numpy-datasets

Nov 28, 2020

Contents

Installation Guide : *Installation*

Developer Guide : *Development*

CHAPTER 1

Tutorials

CHAPTER 2

Gallery

- *Images Datasets*
- *Time-Series Datasets*

2.1 Installation

SymJAX has a couple of prerequisites that need to be installed first.

2.1.1 CPU only installation

Installation of SymJAX and all its dependencies (including Jax). For CPU only support is done simply as follows

```
$ pip install --upgrade jaxlib
$ pip install --upgrade jax
$ pip install --upgrade symjax
```

2.1.2 GPU installation

For the GPU support, the Jax installation needs to be done first and based on the local cuda settings following Jax Installation. In short, the steps involve

1. Installation of GPU drivers/libraries/compilers (`cuda`, `cudnn`, `nvcc`).
2. Install `jax` following Jax Installation.
3. Install SymJAX with

```
$ pip install --upgrade symjax
```

2.1.3 Manual (local/bleeding-edge) installation of SymJAX

In place of the base installation of SymJAX from the latest official release from PyPi, one can install the latest version of SymJAX from the github repository as follows

1. Clone this repository with

```
$ git clone https://github.com/RandallBalestriero/SymJAX
```

2. Install.

```
$ cd SymJAX
$ pip install .
```

Note that whenever changes are made to the SymJAX github repository, one can pull those changes bu running

```
$ git pull
```

from within the cloned repository. However the changes won't impact the installed version unless the install was done with

```
$ pip install -e .
```

2.2 Gallery

Contents

2.2.1 Basic examples

Introductory examples.

2.2.2 Images Datasets

MNIST Dataset

This example shows how to download/load/import MNIST

Out:

```
        ... mnist.pkl.gz already exists
Loading mnist
Dataset mnist loaded in 1.21s.
```

```
import numpy_datasets as nds
import matplotlib.pyplot as plt

mnist = nds.images.mnist.load()

plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, 1 + i)
    plt.imshow(mnist["train_set/images"][i, 0], aspect="auto", cmap="Greys")
    plt.xticks([])
    plt.yticks([])
    plt.title(str(mnist["train_set/labels"][i]))

plt.tight_layout()
```

Total running time of the script: (0 minutes 1.913 seconds)

CIFAR10 Dataset

This example shows how to download/load/import CIFAR10

Out:

```
        ... cifar-10-python.tar.gz already exists

Loading cifar10:   0%|          | 0/5 [00:00<?, ?it/s]
Loading cifar10:  20%|##        | 1/5 [00:03<00:15,  3.91s/it]
Loading cifar10:  40%|####      | 2/5 [00:05<00:09,  3.14s/it]
Loading cifar10:  60%|######    | 3/5 [00:06<00:04,  2.49s/it]
Loading cifar10:  80%|########  | 4/5 [00:06<00:01,  1.85s/it]
Loading cifar10: 100%|##########| 5/5 [00:07<00:00,  1.68s/it]
Loading cifar10: 100%|##########| 5/5 [00:07<00:00,  1.57s/it]
Dataset cifar10 loaded in8.69s.
```

```
import numpy_datasets as nds
import matplotlib.pyplot as plt

cifar10 = nds.images.cifar10.load()

plt.figure(figsize=(10, 4))
for i in range(10):

    plt.subplot(2, 5, 1 + i)

    image = cifar10["train_set/images"][i]
    label = cifar10["train_set/labels"][i]
```

(continues on next page)

(continued from previous page)

```
    plt.imshow(image.transpose((1, 2, 0)) / image.max(), aspect="auto", cmap="Greys")
    plt.xticks([])
    plt.yticks([])
    plt.title("{}:{}".format(label, cifar10["label_to_name"][label]))

plt.tight_layout()
```

Total running time of the script: (0 minutes 9.329 seconds)

