.6 input_files

```
convert_omx_to_czxy.input_files = sorted(input_dir.rglob("*.tif"))
```

5.3.1.7 int

```
convert_omx_to_czxy.int
```

5.3.1.8 original

```
convert_omx_to_czxy.original = tifffile.imread(input_file)
```

5.3.1.9 parser

```
convert_omx_to_czxy.parser = argparse.ArgumentParser()
```

5.3.1.10 required

```
convert_omx_to_czxy.required
```

5.3.1.11 str

```
convert_omx_to_czxy.str
```

5.3.1.12 type

```
convert_omx_to_czxy.type
```

5.4 convert_omx_to_paz Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- action
- args = parser.parse_args()
- input_dir = pathlib.Path(args.input)
- input_files = sorted(input_dir.rglob("∗.tif"))
- original = tifffile.imread(input_file)
- converted = conv_omx_to_paz(original, args.num_phases, args.num_angles)
- imagej

5.4.1 Variable Documentation

5.4.1.1 action

```
convert_omx_to_paz.action
```

5.4.1.2 args

```
convert_omx_to_paz.args = parser.parse_args()
```

5.4.1.3 converted

```
convert_omx_to_paz.converted = conv_omx_to_paz(original, args.num_phases, args.num_angles)
```

5.4.1.4 imagej

```
convert_omx_to_paz.imagej
```

5.4.1.5 input_dir

```
convert_omx_to_paz.input_dir = pathlib.Path(args.input)
```

5.4.1.6 input_files

```
convert_omx_to_paz.input_files = sorted(input_dir.rglob("*.tif"))
```

5.4.1.7 int

```
convert_omx_to_paz.int
```

5.4.1.8 original

```
convert_omx_to_paz.original = tifffile.imread(input_file)
```

5.4.1.9 parser

```
convert_omx_to_paz.parser = argparse.ArgumentParser()
```

5.4.1.10 required

```
convert_omx_to_paz.required
```

5.4.1.11 str

```
convert_omx_to_paz.str
```

5.4.1.12 type

```
convert_omx_to_paz.type
```

5.5 convert_slices_to_volumes Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- tuple_of_ints
- default
- args = parser.parse_args()
- input_dir = pathlib.Path(args.input)
- output_dir = pathlib.Path(args.output)
- input_files = sorted(input_dir.glob("∗.tif"))
- parents
- True
- exist_ok
- volume = np.zeros((len(input_files), 3061, 4096), dtype=np.uint8)
- input_slice = tifffile.imread(file)
- output_file = output_dir / filename
- subvolume
- imagej

5.5.1 Variable Documentation

5.5.1.1 args

```
convert_slices_to_volumes.args = parser.parse_args()
```

5.5.1.2 default

```
convert_slices_to_volumes.default
```

5.5.1.3 exist_ok

```
convert_slices_to_volumes.exist_ok
```

5.5.1.4 imagej

```
convert_slices_to_volumes.imagej
```

5.5.1.5 input_dir

```
convert_slices_to_volumes.input_dir = pathlib.Path(args.input)
```

5.5.1.6 input_files

```
convert_slices_to_volumes.input_files = sorted(input_dir.glob("*.tif"))
```

5.5.1.7 input_slice

```
convert_slices_to_volumes.input_slice = tifffile.imread(file)
```

5.5.1.8 output_dir

```
convert_slices_to_volumes.output_dir = pathlib.Path(args.output)
```

5.5.1.9 output_file

```
convert_slices_to_volumes.output_file = output_dir / filename
```

5.5.1.10 parents

```
convert_slices_to_volumes.parents
```

5.5.1.11 parser

```
convert_slices_to_volumes.parser = argparse.ArgumentParser()
```

5.5.1.12 required

```
convert_slices_to_volumes.required
```

5.5.1.13 str

```
convert_slices_to_volumes.str
```

5.5.1.14 subvolume

```
convert_slices_to_volumes.subvolume
```

5.5.1.15 True

```
convert_slices_to_volumes.True
```

5.5.1.16 tuple_of_ints

```
convert_slices_to_volumes.tuple_of_ints
```

5.5.1.17 type

```
convert_slices_to_volumes.type
```

5.5.1.18 volume

```
convert_slices_to_volumes.volume = np.zeros((len(input_files), 3061, 4096), dtype=np.uint8)
```

5.6 generate_sim Namespace Reference

Classes

- class Simulator

  The Simulator class encapsulates the state of a 3D microscope simulation.
- class SimulationRunner

  Class which performs a batch of simulations, either sequentially or in parallel.

Functions

- def arange_zero (n, spacing=1)
- def threshold_norm (sample)

  Applies a threshold and normalises the sample to improve contrast.

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- default
- args = parser.parse_args()
- runner

5.6.1 Function Documentation

5.6.1.1 arange_zero()

```
def generate_sim.arange_zero (
              n,
              spacing = 1 )
```

5.6.1.2 threshold_norm()

```
def generate_sim.threshold_norm (
              sample )
```

Applies a threshold and normalises the sample to improve contrast.

5.6.2 Variable Documentation

5.6.2.1 args

```
generate_sim.args = parser.parse_args()
```

5.6.2.2 default

```
generate_sim.default
```

5.6.2.3 int

```
generate_sim.int
```

5.6.2.4 parser

```
generate_sim.parser = argparse.ArgumentParser()
```

5.6.2.5 required

```
generate_sim.required
```

5.6.2.6 runner

```
generate_sim.runner
```

Initial value:

```
= SimulationRunner(
    args.input, args.output, range(args.start, args.end), args.z_offset
)
```

5.6.2.7 str

```
generate_sim.str
```

5.6.2.8 type

```
generate_sim.type
```

5.7 image_noising Namespace Reference

Functions

- def save_image_pair (gt_img, split, name, channel_idx)

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- choices
- float
- default
- args = parser.parse_args()
- input_path = pathlib.Path(args.input)
- output_path = pathlib.Path(args.output)
- parents
- output_train_gt_path = output_path.joinpath("Training", "GT")
- output_train_raw_path = output_path.joinpath("Training", "Raw")

- output_val_gt_path = output_path.joinpath("Validation", "GT")
- output_val_raw_path = output_path.joinpath("Validation", "Raw")
- output_test_gt_path = output_path.joinpath("Testing", "GT")
- output_test_raw_path = output_path.joinpath("Testing", "Raw")
- data = sorted(input_path.glob("∗.tif"))
- n_acquisitions = tifffile.imread(data[0]).shape[0] // args.channels
- n_img = len(data)
- train_size = int((1 - args.test_fraction) ∗ n_img)
- val_size = int(args.val_fraction ∗ train_size)
- rng = np.random.default_rng(seed=25042024)
- img_idx_all = list(range(n_img))
- img_idx_test = img_idx_all[train_size:]
- img_idx_train = img_idx_all[: train_size - val_size]
- img_idx_val = img_idx_all[train_size - val_size : train_size]
- gt = tifffile.imread(img_file)
- string split = "train"

5.7.1 Function Documentation

5.7.1.1 save_image_pair()

```
def image_noising.save_image_pair (
              gt_img,
              split,
              name,
              channel_idx )
```

5.7.2 Variable Documentation

5.7.2.1 args

```
image_noising.args = parser.parse_args()
```

5.7.2.2 choices

```
image_noising.choices
```

5.7.2.3 data

```
list image_noising.data = sorted(input_path.glob("*.tif"))
```

5.7.2.4 default

```
image_noising.default
```

5.7.2.5 float

```
image_noising.float
```

5.7.2.6 gt

```
image_noising.gt = tifffile.imread(img_file)
```

5.7.2.7 img_idx_all

```
image_noising.img_idx_all = list(range(n_img))
```

5.7.2.8 img_idx_test

```
image_noising.img_idx_test = img_idx_all[train_size:]
```

5.7.2.9 img_idx_train

```
image_noising.img_idx_train = img_idx_all[: train_size - val_size]
```

5.7.2.10 img_idx_val

```
image_noising.img_idx_val = img_idx_all[train_size - val_size : train_size]
```

5.7.2.11 input_path

```
image_noising.input_path = pathlib.Path(args.input)
```

5.7.2.12 int

```
image_noising.int
```

5.7.2.13 n_acquisitions

```
image_noising.n_acquisitions = tifffile.imread(data[0]).shape[0] // args.channels
```

5.7.2.14 n_img

```
image_noising.n_img = len(data)
```

5.7.2.15 output_path

```
image_noising.output_path = pathlib.Path(args.output)
```

5.7.2.16 output_test_gt_path

```
image_noising.output_test_gt_path = output_path.joinpath("Testing", "GT")
```

5.7.2.17 output_test_raw_path

```
image_noising.output_test_raw_path = output_path.joinpath("Testing", "Raw")
```

5.7.2.18 output_train_gt_path

```
image_noising.output_train_gt_path = output_path.joinpath("Training", "GT")
```

5.7.2.19 output_train_raw_path

```
image_noising.output_train_raw_path = output_path.joinpath("Training", "Raw")
```

5.7.2.20 output_val_gt_path

```
image_noising.output_val_gt_path = output_path.joinpath("Validation", "GT")
```

5.7.2.21 output_val_raw_path

```
image_noising.output_val_raw_path = output_path.joinpath("Validation", "Raw")
```

5.7.2.22 parents

```
image_noising.parents
```

5.7.2.23 parser

```
image_noising.parser = argparse.ArgumentParser()
```

5.7.2.24 required

```
image_noising.required
```

5.7.2.25 rng

```
image_noising.rng = np.random.default_rng(seed=25042024)
```

5.7.2.26 split

```
string image_noising.split = "train"
```

5.7.2.27 str

```
image_noising.str
```

5.7.2.28 train_size

```
image_noising.train_size = int((1 - args.test_fraction) * n_img)
```

5.7.2.29 type

```
image_noising.type
```

5.7.2.30 val_size

```
image_noising.val_size = int(args.val_fraction * train_size)
```

5.8 manage_stack Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- choices
- default
- action
- args = parser.parse_args()
- output_dir = pathlib.Path(args.output_dir)
- parents
- True
- exist_ok
- files = sorted(list(pathlib.Path(args.input_dir).glob(args.glob_str)))
- int stack_number = -1 else args.stack_number
- int number_of_stacks = len(files) // stack_number
- sample = tifffile.imread(files[0])
- stack_handler
- img_data = tifffile.imread(input_file)
- tuple filename
- tuple output_file = output_dir / filename
- output_data = img_data[j ∗ args.z_slices : (j + 1) ∗ args.z_slices]

5.8.1 Variable Documentation

5.8.1.1 action

`manage_stack.action`

5.8.1.2 args

`manage_stack.args = parser.parse_args()`

5.8.1.3 choices

`manage_stack.choices`

5.8.1.4 default

`manage_stack.default`

5.8.1.5 exist_ok

`manage_stack.exist_ok`

5.8.1.6 filename

`tuple manage_stack.filename`

Initial value:

```
=  (
            args.output_name
            + f"_stack{stack_idx*stack_number:04d}"
            + f"_{(stack_idx+1)*stack_number:04d}"
        )
```

5.8.1.7 files

```
manage_stack.files = sorted(list(pathlib.Path(args.input_dir).glob(args.glob_str)))
```

5.8.1.8 img_data

```
manage_stack.img_data = tifffile.imread(input_file)
```

5.8.1.9 int

```
manage_stack.int
```

5.8.1.10 number_of_stacks

```
int manage_stack.number_of_stacks = len(files) // stack_number
```

5.8.1.11 output_data

```
manage_stack.output_data = img_data[j * args.z_slices :  (j + 1) * args.z_slices]
```

5.8.1.12 output_dir

```
manage_stack.output_dir = pathlib.Path(args.output_dir)
```

5.8.1.13 output_file

```
string manage_stack.output_file = output_dir / filename
```

5.8.1.14 parents

```
manage_stack.parents
```

5.8.1.15 parser

```
manage_stack.parser = argparse.ArgumentParser()
```

5.8.1.16 required

```
manage_stack.required
```

5.8.1.17 sample

```
manage_stack.sample = tifffile.imread(files[0])
```

5.8.1.18 stack_handler

```
manage_stack.stack_handler
```

Initial value:

```
=  ImageStack(
             args.dimension,
             stack_number,
             stack_idx,
             sample,
             files,
             args.num_acquisitions,
       )
```

5.8.1.19 stack_number

```
int manage_stack.stack_number = -1 else args.stack_number
```

5.8.1.20 str

```
manage_stack.str
```

5.8.1.21 True

```
manage_stack.True
```

5.8.1.22 type

```
manage_stack.type
```

5.9 rcan Namespace Reference

Namespaces

- data_generator
- data_processing
- model
- plotting
- utils

5.10 rcan.data_generator Namespace Reference

Classes

- class SIM_Dataset

  Generates batches of images with real-time data augmentation.

