simages Documentation

Release 19.0.2.post1

Justin Shenk

Jun 17, 2019

Getting Started

Similar image detection in Python

simages allows detecting similar images in an image folder or numpy array.

CHAPTER 1

Description

Detect similar images (eg, duplicates) in an image folder. Behind the curtain, *simages* uses a PyTorch autoencoder to train embeddings. The embeddings are compared with each other with closely to create a distance matrix. The closest pairs of images are then presented on screen.

If you use *simages* in your publications, please cite:

```
@misc{justin_shenk_2019_3237830,
  author       = {Justin Shenk},
  title        = {justinshenk/simages: v19.0.1},
  month        = jun,
  year         = 2019,
  doi          = {10.5281/zenodo.3237830},
  url          = {https://doi.org/10.5281/zenodo.3237830}
}
```

1.1 Installation

Installing simages is pretty simple.

If you haven't got it, obtain Python (version 3.6 or greater).

Install with pip:

```
pip install simages
```

If you wish to install the latest development version, clone the GitHub repository and use the setup script:

```
git clone https://github.com/justinshenk/simages.git
cd simages
pip install .
```

Next you can go to *the Command Line Interface*.

1.1.1 Dependencies

Installation with pip should also include all dependencies, but a complete list is

- numpy
- matplotlib
- closely
- torch
- torchvision

The optional web and *simages command line interface* requires installing mongodb:

```
sudo apt -y install mongodb-server
# Disable Mongo service autostart
sudo systemctl disable mongodb.service
# Stop Mongo service
sudo service mongodb stop
```

To install optional dependencies (PyMongo and Flask) run:

```
pip install 'simages[all]'
```

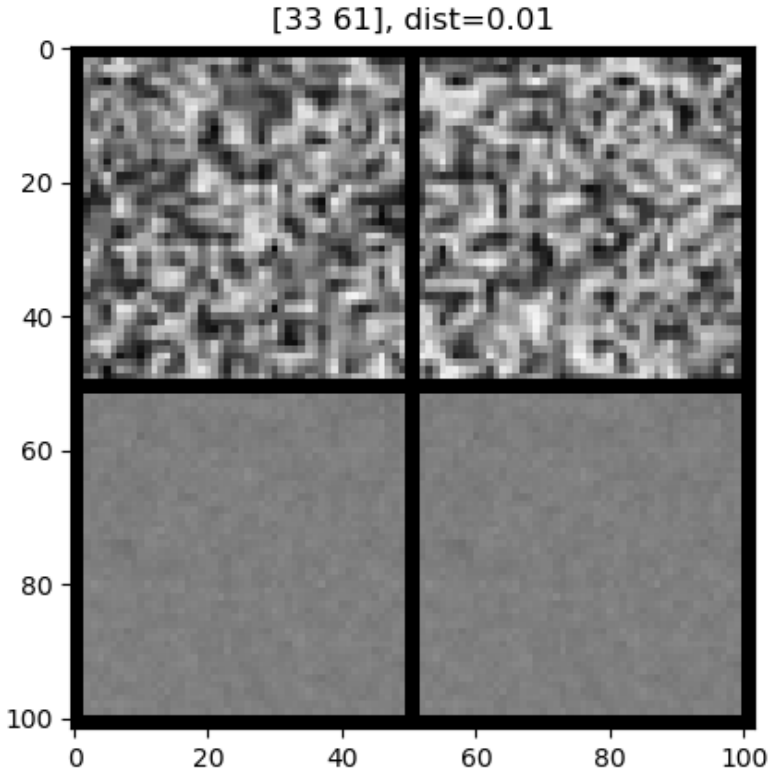
1.2 Gallery

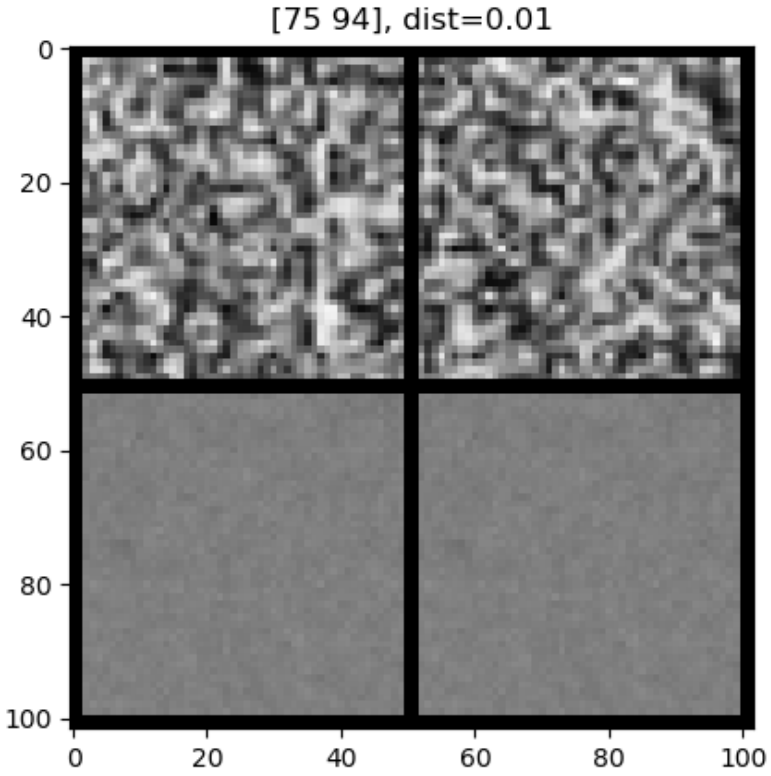
A gallery of examples

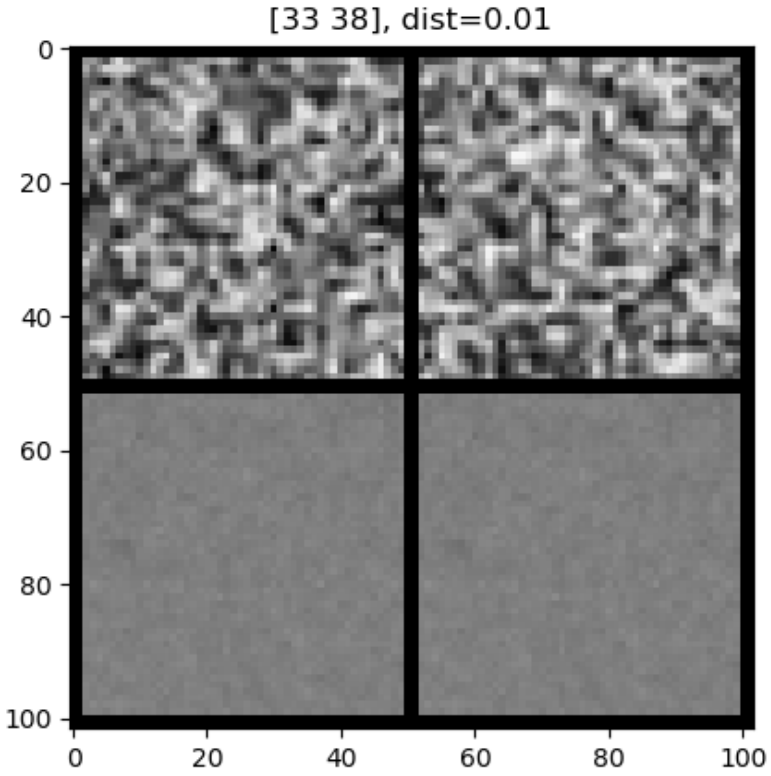
Note: Click *here* to download the full example code