2.2.3 Time-Series Datasets

Speech picidae Dataset

This example shows how to download/load/import speech picidae

Out:

```
        ... PicidaeDataset.zip already exists

  0%|          | 0/3369 [00:00<?, ?it/s]
  8%|7         | 263/3369 [00:00<00:01, 2625.26it/s]
 11%|#1        | 371/3369 [00:00<00:01, 1823.98it/s]
 17%|#6        | 561/3369 [00:00<00:01, 1838.41it/s]/home/vrael/numpy-datasets/numpy_
↪datasets/timeseries/picidae.py:96: WavFileWarning: Chunk (non-data) not understood,␣
↪skipping it.
  wavs.append(wav_read(byt)[1].astype("float32"))

 20%|##        | 679/3369 [00:00<00:02, 1170.92it/s]
 24%|##3       | 804/3369 [00:00<00:02, 1192.80it/s]
 29%|##9       | 987/3369 [00:00<00:01, 1328.36it/s]
 33%|###3      | 1117/3369 [00:00<00:02, 910.67it/s]
 36%|###6      | 1224/3369 [00:01<00:02, 832.09it/s]
 39%|###9      | 1322/3369 [00:01<00:02, 867.97it/s]
 42%|####2     | 1418/3369 [00:01<00:02, 820.91it/s]
 46%|####6     | 1552/3369 [00:01<00:01, 927.36it/s]
 49%|####9     | 1655/3369 [00:01<00:02, 833.44it/s]
 54%|#####4    | 1823/3369 [00:01<00:01, 981.52it/s]
 58%|#####7    | 1938/3369 [00:01<00:01, 899.14it/s]
 61%|######    | 2041/3369 [00:01<00:01, 852.96it/s]
 64%|######3   | 2150/3369 [00:02<00:01, 911.05it/s]
 67%|######6   | 2249/3369 [00:02<00:01, 668.36it/s]
 69%|######9   | 2331/3369 [00:02<00:01, 636.67it/s]
 85%|########5 | 2867/3369 [00:02<00:00, 864.82it/s]
 91%|#########1| 3078/3369 [00:03<00:00, 412.14it/s]
 96%|#########5| 3231/3369 [00:04<00:00, 278.95it/s]
 99%|#########9| 3343/3369 [00:05<00:00, 256.85it/s]
100%|##########| 3369/3369 [00:05<00:00, 642.22it/s]
Dataset picidae loaded in 5.29s.
```

```
import numpy_datasets as nds
import matplotlib.pyplot as plt

picidae = nds.timeseries.picidae.load()

plt.figure(figsize=(10, 4))
for i in range(10):

    plt.subplot(2, 5, 1 + i)
    plt.plot(picidae["wavs"][i])
    plt.title(str(picidae["labels"][i]))

plt.tight_layout()
```

Total running time of the script: (0 minutes 6.016 seconds)

2.3 Development

The numpy-datasets project was started by Randall Balestriero in early 2020. As an open-source project, we highly welcome contributions (current contributors) !

2.3.1 Philosophy

numpy-datasets started from the need to combine the best functionalities of Theano, Tensorflow (v1) and Lasagne. While we propose various deep learning oriented methods, numpy-datasets shall remain as general as possible in its core, methods should be grouped as much as possible into specialized submodules, and a complete documentation should be provided, preferably along with a working example located in the Gallery.

2.3.2 How to contribute

If you are willing to help, we recommend to follow the following steps before requesting a pull request. Recall that

1. **Coding conventions**: we used the PEP8 style guide for Python Code and the black formatting
2. **Docstrings**: we use the numpydoc docstring guide for documenting the functions directly from the docstrings and automatically generating the documentation with sphinx. Please provide codes with up-to-date docstrings.
3. **Continuous Integration**: to ensure that all the numpy-datasets functionalities are tested after each modifition run `pytest` from the main numpy-datasets directory. All tests should pass before considering a change to be successful. If new functionalities are added, it is highly preferable to also add a simple test in the `tests/` directory to ensure that results are as expected. A Github action will automatically test the code at each `push` (see *Test the code*).

Build/Test the doc

To rebuild the documentation, install several packages:

```
pip install -r docs/requirements.txt
```

to generate the documentation, you can do in the `docs` directory and run:

```
make html
```

You can then see the generated documentation in `docs/_build/html/index.html`.