Functions

- def load_SIM_dataset (images, shape, batch_size, transform_function, intensity_threshold, area_threshold, scale_factor, steps_per_epoch, p_min, p_max)

  Wraps SIM_Dataset object in a PyTorch Dataloader object to enable batch loading.

5.10.1 Function Documentation

5.10.1.1 load_SIM_dataset()

```
def rcan.data_generator.load_SIM_dataset (
              images,
              shape,
              batch_size,
              transform_function,
              intensity_threshold,
              area_threshold,
              scale_factor,
              steps_per_epoch,
              p_min,
              p_max )
```

Wraps SIM_Dataset object in a PyTorch Dataloader object to enable batch loading.

Parameters

images	(list[dict]) - List of dictionaries of data pairs with keys ["raw","gt"]. Images in CZXY format
shape	(tuple[int]) - Shape of batch images excluding the channel dimension
batch_size	(int) - Batch size
transform_function	(str or callable or None) - Function used for data augmentation. Typically you will set `transform_function='rotate_and_flip'` to apply combination of randomly selected image rotation and flipping. Alternatively, you can specify an arbitrary transformation function which takes two input images (source and target) and returns transformed images. If `transform_function=None`, no augmentation will be performed
intensity_threshold	(float) - If `intensity_threshold > 0`, pixels whose intensities are greater than this threshold will be considered as foreground
area_ratio_threshold	(float) - Threshold between 0 and 1. If `intensity_threshold > 0`, the generator calculates the ratio of foreground pixels in a target patch, and rejects the patch if the ratio is smaller than this threshold
scale_factor	(int) - Scale factor for the target patch size. Positive and negative values mean up- and down-scaling respectively.
steps_per_epoch	(int) - Determines how many times each image is used to generate a patch per batch
p_min	(float) - Minimum percentile used for scaling
p_max	(float) - Maximum percentile used for scaling

Returns

torch.utils.data.DataLoader object

5.11 rcan.data_processing Namespace Reference

Classes

- class ImageStack

  Handles creation and loading of image hyperstacks in order to make reconstructions using ImageJ easier.

Functions

- def crop_volume (volume, num_steps, start, step, label)

  Takes an image volume and divides part of it into smaller volumes by cropping lateral sections (the full z dimension is used).
- def conv_omx_to_czxy (original, n_phases, n_angles)

  Converts image array from OMX (PZA format) to CZXY format.
- def conv_czxy_to_omx (original, n_phases, n_angles)

  Converts image array from CZXY to OMX format.
- def conv_omx_to_paz (original, n_phases, n_angles)

  Converts image array from OMX (PZA format) to PAZ format.
- def conv_paz_to_omx (original, n_phases, n_angles)

  Converts image array from PAZ to OMX(PZA) format.

5.11.1 Function Documentation

5.11.1.1 conv_czxy_to_omx()

```
def rcan.data_processing.conv_czxy_to_omx (
              original,
              n_phases,
              n_angles )
```

Converts image array from CZXY to OMX format.

Parameters

original	(np.ndarray) - Image array in original format
n_phases	(int) - Number of phases
n_angles	(int) - Number of angles

Returns

np.ndarray Converted image array

5.11.1.2 conv_omx_to_czxy()

```
def rcan.data_processing.conv_omx_to_czxy (
              original,
              n_phases,
              n_angles )
```

Converts image array from OMX (PZA format) to CZXY format.

Parameters

original	(np.ndarray) - Image array in original format
n_phases	(int) - Number of phases
n_angles	(int) - Number of angles

Returns

np.ndarray Converted image array

5.11.1.3 conv_omx_to_paz()

```
def rcan.data_processing.conv_omx_to_paz (
              original,
              n_phases,
              n_angles )
```

Converts image array from OMX (PZA format) to PAZ format.

Parameters

original	(np.ndarray) - Image array in original format
n_phases	(int) - Number of phases
n_angles	(int) - Number of angles

Returns

np.ndarray Converted image array

5.11.1.4 conv_paz_to_omx()

```
def rcan.data_processing.conv_paz_to_omx (
              original,
              n_phases,
              n_angles )
```

Converts image array from PAZ to OMX(PZA) format.

Parameters

original	(np.ndarray) - Image array in original format
n_phases	(int) - Number of phases
n_angles	(int) - Number of angles

Returns

np.ndarray Converted image array

5.11.1.5 crop_volume()

```
def rcan.data_processing.crop_volume (
              volume,
              num_steps,
              start,
              step,
              label )
```

Takes an image volume and divides part of it into smaller volumes by cropping lateral sections (the full z dimension is used).

Parameters

volume	(np.ndarray) - image volume to crop
num_steps	(tuple[int]) - number of images in each lateral dimension (total number of subvolumes is the product)
start	(tuple[int]) - start coordinates for crop region
step	(tuple[int]) - lateral size of subvolume images
label	(str) - prefix for output file names

Returns

generator that yields image subvolumes

5.12 rcan.model Namespace Reference

Classes

- class _channel_attention_block

  Implements channel attention block/layer.
- class _residual_channel_attention_blocks

  Implements residual group based on [1].
- class RCAN

  Builds a residual channel attention network.

Functions

- def _conv (ndim, in_filters, out_filters, kernel_size, padding="same", **kwargs)

  Returns the appropriate torch.nn convolution layer based on parameters.
- def _global_average_pooling (ndim)

  Returns the appropriate torch.nn pooling layer based on parameters.
- def _standardize (x)

  Standardises input data.
- def _destandardize (x)

  Inverse of _standardize.

5.12.1 Function Documentation

5.12.1.1 _conv()

```
def rcan.model._conv (
              ndim,
              in_filters,
              out_filters,
              kernel_size,
              padding = "same",
             ** kwargs )  [private]
```

Returns the appropriate torch.nn convolution layer based on parameters.

Parameters

ndim	(int) - Specifies a 1, 2, or 3 dimensional convolution kernel
in_filters	(int) - Number of hidden input channels
out_filters	(int) - Number of hidden output channels
kernel_size	(int or tuple) Size of convolution kernel
padding	(str, optional) - Border padding strategy. Default: "same"

Returns

torch.nn.Module object of the specified type

5.12.1.2 _destandardize()

```
def rcan.model._destandardize (
              x )  [private]
```

Inverse of _standardize.

Parameters

x	(torch.Tensor) Input

Returns

torch.Tensor representing destandardised output.

5.12.1.3 _global_average_pooling()

```
def rcan.model._global_average_pooling (
              ndim )  [private]
```

Returns the appropriate torch.nn pooling layer based on parameters.

Parameters

ndim	(int) - Specifies a 2 or 3 dimensional convolution kernel

Returns

torch.nn.Module object of the specified type

5.12.1.4 _standardize()

```
def rcan.model._standardize (
              x )  [private]
```

Standardises input data.

Standardize the signal so that the range becomes [-1, 1] (assuming the original range is [0, 1]).

Parameters

x	(torch.Tensor) Input

Returns

torch.Tensor representing standardised output

5.13 rcan.plotting Namespace Reference

Functions

- def plot_learning_curve (losses_train, losses_val, psnr_train, psnr_val, ssim_train, ssim_val, figsize, output↩ _path)

  Plots the learning curve metrics from a model checkpoint according to loss, PSNR, and SSIM.
- def plot_reconstructions (device, output_path, dim, gt_imgs, raw_imgs, model_1_imgs, model_2_↩ imgs=None, cmap="inferno")

  Plots a sample of reconstructions comparing GT vs Raw vs Restored.

5.13.1 Function Documentation

5.13.1.1 plot_learning_curve()

```
def rcan.plotting.plot_learning_curve (
              losses_train,
              losses_val,
              psnr_train,
              psnr_val,
              ssim_train,
              ssim_val,
              figsize,
              output_path )
```

Plots the learning curve metrics from a model checkpoint according to loss, PSNR, and SSIM.

Parameters

losses_train	(list[float]) - List of training losses

Parameters

losses_val	(list[float]) - List of validation losses
psnr_train	(list[float]) - List of training psnrs
psnr_val	(list[float]) - List of validation psnrs
ssim_train	(list[float]) - List of training ssims
ssim_val	(list[float]) - List of validation ssims
figsize	(tuple[int]) - Specifies matplotlib layout size
output_path	(str) - Determines where plot is saved

5.13.1.2 plot_reconstructions()

```
def rcan.plotting.plot_reconstructions (
              device,
              output_path,
              dim,
              gt_imgs,
              raw_imgs,
              model_1_imgs,
              model_2_imgs = None,
              cmap = "inferno" )
```

Plots a sample of reconstructions comparing GT vs Raw vs Restored.

Parameters

device	(torch.device) - Handles the processing unit for torch
output_path	(str) - Determines where the plot is saved
dim	(int) - Dimensionality of the images
gt_imgs	(list[np.ndarray]) - List containing GT image arrays
raw_imgs	(list[np.ndarray]) - List containing Raw image arrays
model_1_imgs	(list[np.ndarray]) - List containing Step 1 image arrays
model_2_imgs	(list[np.ndarray], optional) - List containing Step 2 image arrays. Default: None
cmap	(str) - Matplotlib colormap string

5.14 rcan.utils Namespace Reference

Functions

- def normalize (image, p_min=2, p_max=99.9, dtype="float32")

  Normalizes the image intensity so that the `p_min`-th and the `p_max`-th percentiles are converted to 0 and 1 respectively.
- def apply (model, data, model_input_image_shape, model_output_image_shape, num_input_channels, num_output_channels, batch_size, device, overlap_shape=None, verbose=False)

  Applies a model to an input image.

- def load_rcan_checkpoint (ckpt_path, device)

  Enables loading of RCAN checkpointed model.
- def tuple_of_ints (string)

  Defines behaviour of parsing tuples of ints (argparse).
- def percentile (x)

  Defines behaviour of parsing percentiles (argparse).
- def reshape_to_bcwh (data)

  Reshapes 2D or 3D array to have batch x channel x width x height format, by prepending extra dimensions.
- def normalize_between_zero_and_one (data)

  Coerce pixel values to [0, 1] range.
- def compute_metrics (img, gt_img, psnr, ssim)

  Uses ignite metric objects to compute PSNR and SSIM.

5.14.1 Function Documentation

5.14.1.1 apply()

```
def rcan.utils.apply (
              model,
              data,
              model_input_image_shape,
              model_output_image_shape,
              num_input_channels,
              num_output_channels,
              batch_size,
              device,
              overlap_shape = None,
              verbose = False )
```

Applies a model to an input image.

The input image stack is split into sub-blocks with model's input size, then the model is applied block by block.

Parameters

model	(torch.nn.module) - PyTorch model
data	(array_like or list of array_like) - Input data. Either an image or a list of images
batch_size	(int) - Controls the batch size used to process image data
device	(torch.device) - PyTorch device object to specify processor to use
overlap_shape	(tuple of int or None) - Overlap size between sub-blocks in each dimension. If not specified, a default size ((32, 32) for 2D and (2, 32, 32) for 3D) is used. Results at overlapped areas are blended together linearly

Returns

np.ndarray Result image

5.14.1.2 compute_metrics()

```
def rcan.utils.compute_metrics (
              img,
              gt_img,
              psnr,
              ssim )
```

Uses ignite metric objects to compute PSNR and SSIM.

