

MNIST Diffusion Project

Generated by Doxygen 1.9.1

1 MNIST Diffusion Project	1
1.1 Modules	1
1.1.1 models	1
1.1.2 train	1
2 Namespace Index	3
2.1 Packages	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Namespace Documentation	11
6.1 diffusiontools Namespace Reference	11
6.2 diffusiontools.analysis Namespace Reference	11
6.2.1 Function Documentation	12
6.2.1.1 compute_image_diffusion()	12
6.2.1.2 compute_image_diffusion_custom()	12
6.2.1.3 compute_tsne_kl_div()	13
6.2.1.4 compute_variance()	13
6.2.1.5 degradation_demo()	14
6.2.1.6 plot_gt_direct_diffusion()	14
6.2.1.7 plot_image_diffusion()	15
6.2.1.8 plot_learning_curve()	15
6.2.1.9 plot_mnist_tsne()	15
6.2.1.10 plot_sample_tsne()	16
6.2.1.11 plot_samples()	16
6.3 diffusiontools.models Namespace Reference	17
6.3.1 Function Documentation	17
6.3.1.1 ddpm_schedules()	17
6.4 diffusiontools.train Namespace Reference	18
6.4.1 Function Documentation	18
6.4.1.1 train_model()	18
6.5 final_analysis Namespace Reference	19
6.5.1 Variable Documentation	19
6.5.1.1 _	19
6.5.1.2 ckpt_0001_0120	20
6.5.1.3 ckpt_0001_0490	20
6.5.1.4 ckpt_0002_0080	20

6.5.1.5 ckpt_0002_0490	20
6.5.1.6 ckpt_0009_0020	20
6.5.1.7 ckpt_0009_0080	21
6.5.1.8 ckpt_0010_0140	21
6.5.1.9 ckpt_0010_0180	21
6.5.1.10 density_gt	21
6.5.1.11 device	21
6.5.1.12 direct	21
6.5.1.13 gt_1	22
6.5.1.14 gt_10	22
6.5.1.15 gt_2	22
6.5.1.16 gt_9	22
6.5.1.17 model_1	22
6.5.1.18 model_10	23
6.5.1.19 model_2	23
6.5.1.20 model_9	23
6.5.1.21 sample_1	23
6.5.1.22 sample_10	23
6.5.1.23 sample_10_direct	23
6.5.1.24 sample_2	24
6.5.1.25 sample_9	24
6.5.1.26 sample_9_direct	24
6.5.1.27 SAMPLE_SIZE	24
6.5.1.28 samples_densities	24
6.5.1.29 samples_fitted	24
6.5.1.30 total_var_1	24
6.5.1.31 total_var_10	24
6.5.1.32 total_var_2	25
6.5.1.33 total_var_9	25
6.6 intermediate_analysis Namespace Reference	25
6.6.1 Variable Documentation	25
6.6.1.1 ckpt_0001	25
6.6.1.2 ckpt_0002	26
6.6.1.3 ckpt_0009	26
6.6.1.4 ckpt_0010	26
6.6.1.5 device	26
6.6.1.6 gt_1	26
6.6.1.7 gt_10	27
6.6.1.8 gt_2	27
6.6.1.9 gt_9	27
6.6.1.10 model_1	27
6.6.1.11 model_10	27

6.6.1.12 model_2	28
6.6.1.13 model_9	28
6.7 run Namespace Reference	28
6.7.1 Variable Documentation	28
6.7.1.1 accelerator	29
6.7.1.2 batch_size	29
6.7.1.3 checkpoint_path	29
6.7.1.4 config	29
6.7.1.5 config_id	29
6.7.1.6 criterion	29
6.7.1.7 dataloader_train	29
6.7.1.8 dataloader_val	30
6.7.1.9 dataset	30
6.7.1.10 dataset_train	30
6.7.1.11 dataset_val	30
6.7.1.12 degradation	30
6.7.1.13 gt	30
6.7.1.14 input_file	31
6.7.1.15 loss_fn	31
6.7.1.16 lr_initial	31
6.7.1.17 model	31
6.7.1.18 n_epoch	31
6.7.1.19 n_hidden	31
6.7.1.20 n_T	31
6.7.1.21 noise_max	32
6.7.1.22 noise_min	32
6.7.1.23 optim	32
6.7.1.24 sample_path	32
6.7.1.25 save_interval	32
6.7.1.26 tf	32
7 Class Documentation	33
7.1 diffusiontools.models.CNN Class Reference	33
7.1.1 Detailed Description	34
7.1.2 Constructor & Destructor Documentation	34
7.1.2.1 __init__()	34
7.1.3 Member Function Documentation	35
7.1.3.1 forward()	35
7.1.3.2 time_encoding()	35
7.1.4 Member Data Documentation	35
7.1.4.1 blocks	35
7.1.4.2 time_embed	36

7.2 diffusiontools.models.CNNBlock Class Reference	36
7.2.1 Detailed Description	37
7.2.2 Constructor & Destructor Documentation	37
7.2.2.1 __init__()	37
7.2.3 Member Function Documentation	37
7.2.3.1 forward()	37
7.2.4 Member Data Documentation	38
7.2.4.1 net	38
7.3 diffusiontools.models.DDPM Class Reference	38
7.3.1 Detailed Description	39
7.3.2 Constructor & Destructor Documentation	39
7.3.2.1 __init__()	39
7.3.3 Member Function Documentation	39
7.3.3.1 forward()	40
7.3.3.2 sample()	40
7.3.4 Member Data Documentation	40
7.3.4.1 criterion	40
7.3.4.2 gt	41
7.3.4.3 n_T	41
7.4 diffusiontools.models.DMCustom Class Reference	41
7.4.1 Detailed Description	42
7.4.2 Constructor & Destructor Documentation	42
7.4.2.1 __init__()	42
7.4.3 Member Function Documentation	43
7.4.3.1 degrade()	43
7.4.3.2 forward()	43
7.4.3.3 sample()	44
7.4.4 Member Data Documentation	44
7.4.4.1 criterion	44
7.4.4.2 gt	44
7.4.4.3 n_T	44
7.4.4.4 size	44
8 File Documentation	45
8.1 /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/__init__.py File Reference	45
8.2 /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/analysis.py File Reference	45
8.2.1 Detailed Description	46
8.3 /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py File Reference	46
8.3.1 Detailed Description	47
8.4 /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/train.py File Reference	47
8.4.1 Detailed Description	47
8.5 /home/jhughes2712/projects/m2_assessment/jh2284/src/final_analysis.py File Reference	47

8.6 /home/jhughes2712/projects/m2_assessment/jh2284/src/intermediate_analysis.py File Reference . .	48
8.7 /home/jhughes2712/projects/m2_assessment/jh2284/src/run.py File Reference	49
8.7.1 Detailed Description	50
Index	51

Chapter 1

MNIST Diffusion Project

Software used to investigate the use of generative diffusion modelling to recreate realistic monochrome images of handwritten digits, using the classical MNIST dataset as training data. The software utilises PyTorch to implement the neural network models and the training procedure.

1.1 Modules

1.1.1 models

This module contains code used to instantiate standard image-to-image convolutional neural networks, as well as a DDPM class, with methods that enable training the image-to-image model according to the diffusion paradigm, and similarly performing diffusion sampling.

1.1.2 train

This module contains the `ddpm_train` function, which implements the diffusion training process, and saves the model parameters and samples periodically, according to specified training parameters.

Author

Created by J. Hughes on 06/12/2023.