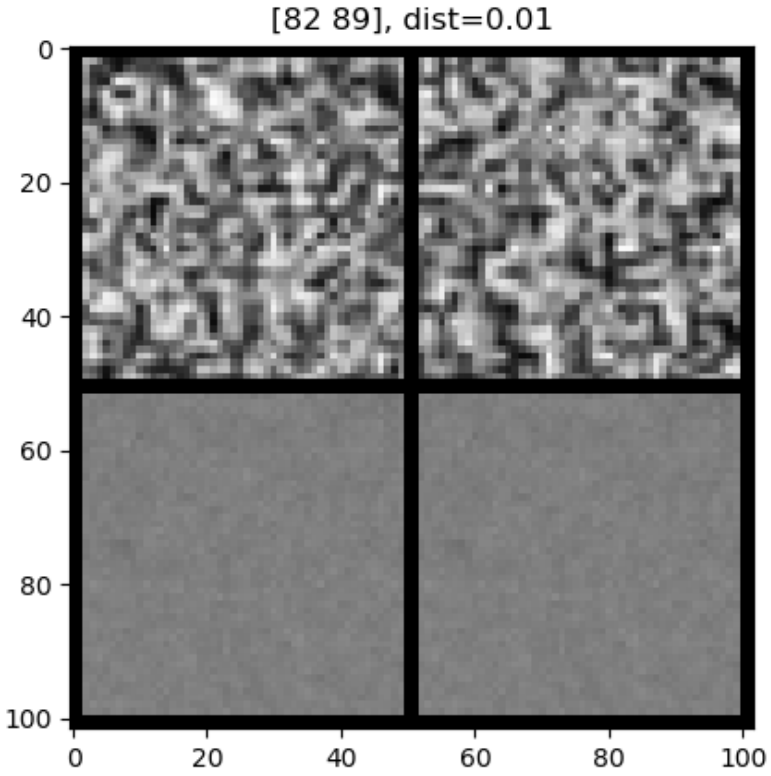
1.2.1 Comparing

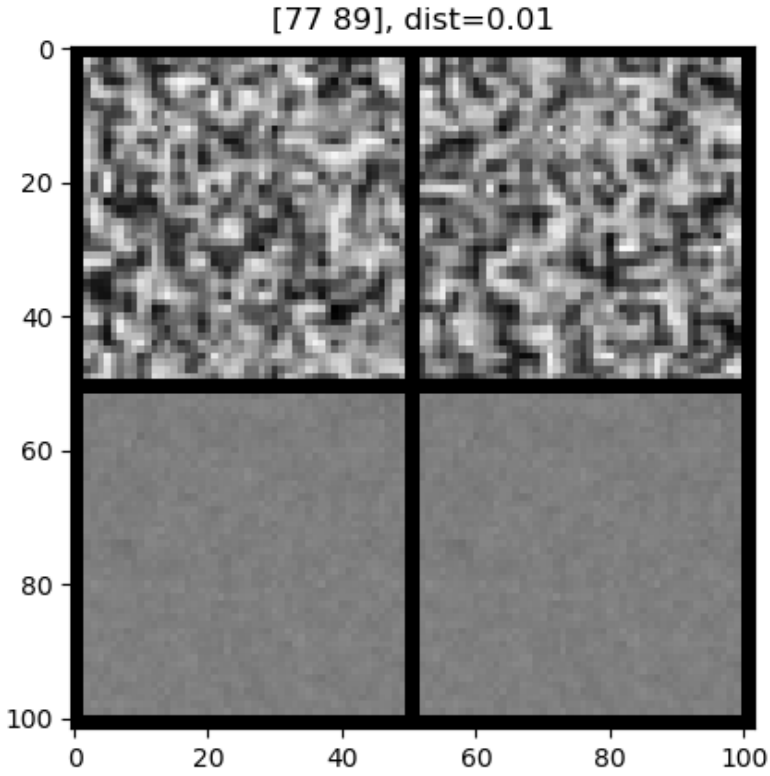
simages allows comparing images using various methods.


[33 61], dist=0.01
0
20
40
60
80
100
0
20
40
60
80
100


[75 94], dist=0.01
0
20
40
60
80
100
0
20
40
60
80
100


[33 38], dist=0.01
0
20
40
60
80
100
0
20
40
60
80
100


[82 89], dist=0.01
0
20
40
60
80
100
0
20
40
60
80
100



-

```
import simages
import numpy as np

X = np.random.random((100, 28, 28))
simages.find_duplicates(X, num_channels=1, show=True)
```

Total running time of the script: (0 minutes 2.831 seconds)

Note: Click *here* to download the full example code

1.2.2 Linkage

simages allows visualizing the similarity of images in a dataset using *simages.linkageplot()*.

Ordered

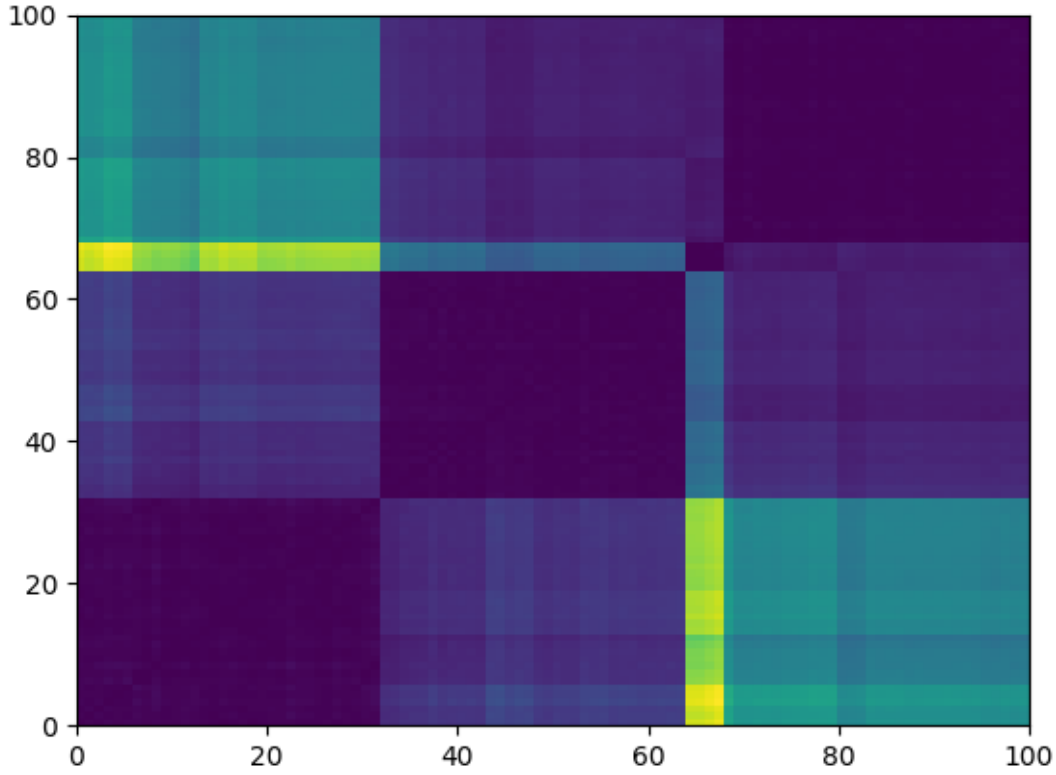
Show ordered linkage plot

```
import simages
from simages import linkageplot
import numpy as np