Parameters

img	(np.ndarray) - Predicted image
gt_img	(np.ndarray) - Reference image
psnr	(ignite.metrics.PSNR) - PSNR object
ssim	(ignite.metrics.SSIM) - SSIM object

Returns

dict of metric values

5.14.1.3 load_rcan_checkpoint()

```
def rcan.utils.load_rcan_checkpoint (
              ckpt_path,
              device )
```

Enables loading of RCAN checkpointed model.

Uses the `hyperparameters` key saved in checkpoint file in order to avoid the need to know the architecture specifications in advance.

Parameters

ckpt_path	(str) - filepath for checkpoint, should end in .pth
device	(torch.device) - handles processing unit for torch

Returns

tuple of checkpoint, and model with weights loaded

5.14.1.4 normalize()

```
def rcan.utils.normalize (
              image,
              p_min = 2,
              p_max = 99.9,
              dtype = "float32" )
```

Normalizes the image intensity so that the `p_min`-th and the `p_max`-th percentiles are converted to 0 and 1 respectively.

Parameters

image	(np.ndarray) - Image to apply the normalization to
p_min	(float, optional) - Percentile that is mapped to zero. Default: 2
p_max	(float, optional) - Percentile that is mapped to one. Default: 99.9
dtype	(str) - Datatype to use for the output

Returns

np.ndarray Image with transformed pixel values

5.14.1.5 References

Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy https://doi.↵org/10.1038/s41592-018-0216-7

5.14.1.6 normalize_between_zero_and_one()

```
def rcan.utils.normalize_between_zero_and_one (
              data )
```

Coerce pixel values to [0, 1] range.

Parameters

data	(np.ndarray or torch.Tensor) - image array to transform

Returns

np.ndarray or torch.Tensor transformed image array

5.14.1.7 percentile()

```
def rcan.utils.percentile (
              x )
```

Defines behaviour of parsing percentiles (argparse).

5.14.1.8 reshape_to_bcwh()

```
def rcan.utils.reshape_to_bcwh (
              data )
```

Reshapes 2D or 3D array to have batch x channel x width x height format, by prepending extra dimensions.

Parameters

data	(np.ndarray) - array to be reshaped

Returns

np.ndarray transformed data

5.14.1.9 tuple_of_ints()

```
def rcan.utils.tuple_of_ints (
              string )
```

Defines behaviour of parsing tuples of ints (argparse).

5.15 recon_postprocess Namespace Reference

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- args = parser.parse_args()
- files = sorted(list(pathlib.Path(args.input_dir).rglob("∗.tif")))
- img_data = tifffile.imread(input_file)

5.15.1 Variable Documentation

5.15.1.1 args

```
recon_postprocess.args = parser.parse_args()
```

5.15.1.2 files

```
recon_postprocess.files = sorted(list(pathlib.Path(args.input_dir).rglob("*.tif")))
```

5.15.1.3 img_data

```
tuple recon_postprocess.img_data = tifffile.imread(input_file)
```

5.15.1.4 parser

```
recon_postprocess.parser = argparse.ArgumentParser()
```

5.15.1.5 required

```
recon_postprocess.required
```

5.15.1.6 str

```
recon_postprocess.str
```

5.15.1.7 type

```
recon_postprocess.type
```

5.16 recon_preprocess Namespace Reference

Functions

- def normalize_acquisition_intensity (data, dim)

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- choices
- percentile
- default
- action
- args = parser.parse_args()
- output_dir = pathlib.Path(args.output_dir)
- parents
- True
- exist_ok
- files = sorted(list(pathlib.Path(args.input_dir).glob("∗.tif")))
- img_data = tifffile.imread(input_file).astype("float32")
- output_file = output_dir / input_file.name

5.16.1 Function Documentation

5.16.1.1 normalize_acquisition_intensity()

```
def recon_preprocess.normalize_acquisition_intensity (
              data,
              dim )
```

5.16.2 Variable Documentation

5.16.2.1 action

```
recon_preprocess.action
```

5.16.2.2 args

```
recon_preprocess.args = parser.parse_args()
```

5.16.2.3 choices

```
recon_preprocess.choices
```

5.16.2.4 default

```
recon_preprocess.default
```

5.16.2.5 exist_ok

```
recon_preprocess.exist_ok
```

5.16.2.6 files

```
recon_preprocess.files = sorted(list(pathlib.Path(args.input_dir).glob("*.tif")))
```

5.16.2.7 img_data

```
int recon_preprocess.img_data = tifffile.imread(input_file).astype("float32")
```

5.16.2.8 int

```
recon_preprocess.int
```

5.16.2.9 output_dir

```
recon_preprocess.output_dir = pathlib.Path(args.output_dir)
```

5.16.2.10 output_file

```
recon_preprocess.output_file = output_dir / input_file.name
```

5.16.2.11 parents

```
recon_preprocess.parents
```

5.16.2.12 parser

```
recon_preprocess.parser = argparse.ArgumentParser()
```

5.16.2.13 percentile

```
recon_preprocess.percentile
```

5.16.2.14 required

```
recon_preprocess.required
```

5.16.2.15 str

```
recon_preprocess.str
```

5.16.2.16 True

```
recon_preprocess.True
```

5.16.2.17 type

```
recon_preprocess.type
```

5.17 stats Namespace Reference

Functions

- def paired_t (gt_data, data)

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- int
- choices
- default
- args = parser.parse_args()
- output_dir = pathlib.Path(args.output_dir)
- parents
- True
- exist_ok
- df
- fig
- ax
- figsize
- psnr_diff_1_max
- psnr_diff_2_max
- psnr_diff_1_min
- psnr_diff_2_min
- tuple hist_range_psnr
- ssim_diff_1_max
- ssim_diff_2_max
- ssim_diff_1_min
- ssim_diff_2_min
- tuple hist_range_ssim
- xlabel
- title
- range
- color
- mean_psnr_1 = np.mean(np.array(df['psnr_model_1']) - np.array(df['psnr_raw']))
- se_psnr_1
- mean_ssim_1 = np.mean(np.array(df['ssim_model_1']) - np.array(df['ssim_raw']))
- se_ssim_1
- mean_psnr_2
- se_psnr_2
- mean_ssim_2
- se_ssim_2
- int psnr_cols = 2 else df.columns[1:3]
- int ssim_cols = 2 else df.columns[3:5]
- dflong
- dflongssim
- data
- x
- y
- hue
- dodge
- legend
- palette
- alpha
- lw

5.17.1 Function Documentation

5.17.1.1 paired_t()

```
def stats.paired_t (
              gt_data,
              data )
```

5.17.2 Variable Documentation

5.17.2.1 alpha

```
stats.alpha
```

5.17.2.2 args

```
stats.args = parser.parse_args()
```

5.17.2.3 ax

```
stats.ax
```

5.17.2.4 choices

```
stats.choices
```

5.17.2.5 color

```
stats.color
```

5.17.2.6 data

```
stats.data
```

5.17.2.7 default

```
stats.default
```

5.17.2.8 df

```
stats.df
```

Initial value:

```
=  pd.read_csv(
    pathlib.Path(args.dataset),
    index_col=False
).drop(columns="Unnamed:  0")
```

5.17.2.9 dflong

```
stats.dflong
```

Initial value:

```
=  pd.melt(
    df,
    id_vars=["file"],
    value_vars=df.columns[1:4],
    var_name="type",
    value_name="psnr"
)
```

5.17.2.10 dflongssim

```
stats.dflongssim
```

Initial value:

```
=  pd.melt(
    df,
    id_vars=["file"],
    value_vars=df.columns[4:7],
    var_name="type",
    value_name="ssim"
)
```

5.17.2.11 dodge

```
stats.dodge
```

5.17.2.12 exist_ok

```
stats.exist_ok
```

5.17.2.13 fig

```
stats.fig
```

5.17.2.14 figsize

```
stats.figsize
```

5.17.2.15 hist_range_psnr

```
tuple stats.hist_range_psnr
```

Initial value:

```
=  (
    min(psnr_diff_1_min, psnr_diff_2_min),
    max(psnr_diff_1_max, psnr_diff_2_max)
)
```

5.17.2.16 hist_range_ssim

```
tuple stats.hist_range_ssim
```

Initial value:

```
=  (
    min(ssim_diff_1_min, ssim_diff_2_min),
    max(ssim_diff_1_max, ssim_diff_2_max)
)
```

5.17.2.17 hue

```
stats.hue
```

5.17.2.18 int

```
stats.int
```

5.17.2.19 legend

```
stats.legend
```

5.17.2.20 lw

```
stats.lw
```

5.17.2.21 mean_psnr_1

```
stats.mean_psnr_1 = np.mean(np.array(df['psnr_model_1']) - np.array(df['psnr_raw']))
```

5.17.2.22 mean_psnr_2

```
stats.mean_psnr_2
```

Initial value:

```
=  np.mean(
        np.array(df['psnr_model_2']) - np.array(df['psnr_raw'])
    )
```

5.17.2.23 mean_ssim_1

```
stats.mean_ssim_1 = np.mean(np.array(df['ssim_model_1']) - np.array(df['ssim_raw']))
```

5.17.2.24 mean_ssim_2

```
stats.mean_ssim_2
```

Initial value:

```
=  np.mean(
          np.array(df['ssim_model_2']) - np.array(df['ssim_raw'])
      )
```

5.17.2.25 output_dir

```
stats.output_dir = pathlib.Path(args.output_dir)
```

5.17.2.26 palette

```
stats.palette
```

5.17.2.27 parents

```
stats.parents
```

5.17.2.28 parser

```
stats.parser = argparse.ArgumentParser()
```

5.17.2.29 psnr_cols

```
int stats.psnr_cols = 2 else df.columns[1:3]
```

5.17.2.30 psnr_diff_1_max

```
stats.psnr_diff_1_max
```

Initial value:

```
=  np.max(
     np.array(df["psnr_model_1"]) - np.array(df["psnr_raw"])
 )
```

5.17.2.31 psnr_diff_1_min

```
stats.psnr_diff_1_min
```

Initial value:
```
=  np.min(
     np.array(df["psnr_model_1"]) - np.array(df["psnr_raw"])
)
```

5.17.2.32 psnr_diff_2_max

```
stats.psnr_diff_2_max
```

Initial value:
```
=  np.max(
     np.array(df["psnr_model_2"]) - np.array(df["psnr_raw"])
)
```

5.17.2.33 psnr_diff_2_min

```
stats.psnr_diff_2_min
```

Initial value:
```
=  np.min(
     np.array(df["psnr_model_2"]) - np.array(df["psnr_raw"])
)
```

5.17.2.34 range

```
stats.range
```

5.17.2.35 required

```
stats.required
```

5.17.2.36 se_psnr_1

```
stats.se_psnr_1
```

Initial value:
```
=  np.std(
     np.array(df['psnr_model_1']) - np.array(df['psnr_raw']), ddof=1
)/np.sqrt(len(df['psnr_model_1']))
```

5.17.2.37 se_psnr_2

```
stats.se_psnr_2
```

Initial value:

```
=  np.std(
        np.array(df['psnr_model_2']) - np.array(df['psnr_raw']), ddof=1
    )/np.sqrt(len(df['psnr_model_2']))
```

5.17.2.38 se_ssim_1

```
stats.se_ssim_1
```

Initial value:

```
=  np.std(
    np.array(df['ssim_model_1']) - np.array(df['ssim_raw']), ddof=1
)/np.sqrt(len(df['ssim_model_1']))
```

5.17.2.39 se_ssim_2

```
stats.se_ssim_2
```

Initial value:

```
=  np.std(
        np.array(df['ssim_model_2']) - np.array(df['ssim_raw']), ddof=1
    )/np.sqrt(len(df['ssim_model_2']))
```

5.17.2.40 ssim_cols

```
int stats.ssim_cols = 2 else df.columns[3:5]
```

5.17.2.41 ssim_diff_1_max

```
stats.ssim_diff_1_max
```

Initial value:

```
=  np.max(
    np.array(df["ssim_model_1"]) - np.array(df["ssim_raw"])
)
```

5.17.2.42 ssim_diff_1_min

```
stats.ssim_diff_1_min
```

Initial value:

```
=  np.min(
     np.array(df["ssim_model_1"]) - np.array(df["ssim_raw"])
 )
```

5.17.2.43 ssim_diff_2_max

```
stats.ssim_diff_2_max
```

Initial value:

```
=  np.max(
     np.array(df["ssim_model_2"]) - np.array(df["ssim_raw"])
 )
```

5.17.2.44 ssim_diff_2_min

```
stats.ssim_diff_2_min
```

Initial value:

```
=  np.min(
     np.array(df["ssim_model_2"]) - np.array(df["ssim_raw"])
 )
```

5.17.2.45 str

```
stats.str
```

5.17.2.46 title

```
stats.title
```

5.17.2.47 True

```
stats.True
```

5.17.2.48 type

```
stats.type
```

5.17.2.49 x

```
stats.x
```

5.17.2.50 xlabel

```
stats.xlabel
```

5.17.2.51 y

```
stats.y
```

5.18 synthetic_sim Namespace Reference

Namespaces

- otf

5.19 synthetic_sim.otf Namespace Reference

Classes

- class PsfParameters

  Class to store PSF parameters.