Chapter 2

Namespace Index

2.1 Packages

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

diffusiontools	11
diffusiontools.analysis	11
diffusiontools.models	17
diffusiontools.train	18
final_analysis	19
intermediate_analysis	25
run	28

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

nn.Module	
diffusiontools.models.CNN	33
diffusiontools.models.CNNBlock	36
diffusiontools.models.DDPM	38
diffusiontools.models.DMCustom	41

Chapter 4

Class Index

4.1 Class List

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

diffusiontools.models.CNN	Defines CNN class used to reconstruct images during diffusion	33
diffusiontools.models.CNNBlock	Class instantiating a convolutional block with layer normalisation and non-linear activation . . .	36
diffusiontools.models.DDPM	Defines Gaussian diffusion model class	38
diffusiontools.models.DMCustom	Class defining custom diffusion model	41

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/jhughes2712/projects/m2_assessment/jh2284/src/ final_analysis.py	47
/home/jhughes2712/projects/m2_assessment/jh2284/src/ intermediate_analysis.py	48
/home/jhughes2712/projects/m2_assessment/jh2284/src/ run.py Script used to create and train diffusion model, based on parameters provided via the config.ini file	49
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusio n tools/ __init__.py	45
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusio n tools/ analysis.py Module containing procedures used to analyse and compare trained diffusion models	45
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusio n tools/ models.py Module containing classes and procedures used to build diffusion models	46
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusio n tools/ train.py Module containing procedures used to train and save diffusion models	47

Chapter 6

Namespace Documentation

6.1 diffusiontools Namespace Reference

Namespaces

- [analysis](#)
- [models](#)
- [train](#)

6.2 diffusiontools.analysis Namespace Reference

Functions

- def [plot_learning_curve](#) (List[float] losses_train, List[float] losses_val, str title, str filename)
Plot changes in loss function across epochs for model training.
- def [compute_image_diffusion](#) (DDPM model, List[float] image_fractions, int n_sample, size, device)
Save intermediate image generations at specified parts of the Gaussian diffusion process.
- torch.Tensor [compute_image_diffusion_custom](#) (DMCustom model, List[int] image_fractions, int n_sample, size, device, bool direct=False)
Save intermediate image generations at specified parts of the custom diffusion process.
- def [plot_image_diffusion](#) (model, List[int] image_fractions, int n_sample, size, device, str title, str filename)
Plot intermediate image generations at specified parts of the custom diffusion process.
- def [plot_samples](#) (np.ndarray samples, int n_row, int n_col, str filename)
Produce matrix of synthetic image samples.
- def [plot_gt_direct_diffusion](#) (np.ndarray direct_sample, np.ndarray diffusion_sample, int n_plot, str filename)
Compare direct and diffused reconstructions.
- def [compute_variance](#) (np.ndarray sample)
Compute the total variance of a sample of vectors.
- def [degradation_demo](#) (DMCustom model, device, List[int] t_list, str title, str filename)
Plot MNIST images of digits 1 to 4 with varying custom degradation levels.
- def [compute_tsne_kl_div](#) (List[np.ndarray] samples, List[str] labels)
Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.
- def [plot_sample_tsne](#) (np.ndarray density_gt, np.ndarray sample_1_fitted, np.ndarray sample_2_fitted, np.ndarray density_1, np.ndarray density_2, str label_1, str label_2, str filename)
Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.
- def [plot_mnist_tsne](#) (str filename)
Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.

6.2.1 Function Documentation

6.2.1.1 compute_image_diffusion()

```
def diffusiontools.analysis.compute_image_diffusion (
    DDPM model,
    List[float] image_fractions,
    int n_sample,
    size,
    device )
```

Save intermediate image generations at specified parts of the Gaussian diffusion process.

Similar to the sample() method of the DDPM class, except intermediate reconstructions are saved.

Parameters

<i>model</i>	DDPM Class instance, which implements the Gaussian noising schedule
<i>image_fractions</i>	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
<i>n_sample</i>	Integer specifying number of samples to generate
<i>size</i>	Image size as (channels, height, width)
<i>device</i>	PyTorch device variable used to control processing unit

Returns

image_idx List of discrete integer time steps at which reconstructions were saved

images List of intermediate image reconstructions.

6.2.1.2 compute_image_diffusion_custom()

```
torch.Tensor diffusiontools.analysis.compute_image_diffusion_custom (
    DMCustom model,
    List[int] image_fractions,
    int n_sample,
    size,
    device,
    bool direct = False )
```

Save intermediate image generations at specified parts of the custom diffusion process.

Generates an initial image by maximally degrading an MNIST test sample according to the model's parameters, and then performing the reverse diffusion process.

Parameters

<i>model</i>	DMCustom Class instance, which implements the custom noising schedule
--------------	---

Parameters

<i>image_fractions</i>	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
<i>n_sample</i>	Integer specifying number of samples to generate
<i>size</i>	Image size as (channels, height, width)
<i>device</i>	PyTorch device variable used to control processing unit
<i>direct</i>	Determines whether the very first reconstruction is saved (without a degradation)

Returns

image_idx List of discrete integer time steps at which reconstructions were saved
images List of intermediate image reconstructions.
direct_img Stack of direct image reconstructions if this was specified

6.2.1.3 compute_tsne_kl_div()

```
def diffusiontools.analysis.compute_tsne_kl_div (
    List[np.ndarray] samples,
    List[str] labels )
```

Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.

Parameters

<i>samples</i>	List of different generated image stacks
<i>labels</i>	Labels for the sample to use in the output

Returns

density_gt Fitted GMM density values for the GT data
samples_fitted t-SNE embeddings for each of the samples
samples_densities Fitted GMM density maps for each sample

6.2.1.4 compute_variance()

```
def diffusiontools.analysis.compute_variance (
    np.ndarray sample )
```

Compute the total variance of a sample of vectors.

Parameters

<i>sample</i>	NumPy array whose first dimension is the image index
---------------	--

Returns

`total_variance` The computed total variance of the vectors

6.2.1.5 degradation_demo()

```
def diffusiontools.analysis.degradation_demo (
    DMCustom model,
    device,
    List[int] t_list,
    str title,
    str filename )
```

Plot MNIST images of digits 1 to 4 with varying custom degradation levels.

Parameters

<i>model</i>	DMCustom class instance
<i>device</i>	PyTorch device variable used to control processing unit
<i>t_list</i>	List of time steps at which to demonstrate the degradation
<i>title</i>	String for title to be included in the figure
<i>filename</i>	String determining saved filename for figure

6.2.1.6 plot_gt_direct_diffusion()

```
def diffusiontools.analysis.plot_gt_direct_diffusion (
    np.ndarray direct_sample,
    np.ndarray diffusion_sample,
    int n_plot,
    str filename )
```

Compare direct and diffused reconstructions.

Produce a 3x[n_plot] grid of images comparing corresponding ground truth vs. direct vs. diffusion image samples, and then compute MSE statistics for the full samples.

Parameters

<i>direct_sample</i>	NumPy array whose first dimension is the image index
<i>diffusion_sample</i>	NumPy array whose first dimension is the image index
<i>filename</i>	String determining saved filename for figure

6.2.1.7 plot_image_diffusion()

```
def diffusiontools.analysis.plot_image_diffusion (
    model,
    List[int] image_fractions,
    int n_sample,
    size,
    device,
    str title,
    str filename )
```

Plot intermediate image generations at specified parts of the custom diffusion process.

Parameters

<i>model</i>	DDPM or DMCustom class instance
<i>image_fractions</i>	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
<i>n_sample</i>	Integer specifying number of samples to generate
<i>size</i>	Image size as (channels, height, width)
<i>device</i>	PyTorch device variable used to control processing unit
<i>title</i>	String for title to be included in the figure
<i>filename</i>	String determining saved filename for figure

6.2.1.8 plot_learning_curve()

```
def diffusiontools.analysis.plot_learning_curve (
    List[float] losses_train,
    List[float] losses_val,
    str title,
    str filename )
```

Plot changes in loss function across epochs for model training.

Parameters

<i>losses_train</i>	List of training losses per epoch
<i>losses_val</i>	List of validation losses per epoch
<i>title</i>	String of title to be displayed in plot
<i>filename</i>	String of filename to save to

6.2.1.9 plot_mnist_tsne()

```
def diffusiontools.analysis.plot_mnist_tsne (
```

```
str filename )
```

Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.