If examples/code-blocks are added to the documension, it has to be tested. To do so, add the specific module/function in the `tests/doc.py` and run:

```
>>> python tests/doc.py
```

if all tests pass, then the changes are ready to be put in a PR. Once the documentation has been changed and all tests pass, the change is ready for review and should be put in a PR.

Every time changes are pushed to Github `master` branch the numpy-datasets documentations (at sym-jax.readthedocs.io) is rebuilt based on the `.readthedocs.yml` and the `docs/conf.py` configuration files. For each automated documentation build you can see the documentation build logs.

Test the code

To run all the numpy-datasets tests, we recommend using `pytest` or `pytest-xdist`. First, install `pytest-xdist` and `pytest-benchmark` by running `pip install pytest-xdist pytest-benchmark`. Then, from the repository root directory run:

```
pytest
```

If all tests pass successfully, the code is ready for a PR.

Pull requests

Once all the tests pass and the documentation is appropriate, commit your changes to a new branch, push that branch to your fork and send us a Pull Request via GitHub's web interface (guide), the PR should include a brief description.

2.4 numpy_datasets.images

numpy_datasets.images.mnist.load([path])	The MNIST database of handwritten digits, available from this page has a training set of 60,000 examples, and a test set of 10,000 examples.
numpy_datasets.images.arabic_characters.load([path])	Arabic Handwritten Characters Dataset
numpy_datasets.images.arabic_digits.load([path])	Arabic Handwritten Digits Dataset
numpy_datasets.images.kmnist.load([dataset, ...])	japanese character (image) classification
numpy_datasets.images.emnist.load([option, path])	Grayscale digit/letter classification.
numpy_datasets.images.fashionmnist.load([path])	Grayscale image classification
numpy_datasets.images.face_pointing.load([path])	head angle classification The head pose database consists of 15 sets of images.

Continued on next page

Table 1 – continued from previous page

numpy_datasets.images.rock_paper_scissors.load([path])	The MNIST database of handwritten digits, available from this page has a training set of 60,000 examples, and a test set of 10,000 examples.
numpy_datasets.images.dsprites.load([path])	greyscale image classification and disentanglement
numpy_datasets.images.svhn.load([path])	Street number classification.
numpy_datasets.images.cifar10.load([path])	Image classification.
numpy_datasets.images.cifar100.load([path])	Image classification.
numpy_datasets.images.celeb.load([path])	face images with attributes CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations.
numpy_datasets.images.ibeans.load([path])	Plant images classification.
numpy_datasets.images.stl10.load([path])	Image classification with extra unlabeled images.
numpy_datasets.images.tinyimagenet.load([path])	Tiny Imagenet has 200 classes.

2.4.1 Detailed description

numpy_datasets.images.mnist.**load**(*path=None*)

The MNIST database of handwritten digits, available from this page has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.arabic_characters.**load**(*path=None*)

Arabic Handwritten Characters Dataset

Astract Handwritten Arabic character recognition systems face several challenges, including the unlimited variation in human handwriting and large public databases. In this work, we model a deep learning architecture that can be effectively apply to recognizing Arabic handwritten characters. A Convolutional Neural Network (CNN) is a special type of feed-forward multilayer trained in supervised mode. The CNN trained and tested our database that contain 16800 of handwritten Arabic characters. In this paper, the optimization methods implemented to increase the performance of CNN. Common machine learning methods usually apply a combination

of feature extractor and trainable classifier. The use of CNN leads to significant improvements across different machine-learning classification algorithms. Our proposed CNN is giving an average 5.1% misclassification error on testing data.

Context The motivation of this study is to use cross knowledge learned from multiple works to enhancement the performance of Arabic handwritten character recognition. In recent years, Arabic handwritten characters recognition with different handwriting styles as well, making it important to find and work on a new and advanced solution for handwriting recognition. A deep learning systems needs a huge number of data (images) to be able to make a good decisions.