Functions

- def calc_psf (params)

  Calculate an approximate Gibson-Lanni PSF based on the parameters provided.

5.19.1 Function Documentation

5.19.1.1 calc_psf()

```
def synthetic_sim.otf.calc_psf (
              params )
```

Calculate an approximate Gibson-Lanni PSF based on the parameters provided.

Code ported from MATLAB, original copyright Jizhou Li, 2016, The Chinese University of Hong Kong.

Parameters

params	(PsfParameters) - dataclass storing the PSF parameters

Returns

np.ndarray representing the PSF

5.20 train Namespace Reference

Functions

- def load_data_paths (config, data_type)
- def train (train_loader, val_loader, optimizer, scheduler, net, batchsize, n_accumulations, saveinterval, nepoch, start_epoch=0, losses_train_epoch=[ ], losses_val_epoch=[ ], psnr_train_epoch=[ ], psnr_val_epoch=[ ], ssim_train_epoch=[ ], ssim_val_epoch=[ ])

Variables

- parser = argparse.ArgumentParser()
- type
- str
- required
- args = parser.parse_args()
- dictionary schema
- config = json.load(f)
- int ndim = tifffile.imread(training_data[0]["raw"]).ndim - 1
- input_shape = config["input_shape"]
- tuple device
- ckpt_path = None if args.model_ckpt is None else pathlib.Path(args.model_ckpt)
- model
- dictionary RCAN_hyperparameters
- ckpt
- train_loader
- val_loader
- optimizer
- scheduler
- output_dir = pathlib.Path(args.output_dir)
- parents
- True
- exist_ok
- n_accumulations
- saveinterval
- nepoch
- start_epoch
- losses_train_epoch
- losses_val_epoch
- psnr_train_epoch
- psnr_val_epoch
- ssim_train_epoch
- ssim_val_epoch

5.20.1 Function Documentation

5.20.1.1 load_data_paths()

```
def train.load_data_paths (
              config,
              data_type )
```

5.20.1.2 train()

```
def train.train (
              train_loader,
              val_loader,
              optimizer,
              scheduler,
              net,
              batchsize,
              n_accumulations,
              saveinterval,
              nepoch,
              start_epoch = 0,
              losses_train_epoch = [],
              losses_val_epoch = [],
              psnr_train_epoch = [],
              psnr_val_epoch = [],
              ssim_train_epoch = [],
              ssim_val_epoch = [] )
```

5.20.2 Variable Documentation

5.20.2.1 args

```
train.args = parser.parse_args()
```

5.20.2.2 ckpt

```
train.ckpt
```

5.20.2.3 ckpt_path

```
train.ckpt_path = None if args.model_ckpt is None else pathlib.Path(args.model_ckpt)
```

5.20.2.4 config

```
train.config = json.load(f)
```

5.20.2.5 device

```
tuple train.device
```

Initial value:

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

5.20.2.6 exist_ok

```
train.exist_ok
```

5.20.2.7 input_shape

```
tuple train.input_shape = config["input_shape"]
```

5.20.2.8 losses_train_epoch

```
train.losses_train_epoch
```

5.20.2.9 losses_val_epoch

```
train.losses_val_epoch
```

5.20.2.10 model

```
train.model
```

Initial value:

```
=  RCAN(
         input_shape,
         num_input_channels=config["num_input_channels"],
         num_hidden_channels=config["num_hidden_channels"],
         num_residual_blocks=config["num_residual_blocks"],
         num_residual_groups=config["num_residual_groups"],
         channel_reduction=config["channel_reduction"],
         residual_scaling=1.0,
         num_output_channels=config["num_output_channels"],
     )
```

5.20.2.11 n_accumulations

```
train.n_accumulations
```

5.20.2.12 ndim

```
int train.ndim = tifffile.imread(training_data[0]["raw"]).ndim - 1
```

5.20.2.13 nepoch

```
train.nepoch
```

5.20.2.14 optimizer

```
train.optimizer
```

Initial value:

```
=  torch.optim.Adam(
     model.parameters(), lr=config["initial_learning_rate"]
 )
```

5.20.2.15 output_dir

```
train.output_dir = pathlib.Path(args.output_dir)
```

5.20.2.16 parents

```
train.parents
```

5.20.2.17 parser

```
train.parser = argparse.ArgumentParser()
```

5.20.2.18 psnr_train_epoch

```
train.psnr_train_epoch
```

5.20.2.19 psnr_val_epoch

```
train.psnr_val_epoch
```

5.20.2.20 RCAN_hyperparameters

```
train.RCAN_hyperparameters
```

Initial value:

```
=  {
        "input_shape":  input_shape,
        "num_input_channels":  config["num_input_channels"],
        "num_hidden_channels":  config["num_hidden_channels"],
        "num_residual_blocks":  config["num_residual_blocks"],
        "num_residual_groups":  config["num_residual_groups"],
        "channel_reduction":  config["channel_reduction"],
        "residual_scaling":  1.0,
        "num_output_channels":  config["num_output_channels"],
    }
```

5.20.2.21 required

```
train.required
```

5.20.2.22 saveinterval

```
train.saveinterval
```

5.20.2.23 scheduler

```
train.scheduler
```

Initial value:

```
=  torch.optim.lr_scheduler.StepLR(
     optimizer, step_size=config["epochs"] // 4, gamma=config["lr_decay"]
 )
```

5.20.2.24 schema

```
dictionary train.schema
```

5.20.2.25 ssim_train_epoch

```
train.ssim_train_epoch
```

5.20.2.26 ssim_val_epoch

```
train.ssim_val_epoch
```

5.20.2.27 start_epoch

```
train.start_epoch
```

5.20.2.28 str

```
train.str
```

5.20.2.29 train_loader

```
train.train_loader
```

Initial value:

```
=  load_SIM_dataset(
    training_data,
    input_shape,
    batch_size=config["batch_size"],
    transform_function=(
        "rotate_and_flip" if config["data_augmentation"] else None
    ),
    intensity_threshold=config["intensity_threshold"],
    area_threshold=config["area_ratio_threshold"],
    scale_factor=1,
    steps_per_epoch=config["steps_per_epoch"],
    p_min=config["p_min"],
    p_max=config["p_max"],
)
```

5.20.2.30 True

```
train.True
```

5.20.2.31 type

```
train.type
```

5.20.2.32 val_loader

```
train.val_loader
```

Initial value:

```
=  load_SIM_dataset(
        validation_data,
        input_shape,
        batch_size=config["batch_size"],
        transform_function=(
            "rotate_and_flip" if config["data_augmentation"] else None
        ),
        intensity_threshold=config["intensity_threshold"],
        area_threshold=config["area_ratio_threshold"],
        scale_factor=1,
        steps_per_epoch=config["steps_per_epoch"],
        p_min=config["p_min"],
        p_max=config["p_max"],
    )
```

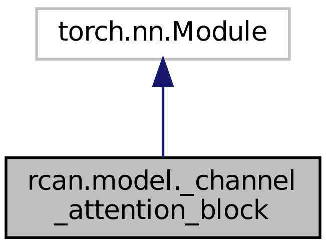
Chapter 6

Class Documentation

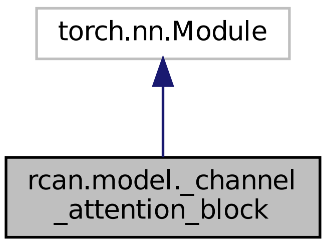
6.1 rcan.model._channel_attention_block Class Reference

Implements channel attention block/layer.

Inheritance diagram for rcan.model._channel_attention_block:



Collaboration diagram for rcan.model._channel_attention_block:



Public Member Functions

- def __init__ (self, ndim, num_channels, reduction=16)

  Initialises class.
- def forward (self, x)

  Forward method for class.

Public Attributes

- global_average_pooling
- conv_1
- conv_2

6.1.1 Detailed Description

Implements channel attention block/layer.

Instantiates a simple attention mechanism which pools all spatial information in each channel, and computes channel attention weights through a series of linear transformations and activation layers. Builds part of the architecture originally presented in [1]. Software implementation based on [2].

6.1.1.1 References

[1] Image Super-Resolution Using Very Deep Residual Channel Attention Networks `https://arxiv.↩ org/abs/1807.02758` [2] Fast, multicolour optical sectioning over extended fields of view by combining interferometric SIM with machine learning `https://doi.org/10.1364/BOE.510912` (Implementation based on CALayer from the paper's source code: `https://github.com/edward-n-ward/ML-OS-↩ SIM/blob/master/RCAN/Training%20code/models.py`)

6.1.2 Constructor & Destructor Documentation

6.1.2.1 __init__()

```
def rcan.model._channel_attention_block.__init__ (
              self,
              ndim,
              num_channels,
              reduction = 16 )
```

Initialises class.

Parameters

ndim	(int) - Feature dimensionality
num_channels	(int) - Number of hidden channels
reduction	(int, optional) - Factor to reduce the number of channels by during the attention weight computation. Default: 16.

6.1.3 Member Function Documentation

6.1.3.1 forward()

```
def rcan.model._channel_attention_block.forward (
              self,
              x )
```

Forward method for class.

Parameters

x	(torch.Tensor) Input

Returns

torch.Tensor representing x multiplied by attention weights across channels.

6.1.4 Member Data Documentation

6.1.4.1 conv_1

```
rcan.model._channel_attention_block.conv_1
```

6.1.4.2 conv_2

```
rcan.model._channel_attention_block.conv_2
```

6.1.4.3 global_average_pooling

```
rcan.model._channel_attention_block.global_average_pooling
```

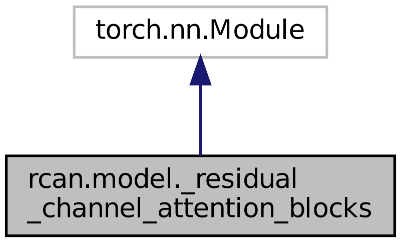
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/sim_project/jh2284/src/rcan/model.py

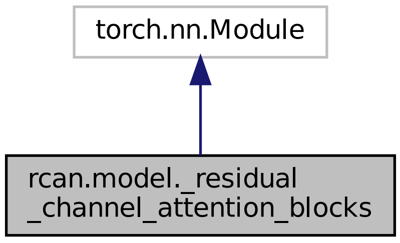
6.2 rcan.model._residual_channel_attention_blocks Class Reference

Implements residual group based on [1].

Inheritance diagram for rcan.model._residual_channel_attention_blocks:



Collaboration diagram for rcan.model._residual_channel_attention_blocks:



Public Member Functions

- def __init__ (self, ndim, num_channels, repeat=1, channel_reduction=8, residual_scaling=1.0)

  Initialises object.
- def forward (self, x)

  Forward method for class.

Public Attributes

- repeat
- residual_scaling
- conv_list
- channel_attention_block_list

6.2.1 Detailed Description

Implements residual group based on [1].