Parameters

<i>filename</i>	String determining saved filename for figure
-----------------	--

6.2.1.10 plot_sample_tsne()

```
def diffusiontools.analysis.plot_sample_tsne (
    np.ndarray density_gt,
    np.ndarray sample_1_fitted,
    np.ndarray sample_2_fitted,
    np.ndarray density_1,
    np.ndarray density_2,
    str label_1,
    str label_2,
    str filename )
```

Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.

Parameters

<i>density_gt</i>	Mesh of fitted GMM density for the 2D ground truth embedding
<i>sample_1_fitted</i>	2D embeddings for sample 1
<i>sample_2_fitted</i>	2D embeddings for sample 2
<i>density_1</i>	Mesh of fitted GMM density for sample 1
<i>density_2</i>	Mesh of fitted GMM density for sample 2
<i>label_1</i>	Legend label for sample 1
<i>label_2</i>	Legend label for sample 2
<i>filename</i>	String determining saved filename for figure

6.2.1.11 plot_samples()

```
def diffusiontools.analysis.plot_samples (
    np.ndarray samples,
    int n_row,
    int n_col,
    str filename )
```

Produce matrix of synthetic image samples.

Parameters

<i>samples</i>	NumPy array whose first dimension is the image index
----------------	--

Parameters

<i>image_fractions</i>	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
<i>n_row</i>	Integer specifying number of rows
<i>n_col</i>	Integer specifying number of columns
<i>filename</i>	String determining saved filename for figure

6.3 diffusiontools.models Namespace Reference

Classes

- class [CNNBlock](#)
Class instantiating a convolutional block with layer normalisation and non-linear activation.
- class [CNN](#)
Defines [CNN](#) class used to reconstruct images during diffusion.
- class [DDPM](#)
Defines Gaussian diffusion model class.
- class [DMCustom](#)
Class defining custom diffusion model.

Functions

- Dict[str, torch.Tensor] [ddpm_schedules](#) (float beta1, float beta2, int T)
Returns pre-computed schedules for [DDPM](#) sampling with a linear noise schedule.

6.3.1 Function Documentation

6.3.1.1 ddpm_schedules()

```
Dict[str, torch.Tensor] diffusiontools.models.ddpm_schedules (
    float beta1,
    float beta2,
    int T )
```

Returns pre-computed schedules for [DDPM](#) sampling with a linear noise schedule.

Parameters

<i>beta1</i>	Minimal noise parameter
<i>beta2</i>	Taximal noise parameter
<i>T</i>	Total number of discrete diffusion time steps

Returns

Dict[str,torch.Tensor] Dictionary storing alpha and beta noise parameter vectors

6.4 diffusiontools.train Namespace Reference

Functions

- def [train_model](#) (nn.Module ddpm, torch.optim.Optimizer optim, DataLoader dataloader_train, DataLoader dataloader_val, Accelerator accelerator, int n_epoch, int save_interval, str sample_path, str checkpoint_path, str config_id)

Handles the training process for a diffusion model.

6.4.1 Function Documentation

6.4.1.1 train_model()

```
def diffusiontools.train.train_model (
    nn.Module ddpm,
    torch.optim.Optimizer optim,
    DataLoader dataloader_train,
    DataLoader dataloader_val,
    Accelerator accelerator,
    int n_epoch,
    int save_interval,
    str sample_path,
    str checkpoint_path,
    str config_id )
```

Handles the training process for a diffusion model.

Parameters

<i>ddpm</i>	Diffusion model object; either DDPM or DMCustom class instance
<i>dataloader_train</i>	Iterable that yields pairs of matched ground truth and degraded image samples for training
<i>dataloader_val</i>	Iterable that yields pairs of matched ground truth and degraded image samples for model validation
<i>accelerator</i>	accelerate package object which simplifies the management of CPU/GPU processing devices
<i>n_epoch</i>	Total number of epochs to train for before stopping
<i>save_interval</i>	Number of epochs to wait for between saving models
<i>sample_path</i>	Specifies where to save model samples (deprecated)
<i>checkpoint_path</i>	Specifies where to save model checkpoint files
<i>config_id</i>	Specifies configuration id to use to label model checkpoint files

6.5 final_analysis Namespace Reference

Variables

- `device` = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- `gt_1`
- `model_1` = DDPM(gt=`gt_1`, betas=(0.0001, 0.02), n_T=1000).to(`device`)
- `ckpt_0001_0490`
- `gt_2`
- `model_2` = DDPM(gt=`gt_2`, betas=(0.001, 0.2), n_T=100).to(`device`)
- `ckpt_0002_0490`
- `gt_9`
- `model_9`
- `ckpt_0009_0080`
- `gt_10`
- `model_10`
- `ckpt_0010_0180`
- `ckpt_0001_0120`
- `ckpt_0002_0080`
- `ckpt_0009_0020`
- `ckpt_0010_0140`
- `int SAMPLE_SIZE` = 200
- `_`
- `sample_1`
- `sample_2`
- `sample_9`
- `sample_9_direct`
- `direct`
- `sample_10`
- `sample_10_direct`
- `total_var_1` = compute_variance(`sample_1`[0])
- `total_var_2` = compute_variance(`sample_2`[0])
- `total_var_9` = compute_variance(`sample_9`[0])
- `total_var_10` = compute_variance(`sample_10`[0])
- `density_gt`
- `samples_fitted`
- `samples_densities`

6.5.1 Variable Documentation

6.5.1.1 `_`

`final_analysis._` [private]

6.5.1.2 ckpt_0001_0120

`final_analysis.ckpt_0001_0120`

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0001_0120.pt", map_location=device  
3 )
```

6.5.1.3 ckpt_0001_0490

`final_analysis.ckpt_0001_0490`

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0001_0490.pt", map_location=device  
3 )
```

6.5.1.4 ckpt_0002_0080

`final_analysis.ckpt_0002_0080`

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0002_0080.pt", map_location=device  
3 )
```

6.5.1.5 ckpt_0002_0490

`final_analysis.ckpt_0002_0490`

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0002_0490.pt", map_location=device  
3 )
```

6.5.1.6 ckpt_0009_0020

`final_analysis.ckpt_0009_0020`

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0009_0020.pt", map_location=device  
3 )
```

6.5.1.7 ckpt_0009_0080

final_analysis.ckpt_0009_0080

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0009_0080.pt", map_location=device  
3 )
```

6.5.1.8 ckpt_0010_0140

final_analysis.ckpt_0010_0140

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0010_0140.pt", map_location=device  
3 )
```

6.5.1.9 ckpt_0010_0180

final_analysis.ckpt_0010_0180

Initial value:

```
1 = torch.load(  
2     "data/DDPM/checkpoint/ddpm_checkpoint_0010_0180.pt", map_location=device  
3 )
```

6.5.1.10 density_gt

final_analysis.density_gt

6.5.1.11 device

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

6.5.1.12 direct

final_analysis.direct

6.5.1.13 gt_1

`final_analysis.gt_1`

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.5.1.14 gt_10

`final_analysis.gt_10`

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.5.1.15 gt_2

`final_analysis.gt_2`

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.5.1.16 gt_9

`final_analysis.gt_9`

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.5.1.17 model_1

`final_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)`

6.5.1.18 model_10

```
final_analysis.model_10
```

Initial value:

```
1 = DMCustom(  
2     gt=gt_10, alphas=(0.035, 0.2), n_T=20, size=(1, 28, 28)  
3 ).to(device)
```

6.5.1.19 model_2

```
final_analysis.model_2 = DDPM(gt=gt_2, betas=(0.001, 0.2), n_T=100).to(device)
```

6.5.1.20 model_9

```
final_analysis.model_9
```

Initial value:

```
1 = DMCustom(  
2     gt=gt_9,  
3     alphas=(0.035, 0.15),  
4     n_T=100,  
5     size=(1, 28, 28),  
6     criterion=nn.L1Loss(),  
7 ).to(device)
```

6.5.1.21 sample_1