```

(continues on next page)

(continued from previous page)

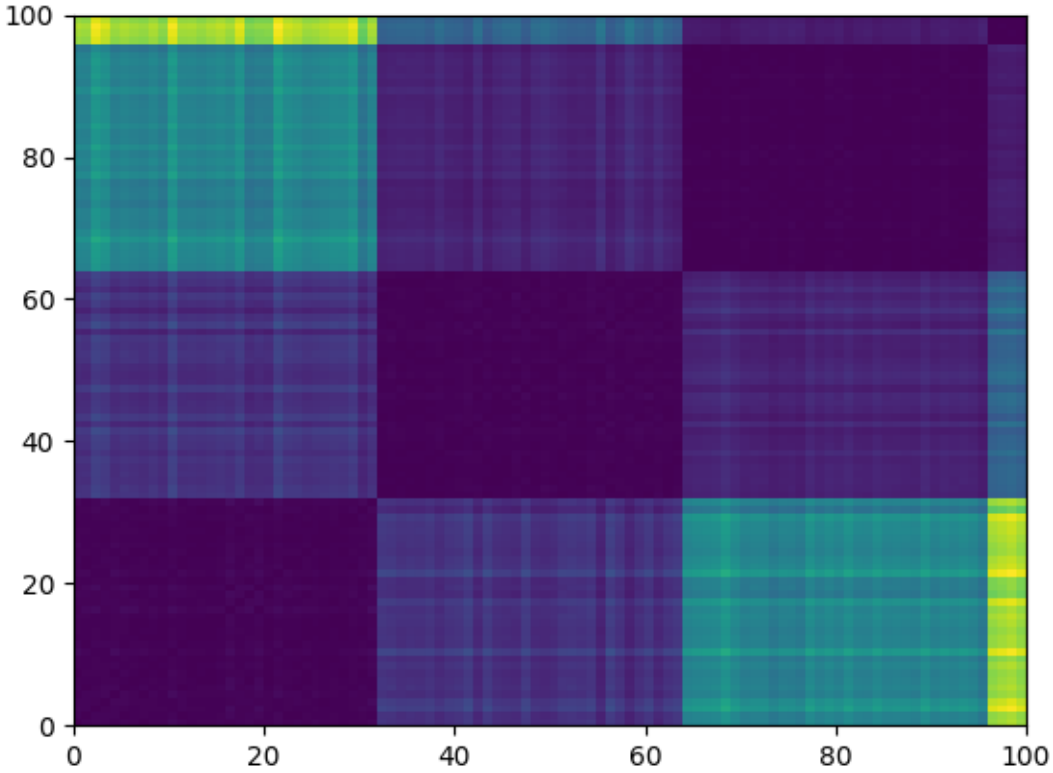
```
X = np.random.random((100, 28, 28))
embeddings = simages.EmbeddingExtractor(X, num_channels=1).embeddings

linkageplot(embeddings)
```



Unordered

Show unordered linkage plot

```
linkageplot(embeddings, ordered=False)
```



Total running time of the script: (0 minutes 3.070 seconds)

1.3 Running simages from the console

simages can be run locally in the terminal with `simages-show`.

Usage:

```
simages-show --data-dir .
```

See all the options for `simages-show` with `simages-show --help`:

Find similar pairs of images in a folder

```
usage: simages-show [-h] [--data-dir DATA_DIR] [--show-train]
                    [--epochs EPOCHS] [--num-channels NUM_CHANNELS]
                    [--pairs PAIRS] [--zdim ZDIM]
```

1.3.1 Named Arguments

--data-dir, -d	Folder containing image data
--show-train, -t	Show training of embedding extractor every epoch

--epochs, -e	Number of passes of dataset through model for training. More is better but takes more time. Default: 2
--num-channels, -c	Number of channels for data (1 for grayscale, 3 for color) Default: 3
--pairs, -p	Number of pairs of images to show Default: 10
--zdim, -z	Compression bits (bigger generally performs better but takes more time) Default: 8

`simages-show` calls *`find_duplicates()`*:

`simages.main.`**`find_duplicates`**(*input: Union[str, numpy.ndarray], n: int = 5, num_epochs: int = 2, num_channels: int = 3, show: bool = False, show_train: bool = False, show_path: bool = True, z_dim: int = 8, db=None, **kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]

Find duplicates in dataset. Either np.ndarray or path to image folder must be specified as *input*.

Parameters

- **`input`** (*`str or np.ndarray`*) – folder directory or N x C x H x W array
- **`n`** (*`int`*) – number of closest pairs to identify
- **`num_epochs`** (*`int`*) – how long to train the autoencoder (more is generally better)
- **`show`** (*`bool`*) – display the closest pairs
- **`show_train`** (*`bool`*) – show output every
- **`show_path`** (*`bool`*) – show image paths of duplicates instead of index
- **`z_dim`** (*`int`*) – size of compression (more is generally better, but slower)
- **`kwargs`** (*`dict`*) – etc, passed to *EmbeddingExtractor*

Returns indices for closest pairs of images distances (np.ndarray): distances of each pair to each other

Return type pairs (np.ndarray)

1.3.2 Web Interface (optional)

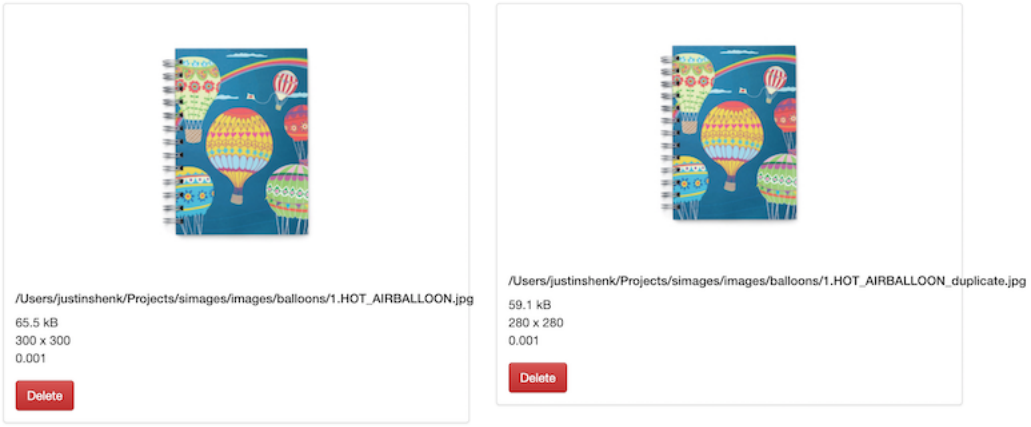
Alternatively, removing duplicate images in a dataset interactively is easy with `simages`.