6.2.1.1 References

[1] Fast, multicolour optical sectioning over extended fields of view by combining interferometric SIM with machine learning `https://doi.org/10.1364/BOE.510912` (Implementation based on ResidualGroup from the paper's source code: `https://github.com/edward-n-ward/ML-OS-SIM/blob/master/↩ RCAN/Training%20code/models.py`)

6.2.2 Constructor & Destructor Documentation

6.2.2.1 __init__()

```
def rcan.model._residual_channel_attention_blocks.__init__ (
              self,
              ndim,
              num_channels,
              repeat = 1,
              channel_reduction = 8,
              residual_scaling = 1.0 )
```

Initialises object.

Parameters

ndim	(int) - Spatial dimension of input features
num_channels	(int) - Number of hidden channels
repeat	(int) - Number of residual blocks in group
channel_reduction	(int) - Channel reduction during attention mechanism
residual_scaling	(float) - output multiplier before residual connection

6.2.3 Member Function Documentation

6.2.3.1 forward()

```
def rcan.model._residual_channel_attention_blocks.forward (
              self,
              x )
```

Forward method for class.

Parameters

x	(torch.Tensor) - Input values

Returns

torch.Tensor representing output values

6.2.4 Member Data Documentation

6.2.4.1 channel_attention_block_list

```
rcan.model._residual_channel_attention_blocks.channel_attention_block_list
```

6.2.4.2 conv_list

```
rcan.model._residual_channel_attention_blocks.conv_list
```

6.2.4.3 repeat

```
rcan.model._residual_channel_attention_blocks.repeat
```

6.2.4.4 residual_scaling

```
rcan.model._residual_channel_attention_blocks.residual_scaling
```

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

- /home/jhughes2712/projects/sim_project/jh2284/src/rcan/model.py

6.3 rcan.data_processing.ImageStack Class Reference

Handles creation and loading of image hyperstacks in order to make reconstructions using ImageJ easier.

Public Member Functions

- def __init__ (self, dim, stack_number, stack_idx, sample, files, n_acq)

  Initialises class.
- def add_image (self, img_data, i)

  Adds an image to the initialised stack.
- def export_stack (self)

  Returns the stack.

Public Attributes

- dim
- n_acq
- sample
- stack
- n_z

6.3.1 Detailed Description

Handles creation and loading of image hyperstacks in order to make reconstructions using ImageJ easier.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 __init__()

```
def rcan.data_processing.ImageStack.__init__ (
                self,
                dim,
                stack_number,
                stack_idx,
                sample,
                files,
                n_acq )
```

Initialises class.

Parameters

dim	(int) - Dimension of images
stack_number	(int) - Number of images in the stack
stack_idx	(int) - The index of the stack within the set of stacks for the files list
sample	(np.ndarray) - Image from the directory which enables correct image stack shape/dtype and error catching
files	(list) - List of all files in directory
n_acq	(int) - Number of SIM acquisitions in the images

6.3.3 Member Function Documentation

6.3.3.1 add_image()

```
def rcan.data_processing.ImageStack.add_image (
              self,
              img_data,
              i )
```

Adds an image to the initialised stack.

Parameters

img_data	(np.ndarray) - Image to be added
i	(int) - Index of the image in the stack

6.3.3.2 export_stack()

```
def rcan.data_processing.ImageStack.export_stack (
              self )
```

Returns the stack.

Returns

np.ndarray

6.3.4 Member Data Documentation

6.3.4.1 dim

```
rcan.data_processing.ImageStack.dim
```

6.3.4.2 n_acq

```
rcan.data_processing.ImageStack.n_acq
```

6.3.4.3 n_z

```
rcan.data_processing.ImageStack.n_z
```

6.3.4.4 sample

```
rcan.data_processing.ImageStack.sample
```

6.3.4.5 stack

```
rcan.data_processing.ImageStack.stack
```

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

- /home/jhughes2712/projects/sim_project/jh2284/src/rcan/data_processing.py

6.4 synthetic_sim.otf.PsfParameters Class Reference

Class to store PSF parameters.

Static Public Attributes

- int
- float
- Callable

6.4.1 Detailed Description

Class to store PSF parameters.

Class to store the parameters used to evaluate an approximate Gibson-Lanni PSF. Default values are provided except for the PSF size.

6.4.2 Member Data Documentation

6.4.2.1 Callable

```
synthetic_sim.otf.PsfParameters.Callable  [static]
```

6.4.2.2 float

```
synthetic_sim.otf.PsfParameters.float  [static]
```

6.4.2.3 int

```
synthetic_sim.otf.PsfParameters.int  [static]
```

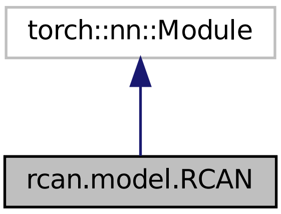
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/sim_project/jh2284/src/synthetic_sim/otf.py

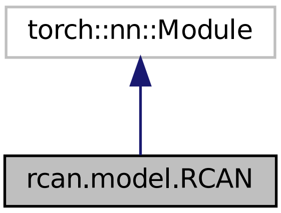
6.5 rcan.model.RCAN Class Reference

Builds a residual channel attention network.

Inheritance diagram for rcan.model.RCAN:



Collaboration diagram for rcan.model.RCAN:



Public Member Functions

- def __init__ (self, input_shape=(16, 256, 256), ∗num_input_channels=9, num_hidden_channels=32, num↩ _residual_blocks=3, num_residual_groups=5, channel_reduction=8, residual_scaling=1.0, num_output_↩ channels=-1)

  Initialises object.
- def forward (self, x)

  Forward method for class.

Public Attributes

- num_residual_groups
- rcab_list
- conv_input
- conv_list
- conv_output

6.5.1 Detailed Description

Builds a residual channel attention network.

Note that the upscale module at the end of the network is omitted so that the input and output of the model have the same size.

6.5.1.1 References

[1] Image Super-Resolution Using Very Deep Residual Channel Attention Networks `https://arxiv.↩ org/abs/1807.02758` [2] Fast, multicolour optical sectioning over extended fields of view by combining interferometric SIM with machine learning `https://doi.org/10.1364/BOE.510912` (Implementation based on RCAN from the paper's source code: `https://github.com/edward-n-ward/ML-OS-↩ SIM/blob/master/RCAN/Training%20code/models.py`)

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
def rcan.model.RCAN.__init__ (
              self,
              input_shape = (16, 256, 256),
             ∗ num_input_channels = 9,
              num_hidden_channels = 32,
              num_residual_blocks = 3,
              num_residual_groups = 5,
              channel_reduction = 8,
              residual_scaling = 1.0,
              num_output_channels = -1 )
```

Initialises object.

Builds a residual channel attention network. Note that the upscale module at the end of the network is omitted so that the input and output of the model have the same size.

Parameters

input_shape	(tuple[int]) - Input shape of the model.
num_channels	(int) - Number of feature channels.
num_residual_blocks	(int) - Number of residual channel attention blocks in each residual group.
num_residual_groups	(int) - Number of residual groups.
channel_reduction	(int) - Channel reduction ratio for channel attention.
residual_scaling	(float) - Scaling factor applied to the residual component in the residual channel attention block.
num_output_channels	(int) - Number of channels in the output image. if negative, it is set to the same number as the input.

Returns

torch.nn.Module PyTorch model instance.

6.5.3 Member Function Documentation

6.5.3.1 forward()

```
def rcan.model.RCAN.forward (
              self,
              x )
```

Forward method for class.

Parameters

x	(torch.Tensor) - Input

Returns

torch.Tensor Output

6.5.4 Member Data Documentation

6.5.4.1 conv_input

```
rcan.model.RCAN.conv_input
```

6.5.4.2 conv_list

```
rcan.model.RCAN.conv_list
```

6.5.4.3 conv_output

```
rcan.model.RCAN.conv_output
```

6.5.4.4 num_residual_groups

```
rcan.model.RCAN.num_residual_groups
```

6.5.4.5 rcab_list

```
rcan.model.RCAN.rcab_list
```

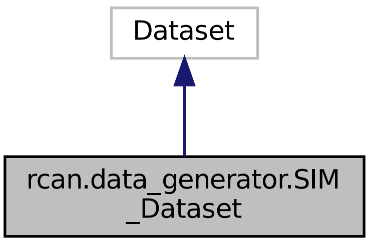
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/sim_project/jh2284/src/rcan/model.py

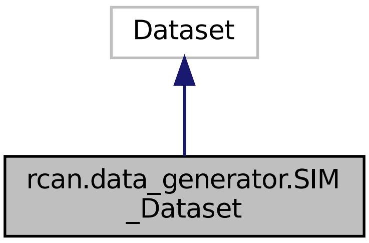
6.6 rcan.data_generator.SIM_Dataset Class Reference

Generates batches of images with real-time data augmentation.

Inheritance diagram for rcan.data_generator.SIM_Dataset:



Collaboration diagram for rcan.data_generator.SIM_Dataset:



Public Member Functions

- def __init__ (self, images, shape, transform_function="rotate_and_flip", intensity_threshold=0.0, area_ratio↩_threshold=0.0, scale_factor=1, steps_per_epoch=1, p_min=2.0, p_max=99.9)

  Initialises object.
- def __getitem__ (self, j)

  Method used during batch loading.
- def __len__ (self)

Public Attributes

- steps_per_epoch
- p_min
- p_max
- output_shape
- output_signature

Private Member Functions

- def _scale (self, shape)

Private Attributes

- _shape
- _transform_function
- _intensity_threshold
- _area_threshold
- _scale_factor
- _y

6.6.1 Detailed Description

Generates batches of images with real-time data augmentation.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

```
def rcan.data_generator.SIM_Dataset.__init__ (
              self,
              images,
              shape,
              transform_function = "rotate_and_flip",
              intensity_threshold = 0.0,
              area_ratio_threshold = 0.0,
              scale_factor = 1,
              steps_per_epoch = 1,
              p_min = 2.0,
              p_max = 99.9 )
```

Initialises object.

Parameters

images	(list[dict]) - List of dictionaries of data pairs with keys ["raw","gt"]. Images in CZXY format
shape	(tuple[int]) - Shape of batch images excluding the channel dimension
transform_function	(str or callable, optional) - Function used for data augmentation. Typically you will set `transform_function='rotate_and_flip'` to apply combination of randomly selected image rotation and flipping. Alternatively, you can specify an arbitrary transformation function which takes two input images (source and target) and returns transformed images. If `transform_function=None`, no augmentation will be performed. Default: "rotate_and_flip"
intensity_threshold	(float, optional) - If `intensity_threshold > 0`, pixels whose intensities are greater than this threshold will be considered as foreground. Default: 0.0
area_ratio_threshold	(float, optional) - Threshold between 0 and 1. If `intensity_threshold > 0`, the generator calculates the ratio of foreground pixels in a target patch, and rejects the patch if the ratio is smaller than this threshold. Default: 0.0
scale_factor	(int, optional) - Scale factor for the target patch size. Positive and negative values mean up- and down-scaling respectively. Default: 1
steps_per_epoch	(int, optional) - Determines how many times each image is used to generate a patch per batch. Default: 1
p_min	(float, optional) - Minimum percentile used for scaling. Default: 2.0
p_max	(float, optional) - Maximum percentile used for scaling. Default: 99.9

6.6.3 Member Function Documentation

6.6.3.1 __getitem__()

```
def rcan.data_generator.SIM_Dataset.__getitem__ (
              self,
              j )
```

Method used during batch loading.

Standardises pixel values and takes patches from the image pair. Also implements the rejection of patches based on area/intensity threshold, if `self._intensity_threshold > 0`. Augments data pair.

Parameters

j	(int) - Index of data to be loaded. Note that if `self.steps_per_epoch > 1`, this can be more than the dataset size, in which case it is interpreted modulo the dataset size.

Returns

tuple(torch.Tensor) raw-gt image pair

6.6.3.2 __len__()

```
def rcan.data_generator.SIM_Dataset.__len__ (
              self )
```

6.6.3.3 _scale()

```
def rcan.data_generator.SIM_Dataset._scale (
              self,
              shape )  [private]
```

6.6.4 Member Data Documentation

6.6.4.1 _area_threshold

```
rcan.data_generator.SIM_Dataset._area_threshold  [private]
```

6.6.4.2 _intensity_threshold

```
rcan.data_generator.SIM_Dataset._intensity_threshold  [private]
```

6.6.4.3 _scale_factor