```
final_analysis.sample_1
```

6.5.1.22 sample_10

```
final_analysis.sample_10
```

6.5.1.23 sample_10_direct

```
final_analysis.sample_10_direct
```

6.5.1.24 sample_2

```
final_analysis.sample_2
```

6.5.1.25 sample_9

```
final_analysis.sample_9
```

6.5.1.26 sample_9_direct

```
final_analysis.sample_9_direct
```

6.5.1.27 SAMPLE_SIZE

```
int final_analysis.SAMPLE_SIZE = 200
```

6.5.1.28 samples_densities

```
final_analysis.samples_densities
```

6.5.1.29 samples_fitted

```
final_analysis.samples_fitted
```

6.5.1.30 total_var_1

```
final_analysis.total_var_1 = compute_variance(sample_1[0])
```

6.5.1.31 total_var_10

```
final_analysis.total_var_10 = compute_variance(sample_10[0])
```


6.5.1.32 total_var_2

```
final_analysis.total_var_2 = compute_variance(sample_2[0])
```

6.5.1.33 total_var_9

```
final_analysis.total_var_9 = compute_variance(sample_9[0])
```

6.6 intermediate_analysis Namespace Reference

Variables

- `device` = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- `ckpt_0001`
- `gt_1`
- `model_1` = DDPM(gt=`gt_1`, betas=(0.0001, 0.02), n_T=1000).to(`device`)
- `ckpt_0002`
- `gt_2`
- `model_2` = DDPM(gt=`gt_2`, betas=(0.0001, 0.02), n_T=1000).to(`device`)
- `ckpt_0009`
- `gt_9`
- `model_9`
- `ckpt_0010`
- `gt_10`
- `model_10`

6.6.1 Variable Documentation

6.6.1.1 ckpt_0001

```
intermediate_analysis.ckpt_0001
```

Initial value:

```
1 = torch.load(  
2     f"data/DDPM/checkpoint/ddpm_checkpoint_0001_{epoch:04d}.pt",  
3     map_location=device,  
4 )
```

6.6.1.2 ckpt_0002

intermediate_analysis.ckpt_0002

Initial value:

```
1 = torch.load(
2     f"data/DDPM/checkpoint/ddpm_checkpoint_0002_{epoch:04d}.pt",
3     map_location=device,
4 )
```

6.6.1.3 ckpt_0009

intermediate_analysis.ckpt_0009

Initial value:

```
1 = torch.load(
2     f"data/DDPM/checkpoint/ddpm_checkpoint_0009_{epoch:04d}.pt",
3     map_location=device,
4 )
```

6.6.1.4 ckpt_0010

intermediate_analysis.ckpt_0010

Initial value:

```
1 = torch.load(
2     f"data/DDPM/checkpoint/ddpm_checkpoint_0010_{epoch:04d}.pt",
3     map_location=device,
4 )
```

6.6.1.5 device

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

6.6.1.6 gt_1

intermediate_analysis.gt_1

Initial value:

```
1 = CNN(
2     in_channels=1,
3     expected_shape=(28, 28),
4     n_hidden=(16, 32, 32, 16),
5     act=nn.GELU,
6 ).to(device)
```

6.6.1.7 gt_10

intermediate_analysis.gt_10

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.6.1.8 gt_2

intermediate_analysis.gt_2

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.6.1.9 gt_9

intermediate_analysis.gt_9

Initial value:

```
1 = CNN(  
2     in_channels=1,  
3     expected_shape=(28, 28),  
4     n_hidden=(16, 32, 32, 16),  
5     act=nn.GELU,  
6 ).to(device)
```

6.6.1.10 model_1

intermediate_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)

6.6.1.11 model_10

intermediate_analysis.model_10

Initial value:

```
1 = DMCustom(  
2     gt=gt_10, alphas=(0.035, 0.2), n_T=20, size=(1, 28, 28)  
3 ).to(device)
```

6.6.1.12 model_2

```
intermediate_analysis.model_2 = DDPM(gt=gt_2, betas=(0.0001, 0.02), n_T=1000).to(device)
```

6.6.1.13 model_9

```
intermediate_analysis.model_9
```

Initial value:

```
1 = DMCustom(
2     gt=gt_9,
3     alphas=(0.035, 0.15),
4     n_T=100,
5     size=(1, 28, 28),
6     criterion=nn.L1Loss(),
7 ).to(device)
```

6.7 run Namespace Reference

Variables

- `config` = `cfg.ConfigParser()`
- `input_file` = `sys.argv[1]`
- `noise_min` = `config.getfloat("model", "noise_min", fallback=1e-4)`
- `noise_max` = `config.getfloat("model", "noise_max", fallback=0.02)`
- `n_T` = `config.getint("model", "n_T", fallback=1000)`
- `degradation` = `config.get("model", "degradation", fallback="gaussian")`
- `n_hidden` = `config.get("model", "n_hidden", fallback="16 32 32 16")`
- `loss_fn` = `config.get("model", "loss_fn", fallback="L2")`
- `n_epoch` = `config.getint("training", "n_epoch", fallback=100)`
- `batch_size` = `config.getint("training", "batch_size", fallback=128)`
- `lr_initial` = `config.getfloat("training", "lr_initial", fallback=2e-4)`
- `checkpoint_path`
- `sample_path`
- `save_interval` = `config.getint("output", "save_interval", fallback=10)`
- `config_id` = `config.getint("output", "config_id", fallback=1234)`
- `tf`
- `dataset` = `MNIST("./data", train=True, download=True, transform=tf)`
- `dataset_train`
- `dataset_val`
- `dataloader_train`
- `dataloader_val`
- `gt`
- `criterion` = `nn.MSELoss()`
- `model`
- `optim` = `torch.optim.Adam(model.parameters(), lr=lr_initial)`
- `accelerator` = `Accelerator()`

6.7.1 Variable Documentation

6.7.1.1 accelerator

```
run.accelerator = Accelerator()
```

6.7.1.2 batch_size

```
run.batch_size = config.getint("training", "batch_size", fallback=128)
```

6.7.1.3 checkpoint_path

```
run.checkpoint_path
```

Initial value:

```
1 = config.get(  
2     "output", "checkpoint_path", fallback="./data/model/checkpoint/"  
3 )
```

6.7.1.4 config

```
run.config = cfg.ConfigParser()
```

6.7.1.5 config_id

```
run.config_id = config.getint("output", "config_id", fallback=1234)
```

6.7.1.6 criterion

```
run.criterion = nn.MSELoss()
```

6.7.1.7 dataloader_train

```
run.dataloader_train
```

Initial value:

```
1 = DataLoader(  
2     dataset_train,  
3     batch_size=batch_size,  
4     shuffle=True,  
5     num_workers=4,  
6     drop_last=True,  
7 )
```

6.7.1.8 dataloader_val

```
run.dataloader_val
```

Initial value:

```
1 = DataLoader(  
2     dataset_val,  
3     batch_size=batch_size,  
4     shuffle=True,  
5     num_workers=4,  
6     drop_last=True,  
7 )
```

6.7.1.9 dataset

```
run.dataset = MNIST("./data", train=True, download=True, transform=tf)
```

6.7.1.10 dataset_train

```
run.dataset_train
```

6.7.1.11 dataset_val

```
run.dataset_val
```

6.7.1.12 degradation

```
run.degradation = config.get("model", "degradation", fallback="gaussian")
```

6.7.1.13 gt

```
run.gt
```

Initial value:

```
1 = CNN(  
2     in_channels=1, expected_shape=(28, 28), n_hidden=n_hidden, act=nn.GELU  
3 )
```

6.7.1.14 input_file

```
run.input_file = sys.argv[1]
```

6.7.1.15 loss_fn

```
run.loss_fn = config.get("model", "loss_fn", fallback="L2")
```

6.7.1.16 lr_initial

```
run.lr_initial = config.getfloat("training", "lr_initial", fallback=2e-4)
```

6.7.1.17 model