- Install mongodb on your system.
- Install additional dependencies (Flask and PyMongo) with `pip install "simages[all]"`
- Add images to the database via `simages add {image_folder_path}`.
- Find duplicates and run the web server with `simages find {image_folder_path}`.

Add your pictures to the database (this will take some time depending on the number of pictures)

```
simages add <images_folder_path>
```

A webpage will come up with all of the similar or duplicate pictures:




```
simages find <images_folder_path>


Usage:
    simages add <path> ... [--db=<db_path>] [--parallel=<num_processes>]
    simages remove <path> ... [--db=<db_path>]
    simages clear [--db=<db_path>]
    simages show [--db=<db_path>]
    simages find <path> [--print] [--delete] [--match-time] [--trash=<trash_path>] [--
↪db=<db_path>] [--epochs=<epochs>]
    simages -h | --help
Options:
    -h, --help                Show this screen
    --db=<db_path>            The location of the database or a MongoDB URI.␣
↪(default: ./db)
    --parallel=<num_processes> The number of parallel processes to run to hash the␣
↪image
                               files (default: number of CPUs).
    find:
        --print               Only print duplicate files rather than displaying HTML␣
↪file
        --delete              Move all found duplicate pictures to the trash. This␣
↪option takes priority over --print.
        --match-time          Adds the extra constraint that duplicate images must␣
↪have the
                              same capture times in order to be considered.
        --trash=<trash_path>  Where files will be put when they are deleted (default:␣
↪./Trash)
        --epochs=<epochs>     Epochs for training [default: 2]
```

`simages` calls *`cli()`*.

1.4 Loading Data

Data is loaded using the *`EmbeddingExtractor`* class.

`EmbeddingExtractor` is used to extract embeddings by

- Train an autoencoder on the images

- Identify similar images from the embeddings of the autoencoder
- Plot and visualize the results

Dataset can be provided as a numpy array or as an image folder path.

class simages.extractor.**EmbeddingExtractor**(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)

Extract embeddings from data with models and allow visualization.

trainloader

Type torch loader

evalloader

Type torch loader

model

Type torch.nn.Module

embeddings

Type np.ndarray

Numpy Array

Load data with:

```
from simages import EmbeddingExtractor
import numpy as np

# Create grayscale (1-channel) samples
X = np.random.random((100,28,28))
extractor =  EmbeddingExtractor(X, num_channels=1)

# Find duplicates
pairs, distances = extractor.find_duplicates()
```

Image Folder:

```
from simages import EmbeddingExtractor

# Point to Folder
data_dir = "downloads"
extractor =  EmbeddingExtractor(data_dir)

# Find duplicates
pairs, distances = extractor.find_duplicates()

# Show duplicates
extractor.show_duplicates(n=5)
```

Duplicates can be identified using the `simages` command:

```
$ simages add `{image_folder}`

$ simages find `{image_folder}`
```

Duplicates can be deleted on the webserver as described at *Removing Duplicates*.

1.5 Building Embeddings

Embeddings are extracted from the images by training a convolutional autoencoder.

The models available are listed in `models`. The default autoencoder is a 3-layer convolutional autoencoder, '~simages.BasicAutoencoder:

```
from simages import EmbeddingExtractor

extractor = EmbeddingExtractor(image_dir)
```

`extractor` allows identifying images corresponding to the embeddings.

For example, if `extractor.duplicates()` returns `pairs [2, 3]`, the images corresponding to embeddings 2 and 3 can be viewed with *simages.extractor.EmbeddingExtractor.image_paths()*:

`EmbeddingExtractor.`**`image_paths`**(*indices*, *short=True*)

Get path to image at *index* of eval/embedding

Parameters

- **`Union[int,list]`** (*indices*) – indices of embeddings in dataset
- **`short`** (*bool*) – truncate filepath to 30 charachters

Returns paths to images in image folder

Return type paths (str or list of str)

After building the embeddings, the embeddings can be extracted with:

```
emeddings = extractor.embeddings
```

Extraction is performed by *EmbeddingExtractor*:

`class` simages.extractor.**`EmbeddingExtractor`**(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)

Bases: `object`

Extract embeddings from data with models and allow visualization.

`trainloader`

Type torch loader

`evalloader`

Type torch loader

model

Type torch.nn.Module

embeddings

Type np.ndarray

__init__(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)

Inits EmbeddingExtractor with input, either *str* or *np.ndarray*, performs training and validation.

Parameters

- **input** (*np.ndarray or str*) – data
- **num_channels** (*int*) – grayscale = 1, color = 3
- **num_epochs** (*int*) – more is better (generally)
- **batch_size** (*int*) – number of images per batch
- **show** (*bool*) – show closest pairs
- **show_path** (*bool*) – show path of duplicates
- **show_train** (*bool*) – show intermediate training results
- **z_dim** (*int*) – compression size
- **metric** (*str*) – distance metric for `scipy.spatial.distance.cdist()` (eg, euclidean, cosine, hamming, etc.)
- **model** (*torch.nn.Module, optional*) – class implementing same methods as *BasicAutoencoder*
- **db_conn_string** (*str*) – Mongodb connection string
- **kwargs** (*dict*) –

get_image(*index: int*) → torch.Tensor

train()

Train autoencoder to build embeddings of dataset. Final embeddings are created in *eval()*.

eval()

Evaluate reconstruction of embeddings built in *train*.

duplicates(*n: int = 10*, *quantile: float = None*) → Tuple[numpy.ndarray, numpy.ndarray]

Identify *n* closest pairs of images, or quantile (for example, closest 0.05).

Parameters

- **n** (*int*) – number of pairs
- **quantile** (*float*) – quantile of total combination (suggested range: 0.001 - 0.01)

static channels_last(*img: numpy.ndarray*) → numpy.ndarray

Move channels from first to last by swapping axes.

show(*img: Union[torch.Tensor, numpy.ndarray], title: str = '', block: bool = True, y_labels=None, unnormalize=True*)

Plot *img* with *title*.