```
rcan.data_generator.SIM_Dataset._scale_factor  [private]
```

6.6.4.4 _shape

```
rcan.data_generator.SIM_Dataset._shape  [private]
```

6.6.4.5 _transform_function

```
rcan.data_generator.SIM_Dataset._transform_function  [private]
```

6.6.4.6 _y

```
rcan.data_generator.SIM_Dataset._y  [private]
```

6.6.4.7 output_shape

```
rcan.data_generator.SIM_Dataset.output_shape
```

6.6.4.8 output_signature

```
rcan.data_generator.SIM_Dataset.output_signature
```

6.6.4.9 p_max

```
rcan.data_generator.SIM_Dataset.p_max
```

6.6.4.10 p_min

```
rcan.data_generator.SIM_Dataset.p_min
```

6.6.4.11 steps_per_epoch

```
rcan.data_generator.SIM_Dataset.steps_per_epoch
```

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

- /home/jhughes2712/projects/sim_project/jh2284/src/rcan/data_generator.py

6.7 generate_sim.SimulationRunner Class Reference

Class which performs a batch of simulations, either sequentially or in parallel.

Public Member Functions

- def __init__ (self, input_dir, output_dir, index_range, z_offset)
- def do_sim (self, i, sim, vol)

  Creates a new random virtual microscope simulator, takes a new sample from the VHP dataset, runs the simulation on the sample, and saves the results, along with the ground truth, in a single TIFF file.
- def run (self)

  Runs a series of simulations sequentially.

Public Attributes

- input_dir
- input_files
- output_dir
- range
- z_offset

6.7.1 Detailed Description

Class which performs a batch of simulations, either sequentially or in parallel.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 __init__()

```
def generate_sim.SimulationRunner.__init__ (
              self,
              input_dir,
              output_dir,
              index_range,
              z_offset )
```

6.7.3 Member Function Documentation

6.7.3.1 do_sim()

```
def generate_sim.SimulationRunner.do_sim (
              self,
              i,
              sim,
              vol )
```

Creates a new random virtual microscope simulator, takes a new sample from the VHP dataset, runs the simulation on the sample, and saves the results, along with the ground truth, in a single TIFF file.

The parameters are saved in an accompanying JSON file.

6.7.3.2 run()

```
def generate_sim.SimulationRunner.run (
              self )
```

Runs a series of simulations sequentially.

6.7.4 Member Data Documentation

6.7.4.1 input_dir

```
generate_sim.SimulationRunner.input_dir
```

6.7.4.2 input_files

```
generate_sim.SimulationRunner.input_files
```

6.7.4.3 output_dir

```
generate_sim.SimulationRunner.output_dir
```

6.7.4.4 range

```
generate_sim.SimulationRunner.range
```

6.7.4.5 z_offset

```
generate_sim.SimulationRunner.z_offset
```

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

- /home/jhughes2712/projects/sim_project/jh2284/src/generate_sim.py

6.8 generate_sim.Simulator Class Reference

The Simulator class encapsulates the state of a 3D microscope simulation.

Public Member Functions

- def __init__ (self, **kwargs)
- def randomise (self)
- def params_dict (self)
- def psf_params (self)
- def wavevectors (self)

    Calculates wavevectors inside the sample for the three beams, for a given number of rotations of those beams.
- def illumination (self)

    Calculates the illumination intensity in the sample; returns ndarray of shape (n_rotations, n_shifts, n_x, n_x, n_z)
- def in_focus_plane (self, sample)

    Returns the designated 'ground truth' plane.
- def psf (self)

    Calculates a PSF if it has not been done already.
- def simulate_sim (self, sample)

    Calculates the 15 simulated SIM images for a given sample.
- def simulate_ideal_superres (self, sample)

    Simulates the best-case scenario for a 3D SIM reconstruction, by convolving the in-focus plane with a small PSF.
- def add_noise (self, image)

    Adds a combination of Gaussian and Poissonian noise to the image.

Public Attributes

- n_shifts
- n_angles
- n_x
- n_z
- n_rotations
- res_axial
- res_lateral
- delta_z_p
- n_sample
- n_i
- n_g
- z
- z_p
- angle_error
- poisson_photons
- signal_to_noise
- lambda0
- k0
- lambda_exc
- k_exc
- beam_position

Private Attributes

- _psf
- _superres_psf
- _illumination

6.8.1 Detailed Description

The Simulator class encapsulates the state of a 3D microscope simulation.

A single instance of this class corresponds to a specific set of microscope parameters. These parameters are randomly chosen upon object creation.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
def generate_sim.Simulator.__init__ (
                self,
               ** kwargs )
```

6.8.3 Member Function Documentation

6.8.3.1 add_noise()

```
def generate_sim.Simulator.add_noise (
              self,
              image )
```

Adds a combination of Gaussian and Poissonian noise to the image.

6.8.3.2 illumination()

```
def generate_sim.Simulator.illumination (
              self )
```

Calculates the illumination intensity in the sample; returns ndarray of shape (n_rotations, n_shifts, n_x, n_x, n_z)

6.8.3.3 in_focus_plane()

```
def generate_sim.Simulator.in_focus_plane (
              self,
              sample )
```

Returns the designated ‘ground truth’ plane.

6.8.3.4 params_dict()

```
def generate_sim.Simulator.params_dict (
              self )
```

6.8.3.5 psf()

```
def generate_sim.Simulator.psf (
              self )
```

Calculates a PSF if it has not been done already.

6.8.3.6 psf_params()

```
def generate_sim.Simulator.psf_params (
              self )
```

6.8.3.7 randomise()

```
def generate_sim.Simulator.randomise (
              self )
```

6.8.3.8 simulate_ideal_superres()

```
def generate_sim.Simulator.simulate_ideal_superres (
              self,
              sample )
```

Simulates the best-case scenario for a 3D SIM reconstruction, by convolving the in-focus plane with a small PSF.

6.8.3.9 simulate_sim()

```
def generate_sim.Simulator.simulate_sim (
              self,
              sample )
```

Calculates the 15 simulated SIM images for a given sample.

6.8.3.10 wavevectors()

```
def generate_sim.Simulator.wavevectors (
              self )
```

Calculates wavevectors inside the sample for the three beams, for a given number of rotations of those beams.

Returns ndarray of shape (n_rotations, n_beams, 3), where n_beams = 3

6.8.4 Member Data Documentation

6.8.4.1 _illumination

```
generate_sim.Simulator._illumination  [private]
```

6.8.4.2 _psf

```
generate_sim.Simulator._psf  [private]
```

6.8.4.3 _superres_psf

```
generate_sim.Simulator._superres_psf  [private]
```

6.8.4.4 angle_error

```
generate_sim.Simulator.angle_error
```

6.8.4.5 beam_position

```
generate_sim.Simulator.beam_position
```

6.8.4.6 delta_z_p

```
generate_sim.Simulator.delta_z_p
```

6.8.4.7 k0

```
generate_sim.Simulator.k0
```

6.8.4.8 k_exc

```
generate_sim.Simulator.k_exc
```

6.8.4.9 lambda0

```
generate_sim.Simulator.lambda0
```

6.8.4.10 lambda_exc

```
generate_sim.Simulator.lambda_exc
```

6.8.4.11 n_angles

```
generate_sim.Simulator.n_angles
```

6.8.4.12 n_g

```
generate_sim.Simulator.n_g
```

6.8.4.13 n_i

```
generate_sim.Simulator.n_i
```

6.8.4.14 n_rotations

```
generate_sim.Simulator.n_rotations
```

6.8.4.15 n_sample

```
generate_sim.Simulator.n_sample
```

6.8.4.16 n_shifts

```
generate_sim.Simulator.n_shifts
```

6.8.4.17 n_x

```
generate_sim.Simulator.n_x
```

6.8.4.18 n_z

```
generate_sim.Simulator.n_z
```

6.8.4.19 poisson_photons

```
generate_sim.Simulator.poisson_photons
```

6.8.4.20 res_axial

```
generate_sim.Simulator.res_axial
```

6.8.4.21 res_lateral

```
generate_sim.Simulator.res_lateral
```

6.8.4.22 signal_to_noise

```
generate_sim.Simulator.signal_to_noise
```

6.8.4.23 z

```
generate_sim.Simulator.z
```

6.8.4.24 z_p

```
generate_sim.Simulator.z_p
```

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

- /home/jhughes2712/projects/sim_project/jh2284/src/generate_sim.py

Chapter 7

File Documentation

7.1 /home/jhughes2712/projects/sim_project/jh2284/src/analyse.py File Reference

Script producing plots and small datasets that summarise the performance of models.

Namespaces

- analyse

Variables

- analyse.parser = argparse.ArgumentParser()
- analyse.type
- analyse.str
- analyse.required
- analyse.default
- analyse.int
- analyse.action
- analyse.args = parser.parse_args()
- analyse.output_dir = pathlib.Path(args.output_dir)
- analyse.parents
- analyse.True
- analyse.exist_ok
- tuple analyse.device
- analyse.ckpt
- analyse.model
- analyse.gt_dir = pathlib.Path(args.gt_dir)
- analyse.raw_dir = pathlib.Path(args.raw_dir)
- analyse.model_1_dir = pathlib.Path(args.model_1_dir)
- analyse.gt_files = sorted(list(gt_dir.glob(args.glob_str)))
- analyse.raw_files = sorted(list(raw_dir.glob(args.glob_str)))
- analyse.model_1_files = sorted(list(model_1_dir.glob(args.glob_str)))
- analyse.model_2_dir = pathlib.Path(args.model_2_dir)
- analyse.model_2_files = sorted(list(model_2_dir.glob(args.glob_str)))
- analyse.N = len(gt_files)

- analyse.psnr = PSNR(data_range=65536, device=device)
- analyse.ssim
- analyse.df
- analyse.gt = reshape_to_bcwh(tifffile.imread(gt_files[i]))
- analyse.raw = reshape_to_bcwh(tifffile.imread(raw_files[i]))
- analyse.model_1 = reshape_to_bcwh(tifffile.imread(model_1_files[i]))
- analyse.model_2 = reshape_to_bcwh(tifffile.imread(model_2_files[i]))
- analyse.rng = np.random.default_rng(seed=31052024)
- analyse.img_idx = list(range(N))
- list analyse.gt_samples = [np.squeeze(tifffile.imread(gt_files[i])) for i in img_idx]
- list analyse.raw_samples = [np.squeeze(tifffile.imread(raw_files[i])) for i in img_idx]
- list analyse.model_1_samples
- list analyse.model_2_samples
- analyse.cmap

7.1.1 Detailed Description

Script producing plots and small datasets that summarise the performance of models.

This script reads directories of reconstructed images, and compares raw versus model reconstructions versus ground truth. The script then produces summary statistics, saves relevant metrics to a .csv file, and produces samples of cropped image regions for comparison.

Arguments:

- g: directory path for ground-truth images
- r: directory path for raw images
- a: directory path for model-1-restored images
- b: directory path for model-2-restored images
- o: output directory for analysis plots, default "figures/"
- x: filepath for model 1 checkpoint (plots learning curve)
- y: filepath for model 2 checkpoint (plots learning curve)
- s: globbing string, to analyse a subset of images
- n: number of sample crops to display, default 0.
- p: plot only mode, skips data analysis

7.2 /home/jhughes2712/projects/sim_project/jh2284/src/apply.py File Reference

Script producing restored images resulting from an RCAN denoiser being applied to low SNR images.

Namespaces

- apply

Variables

- apply.parser = argparse.ArgumentParser()
- apply.type
- apply.str
- apply.required
- apply.int
- apply.choices
- apply.default
- apply.percentile
- apply.action
- apply.args = parser.parse_args()
- apply.input_path = pathlib.Path(args.input)
- apply.output_path = pathlib.Path(args.output)
- apply.parents
- apply.raw_files = sorted(input_path.glob("∗.tif"))
- apply.data = itertools.zip_longest(raw_files, [ ])
- tuple apply.device
- apply.ckpt
- apply.model
- apply.RCAN_hyperparameters = ckpt["hyperparameters"]
- list apply.overlap_shape
- apply.raw = normalize(tifffile.imread(raw_file), args.p_min, args.p_max)
- apply.restored
- apply.output_file = output_path / ("pred_" + raw_file.name)
- apply.imagej

7.2.1 Detailed Description

Script producing restored images resulting from an RCAN denoiser being applied to low SNR images.