```
run.model
```

Initial value:

```
1 = DDPM(  
2     gt=gt, betas=(noise_min, noise_max), n_T=n_T, criterion=criterion  
3 )
```

6.7.1.18 n_epoch

```
run.n_epoch = config.getint("training", "n_epoch", fallback=100)
```

6.7.1.19 n_hidden

```
run.n_hidden = config.get("model", "n_hidden", fallback="16 32 32 16")
```

6.7.1.20 n_T

```
run.n_T = config.getint("model", "n_T", fallback=1000)
```

6.7.1.21 noise_max

```
run.noise_max = config.getfloat("model", "noise_max", fallback=0.02)
```

6.7.1.22 noise_min

```
run.noise_min = config.getfloat("model", "noise_min", fallback=1e-4)
```

6.7.1.23 optim

```
run.optim = torch.optim.Adam(model.parameters(), lr=lr\_initial)
```

6.7.1.24 sample_path

```
run.sample_path
```

Initial value:

```
1 = config.get(  
2     "output", "sample_path", fallback="./data/model/sample/"  
3 )
```

6.7.1.25 save_interval

```
run.save_interval = config.getint("output", "save_interval", fallback=10)
```

6.7.1.26 tf

```
run.tf
```

Initial value:

```
1 = transforms.Compose(  
2     [transforms.ToTensor(), transforms.Normalize((0.5,), (1.0))]  
3 )
```

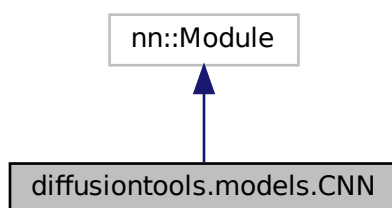

Chapter 7

Class Documentation

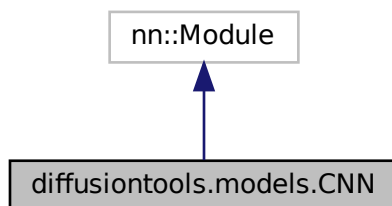
7.1 diffusiontools.models.CNN Class Reference

Defines [CNN](#) class used to reconstruct images during diffusion.

Inheritance diagram for diffusiontools.models.CNN:



Collaboration diagram for diffusiontools.models.CNN:



Public Member Functions

- None `__init__` (self, in_channels, expected_shape=(28, 28), n_hidden=(64, 128, 64), kernel_size=7, last_kernel_size=3, time_embeddings=16, act=nn.GELU)
Initialisation for CNN class.
- torch.Tensor `time_encoding` (self, torch.Tensor t)
Produces embedding vectors for the time steps passed during the input to the network.
- torch.Tensor `forward` (self, torch.Tensor x, torch.Tensor t)
Defines the forward pass for the model.

Public Attributes

- `blocks`
- `time_embed`

7.1.1 Detailed Description

Defines `CNN` class used to reconstruct images during diffusion.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 `__init__()`

```
None diffusiontools.models.CNN.__init__ (
    self,
    in_channels,
    expected_shape = (28, 28),
    n_hidden = (64, 128, 64),
    kernel_size = 7,
    last_kernel_size = 3,
    time_embeddings = 16,
    act = nn.GELU )
```

Initialisation for `CNN` class.

Parameters

<code>in_channels</code>	Number of input channels
<code>expected_shape</code>	image shape as (height, width)
<code>n_hidden</code>	Tuple containing number of hidden channels per hidden layer
<code>kernel_size</code>	Convolution kernel size
<code>last_kernel_size</code>	Used to specify different kernel size in final layer
<code>time_embedding</code>	Determines dimensionality of temporal embeddings for diffusion (which is later doubled before combining with the network)
<code>act</code>	Non-linear activation function to use in the hidden neurons

7.1.3 Member Function Documentation

7.1.3.1 forward()

```
torch.Tensor diffusiontools.models.CNN.forward (
    self,
    torch.Tensor x,
    torch.Tensor t )
```

Defines the forward pass for the model.

Parameters

<i>x</i>	Image stack with shape (batch, chan, height, width)
<i>t</i>	Time step stack with shape (batch,)

Returns

embed Final output of the model

7.1.3.2 time_encoding()

```
torch.Tensor diffusiontools.models.CNN.time_encoding (
    self,
    torch.Tensor t )
```

Produces embedding vectors for the time steps passed during the input to the network.

Parameters

<i>t</i>	Tensor of time steps, one for each batch member
----------	---

Returns

torch.Tensor Embedded vectors that have been passed through trigonometric functions and then learned linear network

7.1.4 Member Data Documentation

7.1.4.1 blocks

```
diffusiontools.models.CNN.blocks
```

7.1.4.2 time_embed

`diffusioontools.models.CNN.time_embed`

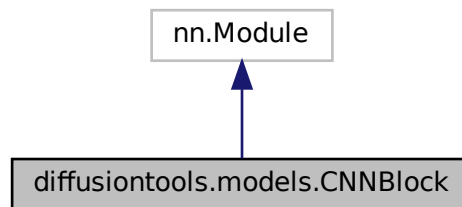
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusioontools/models.py

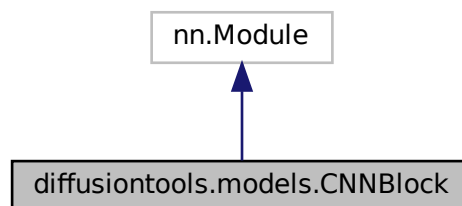
7.2 diffusioontools.models.CNNBlock Class Reference

Class instantiating a convolutional block with layer normalisation and non-linear activation.

Inheritance diagram for `diffusioontools.models.CNNBlock`:



Collaboration diagram for `diffusioontools.models.CNNBlock`:



Public Member Functions

- `def __init__(self, in_channels, out_channels, *expected_shape, act=nn.GELU, kernel_size=7)`
Initialisation for `CNNBlock` class.
- `def forward(self, x)`
Forward pass function for `CNNBlock`.

Public Attributes

- [net](#)

7.2.1 Detailed Description

Class instantiating a convolutional block with layer normalisation and non-linear activation.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 `__init__()`

```
def diffusiontools.models.CNNBlock.__init__ (
    self,
    in_channels,
    out_channels,
    * expected_shape,
    act = nn.GELU,
    kernel_size = 7 )
```

Initialisation for [CNNBlock](#) class.

Parameters

<i>in_channels</i>	Number of input channels
<i>out_channels</i>	Number of output channels
<i>expected_shape</i>	Image shape (height, width)
<i>act</i>	Non-linear activation function
<i>kernel_size</i>	Size of convolutional kernel to use

7.2.3 Member Function Documentation

7.2.3.1 `forward()`

```
def diffusiontools.models.CNNBlock.forward (
    self,
    x )
```

Forward pass function for [CNNBlock](#).

Parameters

x	Input from previous hidden layer
-----	----------------------------------

Returns

torch.tensor Output to pass to next hidden layer

7.2.4 Member Data Documentation

7.2.4.1 net

`diffusiontools.models.CNNBlock.net`

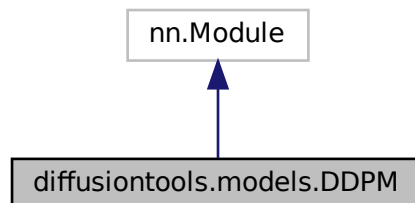
The documentation for this class was generated from the following file:

- `/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py`

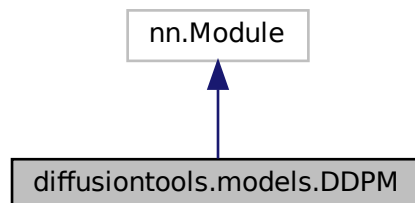
7.3 `diffusiontools.models.DDPM` Class Reference

Defines Gaussian diffusion model class.

Inheritance diagram for `diffusiontools.models.DDPM`:



Collaboration diagram for `diffusiontools.models.DDPM`:



Public Member Functions

- None `__init__` (self, `gt`, Tuple[float, float] `betas`, int `n_T`, nn.Module `criterion`=nn.MSELoss())
Initialisation for DDPM class.
- torch.Tensor `forward` (self, torch.Tensor `x`)
Forward pass for the DDPM model.
- torch.Tensor `sample` (self, int `n_sample`, size, device)
Reverse diffusion sampling for the DDPM model.

Public Attributes

- `gt`
- `n_T`
- `criterion`

7.3.1 Detailed Description

Defines Gaussian diffusion model class.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 `__init__()`