Parameters

- **img** (*torch.Tensor or np.ndarray*) – Image to plot
- **title** (*str*) – plot title
- **block** (*bool*) – block matplotlib plot until window closed

show_images(*indices: Union[list, int], title=''*)
Plot images (from validation data) at *indices* with *title*

image_paths(*indices*, *short=True*)
Get path to image at *index* of eval/embedding

Parameters

- **Union[int,list]** (*indices*) – indices of embeddings in dataset
- **short** (*bool*) – truncate filepath to 30 charachters

Returns paths to images in image folder

Return type paths (str or list of str)

show_duplicates(*n=5*, *path=False) -> (<class 'numpy.ndarray'>*, *<class 'numpy.ndarray'>*)
Show duplicates from comparison of embeddings. Uses *closely* package to get pairs.

Parameters

- **n** (*int*) – how many closest pairs to identify
- **path** (*bool*) – Plot pairs of images with abbreviated paths

Returns pairs as indices distances (np.ndarray): distances of pairs

Return type pairs (np.ndarray)

unnormalize(*image: torch.Tensor*) → torch.Tensor
Unnormalize an image.

Parameters **image** (torch.Tensor) –

Returns image (torch.Tensor)

decode(*embedding: Optional[numpy.ndarray] = None*, *index: Optional[int] = None*, *show: bool = False*, *astensor: bool = False*) → numpy.ndarray
Decode embeddings at *index* or pass *embedding* directly

Parameters

- **embedding** (*np.ndarray, optional*) – embedding of image
- **index** (*int*) – index (of validation set / embeddings) to decode
- **show** (*bool*) – plot the results
- **astensor** (*bool*) – keep as torch.Tensor

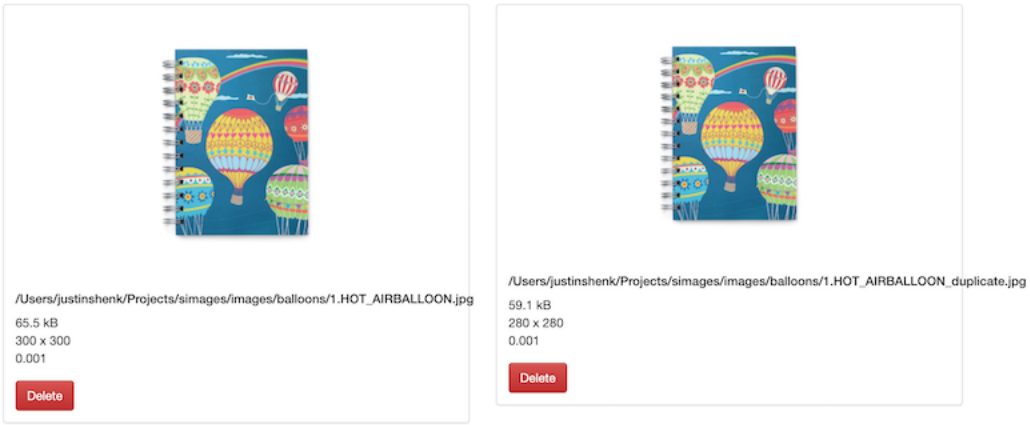
Returns reconstructed image from embedding

Return type image (np.ndarray or torch.Tensor)

1.6 Removing Duplicates

1.6.1 Web Server

Duplicates can be interactively removed with the *web interface*.



Click 'delete' and the image will be removed from the directory.

1.7 Reference

1.7.1 Main methods

The following methods are available via *simages.main*:

A tool to find and remove duplicate pictures (CLI and webserver modified with permission from @philipbl's https://github.com/philipbl/duplicate_images).

Command line:

```
Usage:
    simages add <path> ... [--db=<db_path>] [--parallel=<num_processes>]
    simages remove <path> ... [--db=<db_path>]
    simages clear [--db=<db_path>]
    simages show [--db=<db_path>]
    simages find <path> [--print] [--delete] [--match-time] [--trash=<trash_path>] [--
↪db=<db_path>] [--epochs=<epochs>]
    simages -h | --help
Options:

    -h, --help                Show this screen
    --db=<db_path>            The location of the database or a MongoDB URI.␣
↪(default: ./db)
    --parallel=<num_processes> The number of parallel processes to run to hash the␣
↪image
                               files (default: number of CPUs).
    find:
        --print               Only print duplicate files rather than displaying HTML␣
↪file
        --delete              Move all found duplicate pictures to the trash. This␣
↪option takes priority over --print.
        --match-time          Adds the extra constraint that duplicate images must␣
↪have the
                              same capture times in order to be considered.
        --trash=<trash_path>  Where files will be put when they are deleted (default:␣
↪./Trash)
        --epochs=<epochs>     Epochs for training [default: 2]
```

simages.main.**build_parser**()

simages.main.**parse_arguments**(*args*)

simages.main.**find_duplicates**(*input: Union[str, numpy.ndarray], n: int = 5, num_epochs: int = 2, num_channels: int = 3, show: bool = False, show_train: bool = False, show_path: bool = True, z_dim: int = 8, db=None, **kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]

Find duplicates in dataset. Either np.ndarray or path to image folder must be specified as *input*.

Parameters

- **input** (*str or np.ndarray*) – folder directory or N x C x H x W array
- **n** (*int*) – number of closest pairs to identify
- **num_epochs** (*int*) – how long to train the autoencoder (more is generally better)
- **show** (*bool*) – display the closest pairs
- **show_train** (*bool*) – show output every
- **show_path** (*bool*) – show image paths of duplicates instead of index
- **z_dim** (*int*) – size of compression (more is generally better, but slower)
- **kwargs** (*dict*) – etc, passed to *EmbeddingExtractor*

Returns indices for closest pairs of images distances (np.ndarray): distances of each pair to each other

Return type pairs (np.ndarray)

simages.main.**main**()

Main entry point for *simages-show* via command line.

simages.main.**find_similar**(*db*)

simages.main.**cli**()

1.7.2 Extractor Methods

The following methods are available via *simages.extractor.EmbeddingExtractor*:

class simages.extractor.**EmbeddingExtractor**(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)

Bases: object

Extract embeddings from data with models and allow visualization.

trainloader

Type torch loader

evalloader

Type torch loader

model

Type torch.nn.Module

embeddings

Type np.ndarray

__init__(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)

Inits EmbeddingExtractor with input, either *str* or *np.ndarray*, performs training and validation.

Parameters

- **input** (*np.ndarray or str*) – data
- **num_channels** (*int*) – grayscale = 1, color = 3
- **num_epochs** (*int*) – more is better (generally)
- **batch_size** (*int*) – number of images per batch
- **show** (*bool*) – show closest pairs
- **show_path** (*bool*) – show path of duplicates
- **show_train** (*bool*) – show intermediate training results
- **z_dim** (*int*) – compression size
- **metric** (*str*) – distance metric for `scipy.spatial.distance.cdist()` (eg, euclidean, cosine, hamming, etc.)
- **model** (*torch.nn.Module, optional*) – class implementing same methods as *BasicAutoencoder*
- **db_conn_string** (*str*) – Mongodb connection string
- **kwargs** (*dict*) –

get_image(*index: int*) → torch.Tensor

train()

Train autoencoder to build embeddings of dataset. Final embeddings are created in *eval()*.

eval()

Evaluate reconstruction of embeddings built in *train*.

duplicates(*n: int = 10*, *quantile: float = None*) → Tuple[numpy.ndarray, numpy.ndarray]

Identify *n* closest pairs of images, or quantile (for example, closest 0.05).