This script takes directories of raw images, and a model checkpoint file, and applies the model to the image in a patched fashion. The details of this patching, and the output datatype, can be configured.

Arguments:

- m: model checkpoint filepath
- i: low SNR image directory path
- o: output directory path
- b: specifies pixel bit depth to save for output (8 or 16)
- O: block overlap shape (by default input_shape / 8)
- p_min: input normalization parameter, percentile maps to zero
- p_max: input normalization parameter, percentile maps to one
- normalize_output_range_between_zero_and_one: scaling for output

Adapted from `https://github.com/AiviaCommunity/3D-RCAN/blob/TF2/apply.py`

7.3 /home/jhughes2712/projects/sim_project/jh2284/src/convert_omx_↩ to_czxy.py File Reference

Script enabling .tif file conversion between OMX and CZXY.

Namespaces

- convert_omx_to_czxy

Variables

- convert_omx_to_czxy.parser = argparse.ArgumentParser()
- convert_omx_to_czxy.type
- convert_omx_to_czxy.str
- convert_omx_to_czxy.required
- convert_omx_to_czxy.int
- convert_omx_to_czxy.action
- convert_omx_to_czxy.args = parser.parse_args()
- convert_omx_to_czxy.input_dir = pathlib.Path(args.input)
- convert_omx_to_czxy.input_files = sorted(input_dir.rglob("∗.tif"))
- convert_omx_to_czxy.original = tifffile.imread(input_file)
- convert_omx_to_czxy.converted
- convert_omx_to_czxy.imagej

7.3.1 Detailed Description

Script enabling .tif file conversion between OMX and CZXY.

This script takes directories of image volumes as input, and converts, in place, between the OMX and CZXY formats (in either direction). In the OMX format, the first dimension is of size n_phases x n_z x n_angles; moving along this dimension, the phase changes first, then the z-value, then the angle. The CZXY format is the same, but the z-dimension of the image is separated into the 2nd dimension, so that the first dimension is just n_phases x n_angles.

Arguments:

- i: image directory
- p: number of phases
- a: number of angles
- b: specifies conversion - if not used it will be OMX to CZXY, the b flag reverses this direction.

7.4 /home/jhughes2712/projects/sim_project/jh2284/src/convert_omx_↩ to_paz.py File Reference

Script enabling .tif file conversion between OMX and PAZ.

Namespaces

- convert_omx_to_paz

Variables

- convert_omx_to_paz.parser = argparse.ArgumentParser()
- convert_omx_to_paz.type
- convert_omx_to_paz.str
- convert_omx_to_paz.required
- convert_omx_to_paz.int
- convert_omx_to_paz.action
- convert_omx_to_paz.args = parser.parse_args()
- convert_omx_to_paz.input_dir = pathlib.Path(args.input)
- convert_omx_to_paz.input_files = sorted(input_dir.rglob("∗.tif"))
- convert_omx_to_paz.original = tifffile.imread(input_file)
- convert_omx_to_paz.converted = conv_omx_to_paz(original, args.num_phases, args.num_angles)
- convert_omx_to_paz.imagej

7.4.1 Detailed Description

Script enabling .tif file conversion between OMX and PAZ.

This script takes directories of image volumes as input, and converts, in place, between the OMX and PAZ formats (in either direction). In the OMX format, the first dimension is of size n_phases x n_z x n_angles; moving along this dimension, the phase changes first, then the z-value, then the angle. The PAZ format is the same except the order is changed so that z-values and angels are swapped.

Arguments:

- i: image directory
- p: number of phases
- a: number of angles
- b: specifies conversion - if not used it will be OMX to PAZ, the b flag reverses this direction.

7.5 /home/jhughes2712/projects/sim_project/jh2284/src/convert_slices↩_to_volumes.py File Reference

Script enabling construction of 3D image volumes from large RGB 2D image slices.

Namespaces

- convert_slices_to_volumes

Variables

- convert_slices_to_volumes.parser = argparse.ArgumentParser()
- convert_slices_to_volumes.type
- convert_slices_to_volumes.str
- convert_slices_to_volumes.required
- convert_slices_to_volumes.tuple_of_ints
- convert_slices_to_volumes.default
- convert_slices_to_volumes.args = parser.parse_args()
- convert_slices_to_volumes.input_dir = pathlib.Path(args.input)
- convert_slices_to_volumes.output_dir = pathlib.Path(args.output)
- convert_slices_to_volumes.input_files = sorted(input_dir.glob("∗.tif"))
- convert_slices_to_volumes.parents
- convert_slices_to_volumes.True
- convert_slices_to_volumes.exist_ok
- convert_slices_to_volumes.volume = np.zeros((len(input_files), 3061, 4096), dtype=np.uint8)
- convert_slices_to_volumes.input_slice = tifffile.imread(file)
- convert_slices_to_volumes.output_file = output_dir / filename
- convert_slices_to_volumes.subvolume
- convert_slices_to_volumes.imagej

7.5.1 Detailed Description

Script enabling construction of 3D image volumes from large RGB 2D image slices.

Takes a directory of 2D image slices as input, and converts to 3D volumes. The 2D images are assumed to be ordered z-axially; the number of images is the number of voxels in the z-direction of the 3D volumes. The lateral cross-sections of the 3D images are determined by script arguments. Saves in uint16 depth.

Arguments:

- i: directory path for 2D images
- o: directory path for 3D image volumes
- s: start pixel coordinates (x, y)
- j: crop size for image volume (crop_x, crop_y)
- n: number of crops to take in each direction (steps_x, steps_y)
- l: filename prefix, default "volume"

7.6 /home/jhughes2712/projects/sim_project/jh2284/src/generate_sim.py File Reference

Script simulating the acquisition of 3D SIM image volumes.

Classes

- class generate_sim.Simulator

  The Simulator class encapsulates the state of a 3D microscope simulation.
- class generate_sim.SimulationRunner

  Class which performs a batch of simulations, either sequentially or in parallel.

Namespaces

- generate_sim

Functions

- def generate_sim.arange_zero (n, spacing=1)
- def generate_sim.threshold_norm (sample)

  Applies a threshold and normalises the sample to improve contrast.

Variables

- generate_sim.parser = argparse.ArgumentParser()
- generate_sim.type
- generate_sim.str
- generate_sim.required
- generate_sim.int
- generate_sim.default
- generate_sim.args = parser.parse_args()
- generate_sim.runner

7.6.1 Detailed Description

Script simulating the acquisition of 3D SIM image volumes.

Takes a directory of 3D image volumes as input, and produces synthetic 3-beam SIM volumes of size (15, 32, 256, 256).

Arguments:

- i: directory path of input volumes
- o: directory path of output volumes
- s: start index of sorted input files to process
- e: end index of sorted input files to process
- z: z_offset, used to specify the region of the input volume to use.

7.7 /home/jhughes2712/projects/sim_project/jh2284/src/image_↩ noising.py File Reference

Script which converts a directory of high-SNR SIM images into a training dataset.

Namespaces

- image_noising

Functions

- def image_noising.save_image_pair (gt_img, split, name, channel_idx)

Variables

- image_noising.parser = argparse.ArgumentParser()
- image_noising.type
- image_noising.str
- image_noising.required
- image_noising.int
- image_noising.choices
- image_noising.float
- image_noising.default
- image_noising.args = parser.parse_args()
- image_noising.input_path = pathlib.Path(args.input)
- image_noising.output_path = pathlib.Path(args.output)
- image_noising.parents
- image_noising.output_train_gt_path = output_path.joinpath("Training", "GT")
- image_noising.output_train_raw_path = output_path.joinpath("Training", "Raw")
- image_noising.output_val_gt_path = output_path.joinpath("Validation", "GT")
- image_noising.output_val_raw_path = output_path.joinpath("Validation", "Raw")
- image_noising.output_test_gt_path = output_path.joinpath("Testing", "GT")
- image_noising.output_test_raw_path = output_path.joinpath("Testing", "Raw")
- image_noising.data = sorted(input_path.glob("∗.tif"))
- image_noising.n_acquisitions = tifffile.imread(data[0]).shape[0] // args.channels
- image_noising.n_img = len(data)
- image_noising.train_size = int((1 - args.test_fraction) ∗ n_img)
- image_noising.val_size = int(args.val_fraction ∗ train_size)
- image_noising.rng = np.random.default_rng(seed=25042024)
- image_noising.img_idx_all = list(range(n_img))
- image_noising.img_idx_test = img_idx_all[train_size:]
- image_noising.img_idx_train = img_idx_all[: train_size - val_size]
- image_noising.img_idx_val = img_idx_all[train_size - val_size : train_size]
- image_noising.gt = tifffile.imread(img_file)
- string image_noising.split = "train"

7.7.1 Detailed Description

Script which converts a directory of high-SNR SIM images into a training dataset.

Each image is duplicated so that a low SNR counterpart is produced, simulating the same sample imaged with a lower illumination intensity. The data is then randomly split into train, validation, and testing subsets.

Arguments:

- i: directory path of input image
- o: directory path of output
- d: dimension
- s: scale factor used to simulate the low SNR images.
- tf: the fraction of the full dataset used for the hold-out test set.
- vf: the fraction of the *training* dataset that is reserved for validation during training.

7.8 /home/jhughes2712/projects/sim_project/jh2284/src/manage_↩ stack.py File Reference

Script handling the stacking and unstacking of groups of images, for the purpose of batch reconstructions.

Namespaces

- manage_stack

Variables

- manage_stack.parser = argparse.ArgumentParser()
- manage_stack.type
- manage_stack.str
- manage_stack.required
- manage_stack.int
- manage_stack.choices
- manage_stack.default
- manage_stack.action
- manage_stack.args = parser.parse_args()
- manage_stack.output_dir = pathlib.Path(args.output_dir)
- manage_stack.parents
- manage_stack.True
- manage_stack.exist_ok
- manage_stack.files = sorted(list(pathlib.Path(args.input_dir).glob(args.glob_str)))
- int manage_stack.stack_number = -1 else args.stack_number
- int manage_stack.number_of_stacks = len(files) // stack_number
- manage_stack.sample = tifffile.imread(files[0])
- manage_stack.stack_handler
- manage_stack.img_data = tifffile.imread(input_file)
- tuple manage_stack.filename
- tuple manage_stack.output_file = output_dir / filename
- manage_stack.output_data = img_data[j ∗ args.z_slices : (j + 1) ∗ args.z_slices]

7.8.1 Detailed Description

Script handling the stacking and unstacking of groups of images, for the purpose of batch reconstructions.

Takes a directory of images as input, and either stacks or unstacks the images there according to the configuration. 3D Image Volumes are expected to be in PAZ format. Note in unstack mode, images are saved with a first dimension of length 1 - this is the correct format for training the second step models (CZXY).

Arguments:

- i: directory path of input images
- o: directory path of output images
- n: output image name prefix - only applies in 'stack' mode
- d: dimension

- q: number of SIM acquisitions per image
- g: glob string used to choose images from input directory
- u: if used, sets mode to 'unstack'
- s: start index of sorted input files to process
- e: end index of sorted input files to process
- t: number of images to stack together - only applies in 'stack' mode. Default: -1 (all images are stacked)
- z: number of z slices of images - only applies in 'unstack' mode

7.9 /home/jhughes2712/projects/sim_project/jh2284/src/rcan/__init__.py File Reference

Namespaces

- rcan

7.10 /home/jhughes2712/projects/sim_project/jh2284/src/synthetic_↩ sim/__init__.py File Reference

Namespaces

- synthetic_sim

7.11 /home/jhughes2712/projects/sim_project/jh2284/src/rcan/data_↩ generator.py File Reference

Module that handles processing and batching of data during training loop.

Classes

- class rcan.data_generator.SIM_Dataset

  Generates batches of images with real-time data augmentation.