```
None diffusiontools.models.DDPM.__init__ (
    self,
    gt,
    Tuple[float, float] betas,
    int n_T,
    nn.Module criterion = nn.MSELoss() )
```

Initialisation for DDPM class.

Parameters

<code>gt</code>	Trainable model used to approximate the reconstruction process
<code>betas</code>	Determines the extremes of the noise levels along the diffusion process
<code>n_T</code>	Number of discrete diffusion time steps
<code>criterion</code>	Loss function to use for the denoising process

7.3.3 Member Function Documentation

7.3.3.1 forward()

```
torch.Tensor diffusiontools.models.DDPM.forward (
    self,
    torch.Tensor x )
```

Forward pass for the [DDPM](#) model.

Implements Algorithm 18.1 in Understanding Deep Learning, found at <http://udlbook.com>. Note that unusually, this forward method returns the loss function of the predicted error, rather than the error prediction itself.

Parameters

<i>x</i>	Batched input
----------	---------------

Returns

torch.Tensor Loss function evaluated on the predicted error compared to the true error

7.3.3.2 sample()

```
torch.Tensor diffusiontools.models.DDPM.sample (
    self,
    int n_sample,
    size,
    device )
```

Reverse diffusion sampling for the [DDPM](#) model.

Implements Algorithm 18.2 in Understanding Deep Learning, found at <http://udlbook.com>.

Parameters

<i>n_sample</i>	Number of images to generate
<i>size</i>	Image size tuple in the format (channel, height, width)

Returns

z_t Stack of generated image samples

7.3.4 Member Data Documentation

7.3.4.1 criterion

```
diffusiontools.models.DDPM.criterion
```


7.3.4.2 gt

`diffusiontools.models.DDPM.gt`

7.3.4.3 n_T

`diffusiontools.models.DDPM.n_T`

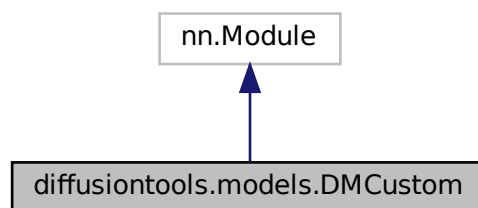
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

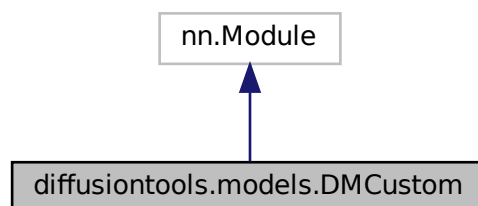
7.4 diffusiontools.models.DMCustom Class Reference

Class defining custom diffusion model.

Inheritance diagram for `diffusiontools.models.DMCustom`:



Collaboration diagram for `diffusiontools.models.DMCustom`:



Public Member Functions

- None `__init__` (self, `gt`, Tuple[float, float] `alphas`, int `n_T`, Tuple[int, int] `size`, nn.Module `criterion`=nn.MSELoss())
Initialisation for `DMCustom` class.
- torch.Tensor `degrade` (self, torch.Tensor `x`, int `t`, device)
Implements custom degradation operation.
- torch.Tensor `forward` (self, torch.Tensor `x`)
Forward pass for the `DMCustom` model.
- torch.Tensor `sample` (self, int `n_sample`, `size`, device)
Reverse diffusion sampling for the `DMCustom` model.

Public Attributes

- `gt`
- `n_T`
- `criterion`
- `size`

7.4.1 Detailed Description

Class defining custom diffusion model.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 `__init__()`

```
None diffusiontools.models.DMCustom.__init__ (
    self,
    gt,
    Tuple[float, float] alphas,
    int n_T,
    Tuple[int, int] size,
    nn.Module criterion = nn.MSELoss() )
```

Initialisation for `DMCustom` class.

Parameters

<code>gt</code>	Trainable model used to approximate the reconstruction process
<code>alphas</code>	Determines the extremes of the noise levels along the diffusion process
<code>n_T</code>	Number of discrete diffusion time steps
<code>size</code>	Image size tuple in the format (channel, height, width)
<code>criterion</code>	Loss function to use for the denoising process

7.4.3 Member Function Documentation

7.4.3.1 degrade()

```
torch.Tensor diffusiontools.models.DMCustom.degrade (
    self,
    torch.Tensor x,
    int t,
    device )
```

Implements custom degradation operation.

Parameters

<i>x</i>	Batched image tensor
<i>t</i>	Integer discrete time-step; note that the same degradation is applied to all images in the stack
<i>device</i>	torch.device object which enables GPU or CPU to be specified

Returns

z_t Stack of images with degradation applied

7.4.3.2 forward()

```
torch.Tensor diffusiontools.models.DMCustom.forward (
    self,
    torch.Tensor x )
```

Forward pass for the [DMCustom](#) model.

Implements Algorithm 18.1 in Understanding Deep Learning, found at <http://udlbook.com>. Note that unusually, this forward method returns the loss function of the predicted error, rather than the error prediction itself.

Parameters

<i>x</i>	Batched input
----------	---------------

Returns

torch.Tensor Loss function evaluated on the predicted error compared to the true error

7.4.3.3 sample()

```
torch.Tensor diffusiontools.models.DMCustom.sample (
    self,
    int n_sample,
    size,
    device )
```

Reverse diffusion sampling for the [DMCustom](#) model.

Implements Algorithm 18.2 in Understanding Deep Learning, found at <http://udlbook.com>.

Parameters

<i>n_sample</i>	Number of images to generate
<i>size</i>	Image size tuple in the format (channel, height, width)

Returns

z_t Stack of generated image samples

7.4.4 Member Data Documentation

7.4.4.1 criterion