Parameters

- **n** (*int*) – number of pairs
- **quantile** (*float*) – quantile of total combination (suggested range: 0.001 - 0.01)

static channels_last(*img: numpy.ndarray*) → numpy.ndarray

Move channels from first to last by swapping axes.

show(*img: Union[torch.Tensor, numpy.ndarray], title: str = '', block: bool = True, y_labels=None, unnormalize=True*)

Plot *img* with *title*.

Parameters

- **img** (*torch.Tensor or np.ndarray*) – Image to plot

- **title** (*str*) – plot title
- **block** (*bool*) – block matplotlib plot until window closed

show_images (*indices: Union[list, int], title=''*)
Plot images (from validation data) at *indices* with *title*

image_paths (*indices*, *short=True*)
Get path to image at *index* of eval/embedding

Parameters

- **Union[int,list]** (*indices*) – indices of embeddings in dataset
- **short** (*bool*) – truncate filepath to 30 charachters

Returns paths to images in image folder

Return type paths (str or list of str)

show_duplicates (*n=5*, *path=False) -> (<class 'numpy.ndarray'>*, *<class 'numpy.ndarray'>*)
Show duplicates from comparison of embeddings. Uses *closely* package to get pairs.

Parameters

- **n** (*int*) – how many closest pairs to identify
- **path** (*bool*) – Plot pairs of images with abbreviated paths

Returns pairs as indices distances (np.ndarray): distances of pairs

Return type pairs (np.ndarray)

unnormalize (*image: torch.Tensor*) → torch.Tensor
Unnormalize an image.

Parameters **image** (torch.Tensor) –

Returns image (torch.Tensor)

decode (*embedding: Optional[numpy.ndarray] = None*, *index: Optional[int] = None*, *show: bool = False*, *astensor: bool = False*) → numpy.ndarray
Decode embeddings at *index* or pass *embedding* directly

Parameters

- **embedding** (*np.ndarray, optional*) – embedding of image
- **index** (*int*) – index (of validation set / embeddings) to decode
- **show** (*bool*) – plot the results
- **astensor** (*bool*) – keep as torch.Tensor

Returns reconstructed image from embedding

Return type image (np.ndarray or torch.Tensor)

1.7.3 Embedding methods

The following methods are available via *simages.embeddings.Embeddings*:

class simages.embeddings.**Embeddings** (*input: Union[numpy.ndarray, str], **kwargs*)
Bases: object

Create embeddings from *input* data by training an autoencoder.

Passes arguments for *EmbeddingExtractor*.

extractor
workhorse for extracting embeddings from dataset

Type *simages.EmbeddingExtractor*

embeddings
embeddings

Type np.ndarray

pairs
n closest pairs

Type np.ndarray

distances
distances between n-closest pairs

Type np.ndarray

__init__ (*input: Union[numpy.ndarray, str], **kwargs*)
Inits Embeddings with data.

array

duplicates (*n: int = 10*)

show_duplicates (*n=5*)
Convenience wrapper for *EmbeddingExtractor.show_duplicates*

images_to_embeddings (*data_dir: str, **kwargs*)

array_to_embeddings (*array: numpy.ndarray, **kwargs*)

1.7.4 Dataset methods

The following classes are available via `simages.dataset`:

A tool to find and remove duplicate pictures (CLI and webserver modified with permission from @philipbl's https://github.com/philipbl/duplicate_images).

Command line:

```
Usage:
    simages add <path> ... [--db=<db_path>] [--parallel=<num_processes>]
    simages remove <path> ... [--db=<db_path>]
    simages clear [--db=<db_path>]
    simages show [--db=<db_path>]
    simages find <path> [--print] [--delete] [--match-time] [--trash=<trash_path>] [--
↪db=<db_path>] [--epochs=<epochs>]
    simages -h | --help
Options:

    -h, --help                Show this screen
    --db=<db_path>            The location of the database or a MongoDB URI.␣
↪(default: ./db)
    --parallel=<num_processes> The number of parallel processes to run to hash the␣
↪image
                               files (default: number of CPUs).
    find:
```

(continues on next page)

(continued from previous page)

```
        --print                 Only print duplicate files rather than displaying HTML␣
↪file
        --delete                Move all found duplicate pictures to the trash. This␣
↪option takes priority over --print.
        --match-time            Adds the extra constraint that duplicate images must␣
↪have the
                                same capture times in order to be considered.
        --trash=<trash_path>    Where files will be put when they are deleted (default:␣
↪./Trash)
        --epochs=<epochs>       Epochs for training [default: 2]
```

simages.main.**build_parser**()

simages.main.**parse_arguments**(*args*)

simages.main.**find_duplicates**(*input: Union[str, numpy.ndarray], n: int = 5, num_epochs: int = 2, num_channels: int = 3, show: bool = False, show_train: bool = False, show_path: bool = True, z_dim: int = 8, db=None, **kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]

Find duplicates in dataset. Either np.ndarray or path to image folder must be specified as *input*.

Parameters

- **input** (*str or np.ndarray*) – folder directory or N x C x H x W array
- **n** (*int*) – number of closest pairs to identify
- **num_epochs** (*int*) – how long to train the autoencoder (more is generally better)
- **show** (*bool*) – display the closest pairs
- **show_train** (*bool*) – show output every
- **show_path** (*bool*) – show image paths of duplicates instead of index
- **z_dim** (*int*) – size of compression (more is generally better, but slower)
- **kwargs** (*dict*) – etc, passed to *EmbeddingExtractor*

Returns indices for closest pairs of images distances (np.ndarray): distances of each pair to each other

Return type pairs (np.ndarray)

simages.main.**main**()

Main entry point for *simages-show* via command line.

simages.main.**find_similar**(*db*)

simages.main.**cli**()

1.7.5 Module contents

Find similar images in a dataset <https://github.com/justinshenk/simages>

class simages.**Embeddings**(*input: Union[numpy.ndarray, str], **kwargs*)

Bases: object

Create embeddings from *input* data by training an autoencoder.

Passes arguments for *EmbeddingExtractor*.

extractor
workhorse for extracting embeddings from dataset

Type *simages.EmbeddingExtractor*

embeddings
embeddings

Type np.ndarray

pairs
n closest pairs

Type np.ndarray

distances
distances between n-closest pairs

Type np.ndarray

__init__(*input: Union[numpy.ndarray, str], **kwargs*)
Inits Embeddings with data.

array

duplicates(*n: int = 10*)

show_duplicates(*n=5*)
Convenience wrapper for *EmbeddingExtractor.show_duplicates*

images_to_embeddings(*data_dir: str*, ***kwargs*)

array_to_embeddings(*array: numpy.ndarray*, ***kwargs*)

class simages.**EmbeddingExtractor**(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)
Bases: object

Extract embeddings from data with models and allow visualization.

trainloader

Type torch loader

evalloader

Type torch loader

model

Type torch.nn.Module

embeddings

Type np.ndarray

__init__(*input: Union[str, numpy.ndarray], num_channels: int = 3, num_epochs: int = 2, batch_size: int = 32, show: bool = False, show_path: bool = False, show_train: bool = False, z_dim: int = 8, metric: str = 'cosine', model: Optional[torch.nn.modules.module.Module] = None, db: Optional = None, **kwargs*)
Inits EmbeddingExtractor with input, either *str* or *np.ndarray*, performs training and validation.