Namespaces

- rcan.data_generator

Functions

- def rcan.data_generator.load_SIM_dataset (images, shape, batch_size, transform_function, intensity_↩ threshold, area_threshold, scale_factor, steps_per_epoch, p_min, p_max)

  Wraps SIM_Dataset object in a PyTorch Dataloader object to enable batch loading.

7.11.1 Detailed Description

Module that handles processing and batching of data during training loop.

This module primarily defines the SIM_Datatset class which handles image cropping, normalization, augmentation, and intensity-threshold-area based rejection.

Migrated from `https://github.com/AiviaCommunity/3D-RCAN/blob/TF2/rcan/data_↩ generator.py`

7.12 /home/jhughes2712/projects/sim_project/jh2284/src/rcan/data_↩ processing.py File Reference

Contains tools used to pre-process image data.

Classes

- class rcan.data_processing.ImageStack

  Handles creation and loading of image hyperstacks in order to make reconstructions using ImageJ easier.

Namespaces

- rcan.data_processing

Functions

- def rcan.data_processing.crop_volume (volume, num_steps, start, step, label)

  Takes an image volume and divides part of it into smaller volumes by cropping lateral sections (the full z dimension is used).
- def rcan.data_processing.conv_omx_to_czxy (original, n_phases, n_angles)

  Converts image array from OMX (PZA format) to CZXY format.
- def rcan.data_processing.conv_czxy_to_omx (original, n_phases, n_angles)

  Converts image array from CZXY to OMX format.
- def rcan.data_processing.conv_omx_to_paz (original, n_phases, n_angles)

  Converts image array from OMX (PZA format) to PAZ format.
- def rcan.data_processing.conv_paz_to_omx (original, n_phases, n_angles)

  Converts image array from PAZ to OMX(PZA) format.

7.12.1 Detailed Description

Contains tools used to pre-process image data.

7.13 /home/jhughes2712/projects/sim_project/jh2284/src/rcan/model.py File Reference

Module defining the RCAN model architecture.

Classes

- class rcan.model._channel_attention_block

    Implements channel attention block/layer.
- class rcan.model._residual_channel_attention_blocks

    Implements residual group based on [1].
- class rcan.model.RCAN

    Builds a residual channel attention network.

Namespaces

- rcan.model

Functions

- def rcan.model._conv (ndim, in_filters, out_filters, kernel_size, padding="same", ∗∗kwargs)

    Returns the appropriate torch.nn convolution layer based on parameters.
- def rcan.model._global_average_pooling (ndim)

    Returns the appropriate torch.nn pooling layer based on parameters.
- def rcan.model._standardize (x)

    Standardises input data.
- def rcan.model._destandardize (x)

    Inverse of _standardize.

7.13.1 Detailed Description

Module defining the RCAN model architecture.

Module that defines a number of classes inheriting from nn.Module, implementing different levels of the RCAN architecture. This includes the channel attention layer, residual channel attention block, and RCAN itself.

Migrated from `https://github.com/AiviaCommunity/3D-RCAN/blob/TF2/rcan/model.py`

Copyright 2021 SVision Technologies LLC. Copyright 2021-2022 Leica Microsystems, Inc. Creative Commons Attribution-NonCommercial 4.0 International Public License (CC BY-NC 4.0) `https://creativecommons.↩ org/licenses/by-nc/4.0/`

7.14 /home/jhughes2712/projects/sim_↩ project/jh2284/src/rcan/plotting.py File Reference

Module providing helper functions for matplotlib plots.

Namespaces

- rcan.plotting

Functions

- def rcan.plotting.plot_learning_curve (losses_train, losses_val, psnr_train, psnr_val, ssim_train, ssim_val, figsize, output_path)

  Plots the learning curve metrics from a model checkpoint according to loss, PSNR, and SSIM.
- def rcan.plotting.plot_reconstructions (device, output_path, dim, gt_imgs, raw_imgs, model_1_imgs, model↩ _2_imgs=None, cmap="inferno")

  Plots a sample of reconstructions comparing GT vs Raw vs Restored.

7.14.1 Detailed Description

Module providing helper functions for matplotlib plots.

Provides tools to assist with analysis of trained networks, including samples of restored reconstructions, metrics, and model progress during training.

7.15 /home/jhughes2712/projects/sim_project/jh2284/src/rcan/utils.py File Reference

Contains utility functions for the training loop and inference.

Namespaces

- rcan.utils

Functions

- def rcan.utils.normalize (image, p_min=2, p_max=99.9, dtype="float32")

  Normalizes the image intensity so that the `p_min`-th and the `p_max`-th percentiles are converted to 0 and 1 respectively.
- def rcan.utils.apply (model, data, model_input_image_shape, model_output_image_shape, num_input_↩ channels, num_output_channels, batch_size, device, overlap_shape=None, verbose=False)

  Applies a model to an input image.
- def rcan.utils.load_rcan_checkpoint (ckpt_path, device)

  Enables loading of RCAN checkpointed model.
- def rcan.utils.tuple_of_ints (string)

  Defines behaviour of parsing tuples of ints (argparse).
- def rcan.utils.percentile (x)

  Defines behaviour of parsing percentiles (argparse).
- def rcan.utils.reshape_to_bcwh (data)

  Reshapes 2D or 3D array to have batch x channel x width x height format, by prepending extra dimensions.
- def rcan.utils.normalize_between_zero_and_one (data)

  Coerce pixel values to [0, 1] range.
- def rcan.utils.compute_metrics (img, gt_img, psnr, ssim)

  Uses ignite metric objects to compute PSNR and SSIM.

7.15.1 Detailed Description

Contains utility functions for the training loop and inference.

Migrated from https://github.com/AiviaCommunity/3D-RCAN/blob/TF2/rcan/utils.py

7.16 /home/jhughes2712/projects/sim_project/jh2284/src/recon_↩ postprocess.py File Reference

Script handling the postprocessing of SIM reconstructions.

Namespaces

- recon_postprocess

Variables

- recon_postprocess.parser = argparse.ArgumentParser()
- recon_postprocess.type
- recon_postprocess.str
- recon_postprocess.required
- recon_postprocess.args = parser.parse_args()
- recon_postprocess.files = sorted(list(pathlib.Path(args.input_dir).rglob("∗.tif")))
- recon_postprocess.img_data = tifffile.imread(input_file)

7.16.1 Detailed Description

Script handling the postprocessing of SIM reconstructions.

Takes a directory of images as input, clips zero values, and scales to the full 16-bit depth range. Operates in-place.

Arguments:

- i: directory path of input images

7.17 /home/jhughes2712/projects/sim_project/jh2284/src/recon_↩ preprocess.py File Reference

Script handling the preprocessing of images before SIM reconstruction.

Namespaces

- recon_preprocess

Functions

- def recon_preprocess.normalize_acquisition_intensity (data, dim)

Variables

- recon_preprocess.parser = argparse.ArgumentParser()
- recon_preprocess.type
- recon_preprocess.str
- recon_preprocess.required
- recon_preprocess.int
- recon_preprocess.choices
- recon_preprocess.percentile
- recon_preprocess.default
- recon_preprocess.action
- recon_preprocess.args = parser.parse_args()
- recon_preprocess.output_dir = pathlib.Path(args.output_dir)
- recon_preprocess.parents
- recon_preprocess.True
- recon_preprocess.exist_ok
- recon_preprocess.files = sorted(list(pathlib.Path(args.input_dir).glob("∗.tif")))
- recon_preprocess.img_data = tifffile.imread(input_file).astype("float32")
- recon_preprocess.output_file = output_dir / input_file.name

7.17.1 Detailed Description

Script handling the preprocessing of images before SIM reconstruction.

Takes a directory of images as input, equalizes the total acquisition, intensities within each image, subtracts background and extreme pixels on a percentile basis, then scales to the full 16-bit depth range.

Arguments:

- i: directory path of input images
- o: directory path of output images
- d: dimension
- l: lower percentile used for clipping (background)
- u: upper percentile used for clipping (bright values)
- n: turns on normalization of acquisition intensity

7.18 /home/jhughes2712/projects/sim_project/jh2284/src/stats.py File Reference

Namespaces

- stats

Functions

- def stats.paired_t (gt_data, data)

Variables

- stats.parser = argparse.ArgumentParser()
- stats.type
- stats.str
- stats.required
- stats.int
- stats.choices
- stats.default
- stats.args = parser.parse_args()
- stats.output_dir = pathlib.Path(args.output_dir)
- stats.parents
- stats.True
- stats.exist_ok
- stats.df
- stats.fig
- stats.ax
- stats.figsize
- stats.psnr_diff_1_max
- stats.psnr_diff_2_max
- stats.psnr_diff_1_min
- stats.psnr_diff_2_min
- tuple stats.hist_range_psnr
- stats.ssim_diff_1_max
- stats.ssim_diff_2_max
- stats.ssim_diff_1_min
- stats.ssim_diff_2_min
- tuple stats.hist_range_ssim
- stats.xlabel
- stats.title
- stats.range
- stats.color
- stats.mean_psnr_1 = np.mean(np.array(df['psnr_model_1']) - np.array(df['psnr_raw']))
- stats.se_psnr_1
- stats.mean_ssim_1 = np.mean(np.array(df['ssim_model_1']) - np.array(df['ssim_raw']))
- stats.se_ssim_1
- stats.mean_psnr_2
- stats.se_psnr_2
- stats.mean_ssim_2
- stats.se_ssim_2

- int stats.psnr_cols = 2 else df.columns[1:3]
- int stats.ssim_cols = 2 else df.columns[3:5]
- stats.dflong
- stats.dflongssim
- stats.data
- stats.x
- stats.y
- stats.hue
- stats.dodge
- stats.legend
- stats.palette
- stats.alpha
- stats.lw

7.19 /home/jhughes2712/projects/sim_project/jh2284/src/synthetic_↩ sim/otf.py File Reference

Classes

- class synthetic_sim.otf.PsfParameters

  Class to store PSF parameters.

Namespaces

- synthetic_sim.otf

Functions

- def synthetic_sim.otf.calc_psf (params)

  Calculate an approximate Gibson-Lanni PSF based on the parameters provided.

7.20 /home/jhughes2712/projects/sim_project/jh2284/src/train.py File Reference

Script used to train RCAN.

Namespaces

- train

Functions

- def train.load_data_paths (config, data_type)
- def train.train (train_loader, val_loader, optimizer, scheduler, net, batchsize, n_accumulations, saveinterval, nepoch, start_epoch=0, losses_train_epoch=[ ], losses_val_epoch=[ ], psnr_train_epoch=[ ], psnr_val_↩ epoch=[ ], ssim_train_epoch=[ ], ssim_val_epoch=[ ])

Variables

- train.parser = argparse.ArgumentParser()
- train.type
- train.str
- train.required
- train.args = parser.parse_args()
- dictionary train.schema
- train.config = json.load(f)
- int train.ndim = tifffile.imread(training_data[0]["raw"]).ndim - 1
- train.input_shape = config["input_shape"]
- tuple train.device
- train.ckpt_path = None if args.model_ckpt is None else pathlib.Path(args.model_ckpt)
- train.model
- dictionary train.RCAN_hyperparameters
- train.ckpt
- train.train_loader
- train.val_loader
- train.optimizer
- train.scheduler
- train.output_dir = pathlib.Path(args.output_dir)
- train.parents
- train.True
- train.exist_ok
- train.n_accumulations
- train.saveinterval
- train.nepoch
- train.start_epoch
- train.losses_train_epoch
- train.losses_val_epoch
- train.psnr_train_epoch
- train.psnr_val_epoch
- train.ssim_train_epoch
- train.ssim_val_epoch

7.20.1 Detailed Description

Script used to train RCAN.

Reads the specified config.json file, and trains an RCAN model accordingly. Intermediate training progress is saved using model checkpoints. Can handle resumed model training if a previous checkpoint is provided.

Arguments:

- c: filepath for config JSON file
- o: path of model checkpoint directory
- m: filepath of intermediate model checkpoint (if given, training resumes from this checkpoint)

Adapted from `https://github.com/AiviaCommunity/3D-RCAN/blob/TF2/train.py`

Index