```
diffusiontools.models.DMCustom.criterion
```

7.4.4.2 gt

```
diffusiontools.models.DMCustom.gt
```

7.4.4.3 n_T

```
diffusiontools.models.DMCustom.n_T
```

7.4.4.4 size

```
diffusiontools.models.DMCustom.size
```

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

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

Chapter 8

File Documentation

8.1 `/home/jhughes2712/projects/m2_↔assessment/jh2284/src/diffusiontools/__init__.py` File Reference

Namespaces

- [diffusiontools](#)

8.2 `/home/jhughes2712/projects/m2_↔assessment/jh2284/src/diffusiontools/analysis.py` File Reference

Module containing procedures used to analyse and compare trained diffusion models.

Namespaces

- [diffusiontools.analysis](#)

Functions

- def [diffusiontools.analysis.plot_learning_curve](#) (List[float] losses_train, List[float] losses_val, str title, str filename)
Plot changes in loss function across epochs for model training.
- def [diffusiontools.analysis.compute_image_diffusion](#) (DDPM model, List[float] image_fractions, int n_sample, size, device)
Save intermediate image generations at specified parts of the Gaussian diffusion process.
- torch.Tensor [diffusiontools.analysis.compute_image_diffusion_custom](#) (DMCustom model, List[int] image_↔fractions, int n_sample, size, device, bool direct=False)
Save intermediate image generations at specified parts of the custom diffusion process.
- def [diffusiontools.analysis.plot_image_diffusion](#) (model, List[int] image_fractions, int n_sample, size, device, str title, str filename)
Plot intermediate image generations at specified parts of the custom diffusion process.
- def [diffusiontools.analysis.plot_samples](#) (np.ndarray samples, int n_row, int n_col, str filename)

- *Produce matrix of synthetic image samples.*
- def [diffusio.tools.analysis.plot_gt_direct_diffusion](#) (np.ndarray direct_sample, np.ndarray diffusion_sample, int n_plot, str filename)
- *Compare direct and diffused reconstructions.*
- def [diffusio.tools.analysis.compute_variance](#) (np.ndarray sample)
- *Compute the total variance of a sample of vectors.*
- def [diffusio.tools.analysis.degradation_demo](#) (DMCustom model, device, List[int] t_list, str title, str filename)
- *Plot MNIST images of digits 1 to 4 with varying custom degradation levels.*
- def [diffusio.tools.analysis.compute_tsne_kl_div](#) (List[np.ndarray] samples, List[str] labels)
- *Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.*
- def [diffusio.tools.analysis.plot_sample_tsne](#) (np.ndarray density_gt, np.ndarray sample_1_fitted, np.ndarray sample_2_fitted, np.ndarray density_1, np.ndarray density_2, str label_1, str label_2, str filename)
- *Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.*
- def [diffusio.tools.analysis.plot_mnist_tsne](#) (str filename)
- *Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.*

8.2.1 Detailed Description

Module containing procedures used to analyse and compare trained diffusion models.

8.3 /home/jhughes2712/projects/m2_ assessment/jh2284/src/diffusio.tools/models.py File Reference

Module containing classes and procedures used to build diffusion models.

Classes

- class [diffusio.tools.models.CNNBlock](#)
- *Class instantiating a convolutional block with layer normalisation and non-linear activation.*
- class [diffusio.tools.models.CNN](#)
- *Defines [CNN](#) class used to reconstruct images during diffusion.*
- class [diffusio.tools.models.DDPM](#)
- *Defines Gaussian diffusion model class.*
- class [diffusio.tools.models.DMCustom](#)
- *Class defining custom diffusion model.*

Namespaces

- [diffusio.tools.models](#)

Functions

- Dict[str, torch.Tensor] [diffusio.tools.models.ddpm_schedules](#) (float beta1, float beta2, int T)
- *Returns pre-computed schedules for [DDPM](#) sampling with a linear noise schedule.*

8.3.1 Detailed Description

Module containing classes and procedures used to build diffusion models.

Script used to evaluate the quality of diffusion samples at intermediate stages of training.

Script used to produce plots and statistics related to final trained diffusion models.

8.4 /home/jhughes2712/projects/m2_[↔]assessment/jh2284/src/diffusio[↔]ntools/train.py File Reference

Module containing procedures used to train and save diffusion models.

Namespaces

- [diffusio[↔]ntools.train](#)

Functions

- def [diffusio[↔]ntools.train.train_model](#) (nn.Module ddpm, torch.optim.Optimizer optim, DataLoader dataloader_[↔]_train, DataLoader dataloader_val, Accelerator accelerator, int n_epoch, int save_interval, str sample_path, str checkpoint_path, str config_id)
Handles the training process for a diffusion model.

8.4.1 Detailed Description

Module containing procedures used to train and save diffusion models.

8.5 /home/jhughes2712/projects/m2_[↔]assessment/jh2284/src/final_[↔]analysis.py File Reference

Namespaces

- [final_[↔]analysis](#)

Variables

- `final_analysis.device` = `torch.device("cuda" if torch.cuda.is_available() else "cpu")`
- `final_analysis.gt_1`
- `final_analysis.model_1` = `DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)`
- `final_analysis.ckpt_0001_0490`
- `final_analysis.gt_2`
- `final_analysis.model_2` = `DDPM(gt=gt_2, betas=(0.001, 0.2), n_T=100).to(device)`
- `final_analysis.ckpt_0002_0490`
- `final_analysis.gt_9`
- `final_analysis.model_9`
- `final_analysis.ckpt_0009_0080`
- `final_analysis.gt_10`
- `final_analysis.model_10`
- `final_analysis.ckpt_0010_0180`
- `final_analysis.ckpt_0001_0120`
- `final_analysis.ckpt_0002_0080`
- `final_analysis.ckpt_0009_0020`
- `final_analysis.ckpt_0010_0140`
- `int final_analysis.SAMPLE_SIZE` = 200
- `final_analysis._`
- `final_analysis.sample_1`
- `final_analysis.sample_2`
- `final_analysis.sample_9`
- `final_analysis.sample_9_direct`
- `final_analysis.direct`
- `final_analysis.sample_10`
- `final_analysis.sample_10_direct`
- `final_analysis.total_var_1` = `compute_variance(sample_1[0])`
- `final_analysis.total_var_2` = `compute_variance(sample_2[0])`
- `final_analysis.total_var_9` = `compute_variance(sample_9[0])`
- `final_analysis.total_var_10` = `compute_variance(sample_10[0])`
- `final_analysis.density_gt`
- `final_analysis.samples_fitted`
- `final_analysis.samples_densities`

8.6 /home/jhughes2712/projects/m2_↵ assessment/jh2284/src/intermediate_analysis.py File Reference

Namespaces

- `intermediate_analysis`

Variables

- `intermediate_analysis.device` = `torch.device("cuda" if torch.cuda.is_available() else "cpu")`
- `intermediate_analysis.ckpt_0001`
- `intermediate_analysis.gt_1`
- `intermediate_analysis.model_1` = `DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)`
- `intermediate_analysis.ckpt_0002`
- `intermediate_analysis.gt_2`
- `intermediate_analysis.model_2` = `DDPM(gt=gt_2, betas=(0.0001, 0.02), n_T=1000).to(device)`
- `intermediate_analysis.ckpt_0009`
- `intermediate_analysis.gt_9`
- `intermediate_analysis.model_9`
- `intermediate_analysis.ckpt_0010`
- `intermediate_analysis.gt_10`
- `intermediate_analysis.model_10`

8.7 /home/jhughes2712/projects/m2_assessment/jh2284/src/run.py File Reference

Script used to create and train diffusion model, based on parameters provided via the config.ini file.

Namespaces

- `run`

Variables

- `run.config` = `cfg.ConfigParser()`
- `run.input_file` = `sys.argv[1]`
- `run.noise_min` = `config.getfloat("model", "noise_min", fallback=1e-4)`
- `run.noise_max` = `config.getfloat("model", "noise_max", fallback=0.02)`