Parameters

- **input** (*np.ndarray or str*) – data
- **num_channels** (*int*) – grayscale = 1, color = 3
- **num_epochs** (*int*) – more is better (generally)
- **batch_size** (*int*) – number of images per batch
- **show** (*bool*) – show closest pairs
- **show_path** (*bool*) – show path of duplicates
- **show_train** (*bool*) – show intermediate training results
- **z_dim** (*int*) – compression size
- **metric** (*str*) – distance metric for `scipy.spatial.distance.cdist()` (eg, euclidean, cosine, hamming, etc.)
- **model** (*torch.nn.Module, optional*) – class implementing same methods as *BasicAutoencoder*
- **db_conn_string** (*str*) – Mongodb connection string
- **kwargs** (*dict*) –

get_image(*index: int*) → torch.Tensor

train()
Train autoencoder to build embeddings of dataset. Final embeddings are created in *eval()*.

eval()
Evaluate reconstruction of embeddings built in *train*.

duplicates(*n: int = 10*, *quantile: float = None*) → Tuple[numpy.ndarray, numpy.ndarray]
Identify *n* closest pairs of images, or quantile (for example, closest 0.05).

Parameters

- **n** (*int*) – number of pairs
- **quantile** (*float*) – quantile of total combination (suggested range: 0.001 - 0.01)

static channels_last(*img: numpy.ndarray*) → numpy.ndarray
Move channels from first to last by swapping axes.

show(*img: Union[torch.Tensor, numpy.ndarray], title: str = '', block: bool = True, y_labels=None, unnormalize=True*)
Plot *img* with *title*.

Parameters

- **img** (*torch.Tensor or np.ndarray*) – Image to plot
- **title** (*str*) – plot title
- **block** (*bool*) – block matplotlib plot until window closed

show_images(*indices: Union[list, int], title=''*)
Plot images (from validation data) at *indices* with *title*

image_paths(*indices*, *short=True*)
Get path to image at *index* of eval/embedding

Parameters

- **Union[int,list]** (*indices*) – indices of embeddings in dataset
- **short** (*bool*) – truncate filepath to 30 charachters

Returns paths to images in image folder

Return type paths (str or list of str)

show_duplicates(*n=5, path=False) -> (<class 'numpy.ndarray'>, <class 'numpy.ndarray'>*)
Show duplicates from comparison of embeddings. Uses *closely* package to get pairs.

Parameters

- **n** (*int*) – how many closest pairs to identify
- **path** (*bool*) – Plot pairs of images with abbreviated paths

Returns pairs as indices distances (np.ndarray): distances of pairs

Return type pairs (np.ndarray)

unnormalize(*image: torch.Tensor*) → torch.Tensor
Unnormalize an image.

Parameters **image** (torch.Tensor) –

Returns image (torch.Tensor)

decode(*embedding: Optional[numpy.ndarray] = None, index: Optional[int] = None, show: bool = False, astensor: bool = False*) → numpy.ndarray
Decode embeddings at *index* or pass *embedding* directly

Parameters

- **embedding** (*np.ndarray, optional*) – embedding of image
- **index** (*int*) – index (of validation set / embeddings) to decode
- **show** (*bool*) – plot the results
- **astensor** (*bool*) – keep as torch.Tensor

Returns reconstructed image from embedding

Return type image (np.ndarray or torch.Tensor)

simages.**find_duplicates**(*input: Union[str, numpy.ndarray], n: int = 5, num_epochs: int = 2, num_channels: int = 3, show: bool = False, show_train: bool = False, show_path: bool = True, z_dim: int = 8, db=None, **kwargs*) → Tuple[numpy.ndarray, numpy.ndarray]
Find duplicates in dataset. Either np.ndarray or path to image folder must be specified as *input*.

Parameters

- **input** (*str or np.ndarray*) – folder directory or N x C x H x W array
- **n** (*int*) – number of closest pairs to identify
- **num_epochs** (*int*) – how long to train the autoencoder (more is generally better)
- **show** (*bool*) – display the closest pairs
- **show_train** (*bool*) – show output every
- **show_path** (*bool*) – show image paths of duplicates instead of index
- **z_dim** (*int*) – size of compression (more is generally better, but slower)
- **kwargs** (*dict*) – etc, passed to *EmbeddingExtractor*

Returns indices for closest pairs of images distances (np.ndarray): distances of each pair to each other

Return type pairs (np.ndarray)

class simages.**PILDataset**(*pil_list: list*, *transform: Optional[Callable] = None*)
Bases: torch.utils.data.dataset.Dataset

PIL dataset.

__init__(*pil_list: list*, *transform: Optional[Callable] = None*)

Parameters

- **pil_list** (*list of PIL images*) –
- **transform** (*callable, optional*) – Optional transform to be applied on a sample.

class simages.**ImageFolder**(*root: str*, *loader: Callable = <function default_loader>*, *extensions: Optional[list] = None*, *transform: Optional[list] = None*, *is_valid_file: Optional[Callable] = None*)
Bases: torchvision.datasets.vision.VisionDataset

A generic data loader where the samples are arranged in this way:

```
root/xxx.ext
root/xxy.ext
root/xxz.ext
```

Parameters

- **root** (*string*) – Root directory path.
- **loader** (*callable*) – A function to load a sample given its path.
- **extensions** (*tuple[string]*) – A list of allowed extensions. both extensions and is_valid_file should not be passed.
- **transform** (*callable, optional*) – A function/transform that takes in a sample and returns a transformed version. E.g, transforms.RandomCrop for images.
- **is_valid_file** – A function that takes path of an Image file and check if the file is a valid_file (used to check of corrupt files) both extensions and is_valid_file should not be passed.

__getitem__(*index: int*)

Parameters **index** (*int*) – Index

Returns (sample, target) where target is class_index of the target class.

Return type tuple

class simages.**BasicAutoencoder**(*num_channels: int = 1*, *z_dim: int = 8*, *hw=48*)
Bases: torch.nn.modules.module.Module

__init__(*num_channels: int = 1*, *z_dim: int = 8*, *hw=48*)
Basic autoencoder - default for simages.

Parameters

- **num_channels** (*int*) – grayscale = 1, color = 3
- **z_dim** (*int*) – number of embedding units to compress image to
- **hw** (*int*) – height and width for input/output image

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

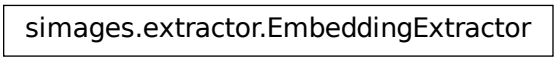
decode(*x*)

simages.**linkageplot**(*embeddings: numpy.ndarray*, *ordered=True*)
Plot linkage between embeddings in hierarchical clustering of the distance matrix

Parameters

- **embeddings** (*np.ndarray*) – embeddings of images in dataset
- **ordered** (*bool*) – order distance matrix before plotting

Inheritance diagram:



1.8 Contributing to simages

(Contribution guidelines largely copied from geopandas)

1.8.1 Overview

Contributions to simages are very welcome. They are likely to be accepted more quickly if they follow these guidelines.