- `run.n_T` = `config.getint("model", "n_T", fallback=1000)`
- `run.degradation` = `config.get("model", "degradation", fallback="gaussian")`
- `run.n_hidden` = `config.get("model", "n_hidden", fallback="16 32 32 16")`
- `run.loss_fn` = `config.get("model", "loss_fn", fallback="L2")`
- `run.n_epoch` = `config.getint("training", "n_epoch", fallback=100)`
- `run.batch_size` = `config.getint("training", "batch_size", fallback=128)`
- `run.lr_initial` = `config.getfloat("training", "lr_initial", fallback=2e-4)`
- `run.checkpoint_path`
- `run.sample_path`
- `run.save_interval` = `config.getint("output", "save_interval", fallback=10)`
- `run.config_id` = `config.getint("output", "config_id", fallback=1234)`
- `run.tf`
- `run.dataset` = `MNIST("./data", train=True, download=True, transform=tf)`
- `run.dataset_train`
- `run.dataset_val`
- `run.dataloader_train`
- `run.dataloader_val`
- `run.gt`
- `run.criterion` = `nn.MSELoss()`
- `run.model`
- `run.optim` = `torch.optim.Adam(model.parameters(), lr=lr_initial)`
- `run.accelerator` = `Accelerator()`

8.7.1 Detailed Description

Script used to create and train diffusion model, based on parameters provided via the config.ini file.

Author

Created by J. Hughes on 18/03/2024.

Index

/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/__init__.py, 45
intermediate_analysis, 26
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/analysis.py, 45
final_analysis, 21
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py, 46
final_analysis, 21
/home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/image_diffusion.py, 47
diffusio[n]tools.analysis, 12
/home/jhughes2712/projects/m2_assessment/jh2284/src/final_image_diffusion_custom.py, 47
diffusio[n]tools.analysis, 12
/home/jhughes2712/projects/m2_assessment/jh2284/src/intermediate_analysis.py, 48
diffusio[n]tools.analysis, 13
/home/jhughes2712/projects/m2_assessment/jh2284/src/run.py, 49
compute_variance
diffusio[n]tools.analysis, 13
—
final_analysis, 19
__init__
diffusio[n]tools.models.CNN, 34
diffusio[n]tools.models.CNNBlock, 37
diffusio[n]tools.models.DDPM, 39
diffusio[n]tools.models.DMCustom, 42
accelerator
run, 28
batch_size
run, 29
blocks
diffusio[n]tools.models.CNN, 35
checkpoint_path
run, 29
ckpt_0001
intermediate_analysis, 25
ckpt_0001_0120
final_analysis, 19
ckpt_0001_0490
final_analysis, 20
ckpt_0002
intermediate_analysis, 25
ckpt_0002_0080
final_analysis, 20
ckpt_0002_0490
final_analysis, 20
ckpt_0009
intermediate_analysis, 26
ckpt_0009_0020
final_analysis, 20
ckpt_0009_0080
final_analysis, 20
config
run, 29
config_id
run, 29
criterion
diffusio[n]tools.models.DDPM, 40
diffusio[n]tools.models.DMCustom, 44
run, 29
dataloader_train
run, 29
dataloader_val
run, 29
dataset
run, 30
dataset_train
run, 30
dataset_val
run, 30
ddpm_schedules
diffusio[n]tools.models, 17
degradation
run, 30
degradation_demo
diffusio[n]tools.analysis, 14
degrade
diffusio[n]tools.models.DMCustom, 43
density_gt
final_analysis, 21
device
final_analysis, 21
intermediate_analysis, 26
diffusio[n]tools, 11
diffusio[n]tools.analysis, 11
compute_image_diffusion, 12
compute_image_diffusion_custom, 12

- compute_tsne_kl_div, 13
- compute_variance, 13
- degradation_demo, 14
- plot_gt_direct_diffusion, 14
- plot_image_diffusion, 15
- plot_learning_curve, 15
- plot_mnist_tsne, 15
- plot_sample_tsne, 16
- plot_samples, 16
- diffusiontools.models, 17
 - ddpm_schedules, 17
- diffusiontools.models.CNN, 33
 - __init__, 34
 - blocks, 35
 - forward, 35
 - time_embed, 35
 - time_encoding, 35
- diffusiontools.models.CNNBlock, 36
 - __init__, 37
 - forward, 37
 - net, 38
- diffusiontools.models.DDPM, 38
 - __init__, 39
 - criterion, 40
 - forward, 39
 - gt, 40
 - n_T, 41
 - sample, 40
- diffusiontools.models.DMCustom, 41
 - __init__, 42
 - criterion, 44
 - degrade, 43
 - forward, 43
 - gt, 44
 - n_T, 44
 - sample, 43
 - size, 44
- diffusiontools.train, 18
 - train_model, 18
- direct
 - final_analysis, 21
- final_analysis, 19
 - _, 19
 - ckpt_0001_0120, 19
 - ckpt_0001_0490, 20
 - ckpt_0002_0080, 20
 - ckpt_0002_0490, 20
 - ckpt_0009_0020, 20
 - ckpt_0009_0080, 20
 - ckpt_0010_0140, 21
 - ckpt_0010_0180, 21
 - density_gt, 21
 - device, 21
 - direct, 21
 - gt_1, 21
 - gt_10, 22
 - gt_2, 22
 - gt_9, 22
 - model_1, 22
 - model_10, 22
 - model_2, 23
 - model_9, 23
 - sample_1, 23
 - sample_10, 23
 - sample_10_direct, 23
 - sample_2, 23
 - sample_9, 24
 - sample_9_direct, 24
 - SAMPLE_SIZE, 24
 - samples_densities, 24
 - samples_fitted, 24
 - total_var_1, 24
 - total_var_10, 24
 - total_var_2, 24
 - total_var_9, 25
- forward
 - diffusiontools.models.CNN, 35
 - diffusiontools.models.CNNBlock, 37
 - diffusiontools.models.DDPM, 39
 - diffusiontools.models.DMCustom, 43
- gt
 - diffusiontools.models.DDPM, 40
 - diffusiontools.models.DMCustom, 44
 - run, 30
- gt_1
 - final_analysis, 21
 - intermediate_analysis, 26
- gt_10
 - final_analysis, 22
 - intermediate_analysis, 26
- gt_2
 - final_analysis, 22
 - intermediate_analysis, 27
- gt_9
 - final_analysis, 22
 - intermediate_analysis, 27
- input_file
 - run, 30
- intermediate_analysis, 25
 - ckpt_0001, 25
 - ckpt_0002, 25
 - ckpt_0009, 26
 - ckpt_0010, 26
 - device, 26
 - gt_1, 26
 - gt_10, 26
 - gt_2, 27
 - gt_9, 27
 - model_1, 27
 - model_10, 27
 - model_2, 27
 - model_9, 28
- loss_fn
 - run, 31

- lr_initial
 - run, 31
- model
 - run, 31
- model_1
 - final_analysis, 22
 - intermediate_analysis, 27
- model_10
 - final_analysis, 22
 - intermediate_analysis, 27
- model_2
 - final_analysis, 23
 - intermediate_analysis, 27
- model_9
 - final_analysis, 23
 - intermediate_analysis, 28
- n_epoch
 - run, 31
- n_hidden
 - run, 31
- n_T
 - diffusiontools.models.DDPM, 41
 - diffusiontools.models.DMCustom, 44
 - run, 31
- net
 - diffusiontools.models.CNNBlock, 38
- noise_max
 - run, 31
- noise_min
 - run, 32
- optim
 - run, 32
- plot_gt_direct_diffusion
 - diffusiontools.analysis, 14
- plot_image_diffusion
 - diffusiontools.analysis, 15
- plot_learning_curve
 - diffusiontools.analysis, 15
- plot_mnist_tsne
 - diffusiontools.analysis, 15
- plot_sample_tsne
 - diffusiontools.analysis, 16
- plot_samples
 - diffusiontools.analysis, 16
- run, 28
 - accelerator, 28
 - batch_size, 29
 - checkpoint_path, 29
 - config, 29
 - config_id, 29
 - criterion, 29
 - dataloader_train, 29
 - dataloader_val, 29
 - dataset, 30
 - dataset_train, 30
 - dataset_val, 30
 - degradation, 30
 - gt, 30
 - input_file, 30
 - loss_fn, 31
 - lr_initial, 31
 - model, 31
 - n_epoch, 31
 - n_hidden, 31
 - n_T, 31
 - noise_max, 31
 - noise_min, 32
 - optim, 32
 - sample_path, 32
 - save_interval, 32
 - tf, 32
- sample
 - diffusiontools.models.DDPM, 40
 - diffusiontools.models.DMCustom, 43
- sample_1
 - final_analysis, 23
- sample_10
 - final_analysis, 23
- sample_10_direct
 - final_analysis, 23
- sample_2
 - final_analysis, 23
- sample_9
 - final_analysis, 24
- sample_9_direct
 - final_analysis, 24
- sample_path
 - run, 32
- SAMPLE_SIZE
 - final_analysis, 24
- samples_densities
 - final_analysis, 24
- samples_fitted
 - final_analysis, 24
- save_interval
 - run, 32
- size
 - diffusiontools.models.DMCustom, 44
- tf
 - run, 32
- time_embed
 - diffusiontools.models.CNN, 35
- time_encoding
 - diffusiontools.models.CNN, 35
- total_var_1
 - final_analysis, 24
- total_var_10
 - final_analysis, 24
- total_var_2
 - final_analysis, 24
- total_var_9

final_analysis, [25](#)
train_model
diffusio.tools.train, [18](#)