At this stage of simages development, the priorities are to define a simple, usable, and stable API and to have clean, maintainable, readable code. Performance matters, but not at the expense of those goals.

In general, simages follows the conventions of the pandas project where applicable.

In particular, when submitting a pull request:

- All existing tests should pass. Please make sure that the test suite passes, both locally and on Travis CI. Status on Travis will be visible on a pull request. If you want to enable Travis CI on your own fork, please read the pandas guidelines link above or the getting started docs.
- New functionality should include tests. Please write reasonable tests for your code and make sure that they pass on your pull request.
- Classes, methods, functions, etc. should have docstrings. The first line of a docstring should be a standalone summary. Parameters and return values should be ducumented explicitly.
- simages supports python 3 (3.6+). Use modern python idioms when possible.

- Follow PEP 8 when possible.
- Imports should be grouped with standard library imports first, 3rd-party libraries next, and simages imports third. Within each grouping, imports should be alphabetized. Always use absolute imports when possible, and explicit relative imports for local imports when necessary in tests.

Seven Steps for Contributing

There are seven basic steps to contributing to *simages*:

1) Fork the *simages* git repository
2) Create a development environment
3) Install *simages* dependencies
4) Make a `development` build of *simages*
5) Make changes to code and add tests
6) Update the documentation
7) Submit a Pull Request

Each of these 7 steps is detailed below.

1.8.2 1) Forking the *simages* repository using Git

To the new user, working with Git is one of the more daunting aspects of contributing to *simages**. It can very quickly become overwhelming, but sticking to the guidelines below will help keep the process straightforward and mostly trouble free. As always, if you are having difficulties please feel free to ask for help.

The code is hosted on GitHub. To contribute you will need to sign up for a free GitHub account. We use Git for version control to allow many people to work together on the project.

Some great resources for learning Git:

- Software Carpentry's Git Tutorial
- Atlassian
- the GitHub help pages.
- Matthew Brett's Pydagogue.

Getting started with Git

GitHub has instructions for installing git, setting up your SSH key, and configuring git. All these steps need to be completed before you can work seamlessly between your local repository and GitHub.

Forking

You will need your own fork to work on the code. Go to the simages project page and hit the `Fork` button. You will want to clone your fork to your machine:

```
git clone git@github.com:your-user-name/simages.git simages-yourname
cd simages-yourname
git remote add upstream git://github.com/justinshenk/simages.git
```

This creates the directory *simages-yourname* and connects your repository to the upstream (main project) *simages* repository.

The testing suite will run automatically on Travis-CI once your pull request is submitted. However, if you wish to run the test suite on a branch prior to submitting the pull request, then Travis-CI needs to be hooked up to your GitHub repository. Instructions for doing so are here.

Creating a branch

You want your master branch to reflect only production-ready code, so create a feature branch for making your changes. For example:

```
git branch shiny-new-feature
git checkout shiny-new-feature
```

The above can be simplified to:

```
git checkout -b shiny-new-feature
```

This changes your working directory to the shiny-new-feature branch. Keep any changes in this branch specific to one bug or feature so it is clear what the branch brings to *simages*. You can have many shiny-new-features and switch in between them using the git checkout command.

To update this branch, you need to retrieve the changes from the master branch:

```
git fetch upstream
git rebase upstream/master
```

This will replay your commits on top of the latest simages git master. If this leads to merge conflicts, you must resolve these before submitting your pull request. If you have uncommitted changes, you will need to `stash` them prior to updating. This will effectively store your changes and they can be reapplied after updating.

1.8.3 2) Creating a development environment

A development environment is a virtual space where you can keep an independent installation of *simages*. This makes it easy to keep both a stable version of python in one place you use for work, and a development version (which you may break while playing with code) in another.

An easy way to create a *simages* development environment is as follows:

- Install either Anaconda or miniconda
- Make sure that you have *cloned the repository*
- `cd` to the *simages** source directory

Tell conda to create a new environment, named `simages_dev`, or any other name you would like for this environment, by running:

```
conda create -n simages_dev
```

For a python 3 environment:

```
conda create -n simages_dev python=3.6
```

This will create the new environment, and not touch any of your existing environments, nor any existing python installation.

To work in this environment, Windows users should `activate` it as follows:

```
activate simages_dev
```

Mac OSX and Linux users should use:

```
source activate simages_dev
```

You will then see a confirmation message to indicate you are in the new development environment.

To view your environments:

```
conda info -e
```

To return to you home root environment:

```
deactivate
```

See the full conda docs here.

At this point you can easily do a *development* install, as detailed in the next sections.

1.8.4 3) Installing Dependencies

To run *simages* in an development environment, you must first install *simages*'s dependencies. We suggest doing so using the following commands (executed after your development environment has been activated):

```
pip install requirements-dev.txt
```

This should install all necessary dependencies.

Next activate pre-commit hooks by running:

```
pre-commit install
```

1.8.5 4) Making a development build

Once dependencies are in place, make an in-place build by navigating to the git clone of the *simages* repository and running:

```
python setup.py develop
```

1.8.6 5) Making changes and writing tests

simages is serious about testing and strongly encourages contributors to embrace test-driven development (TDD). This development process "relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test." So, before actually writing any code, you should write your tests. Often the test can be taken from the original GitHub issue. However, it is always worth considering additional use cases and writing corresponding tests.

Adding tests is one of the most common requests after code is pushed to *simages*. Therefore, it is worth getting in the habit of writing tests ahead of time so this is never an issue.

simages uses the pytest testing system and the convenient extensions in numpy.testing.

Writing tests

All tests should go into the `tests` directory. This folder contains many current examples of tests, and we suggest looking to these for inspiration.

Running the test suite

The tests can then be run directly inside your Git clone (without having to install *simages*) by typing:

```
pytest
```

1.8.7 6) Updating the Documentation

simages documentation resides in the *doc* folder. Changes to the docs are make by modifying the appropriate file in the *source* folder within *doc*. *simages* docs us reStructuredText syntax, which is explained here and the docstrings follow the Numpy Docstring standard.

Once you have made your changes, you can build the docs by navigating to the *doc* folder and typing:

```
make html
```

The resulting html pages will be located in *doc/build/html*.

1.8.8 7) Submitting a Pull Request

Once you've made changes and pushed them to your forked repository, you then submit a pull request to have them integrated into the *simages* code base.

You can find a pull request (or PR) tutorial in the GitHub's Help Docs.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

s

Index

Symbols

A

B

C

D

E

F