Content The data-set is composed of 16,800 characters written by 60 participants, the age range is between 19 to 40 years, and 90% of participants are right-hand. Each participant wrote each character (from 'alef' to 'yeh') ten times on two forms as shown in Fig. 7(a) & 7(b). The forms were scanned at the resolution of 300 dpi. Each block is segmented automatically using Matlab 2016a to determining the coordinates for each block. The database is partitioned into two sets: a training set (13,440 characters to 480 images per class) and a test set (3,360 characters to 120 images per class). Writers of training set and test set are exclusive. Ordering of including writers to test set are randomized to make sure that writers of test set are not from a single institution (to ensure variability of the test set).

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.arabic_digits.**load**(*path=None*)

Arabic Handwritten Digits Dataset

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.kmnist.**load**(*dataset='kmnist'*, *path=None*)

japanese character (image) classification

Kuzushiji-MNIST is a drop-in replacement for the MNIST dataset (28x28 grayscale, 70,000 images), provided in the original MNIST format as well as a NumPy format. Since MNIST restricts us to 10 classes, we chose one character to represent each of the 10 rows of Hiragana when creating Kuzushiji-MNIST.

Kuzushiji-49, as the name suggests, has 49 classes (28x28 grayscale, 270,912 images), is a much larger, but imbalanced dataset containing 48 Hiragana characters and one Hiragana iteration mark.

Kuzushiji-MNIST

Kuzushiji-MNIST contains 70,000 28x28 grayscale images spanning 10 classes (one from each column of hiragana), and is perfectly balanced like the original MNIST dataset (6k/1k train/test for each class). File Examples Download (MNIST format) Download (NumPy format) Training images 60,000 train-images-idx3-ubyte.gz (18MB) kmnist-train-imgs.npz (18MB) Training labels 60,000 train-labels-idx1-ubyte.gz (30KB) kmnist-train-labels.npz (30KB) Testing images 10,000 t10k-images-idx3-ubyte.gz (3MB) kmnist-test-imgs.npz (3MB) Testing labels 10,000 t10k-labels-idx1-ubyte.gz (5KB) kmnist-test-labels.npz (5KB)

Mapping from class indices to characters: kmnist_classmap.csv (1KB)

We recommend using standard top-1 accuracy on the test set for evaluating on Kuzushiji-MNIST. Which format do I download?

If you're looking for a drop-in replacement for the MNIST or Fashion-MNIST dataset (for tools that currently work with these datasets), download the data in MNIST format.

Otherwise, it's recommended to download in NumPy format, which can be loaded into an array as easy as: arr = np.load(filename)['arr_0']. Kuzushiji-49

Kuzushiji-49 contains 270,912 images spanning 49 classes, and is an extension of the Kuzushiji-MNIST dataset. File Examples Download (NumPy format) Training images 232,365 k49-train-imgs.npz (63MB) Training labels 232,365 k49-train-labels.npz (200KB) Testing images 38,547 k49-test-imgs.npz (11MB) Testing labels 38,547 k49-test-labels.npz (50KB)

Mapping from class indices to characters: k49_classmap.csv (1KB)

We recommend using balanced accuracy on the test set for evaluating on Kuzushiji-49. We use the following implementation of balanced accuracy:

License

Both the dataset itself and the contents of this repo are licensed under a permissive CC BY-SA 4.0 license, except where specified within some benchmark scripts. CC BY-SA 4.0 license requires attribution, and we would suggest to use the following attribution to the KMNIST dataset.

"KMNIST Dataset" (created by CODH), adapted from "Kuzushiji Dataset" (created by NIJL and others), doi:10.20676/00000341

Parameters

- **dataset** (*str (optional)*) – *"kmnist"* or *"k49mnist"*
- **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.emnist.**load**(*option='byclass'*, *path=None*)

Grayscale digit/letter classification.

The EMNIST Dataset

Authors:

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik

The MARCS Institute for Brain, Behaviour and Development Western Sydney University Penrith, Australia 2751

Email: g.cohen@westernsydney.edu.au

What is it?

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 (https://www.nist.gov/srd/nist-special-database-19) and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset (http://yann.lecun.com/exdb/mnist/). Further information on the dataset contents and conversion process can be found in the paper available at https://arxiv.org/abs/1702.05373v1.

Formats:

The dataset is provided in two file formats. Both versions of the dataset contain identical information, and are provided entirely for the sake of convenience. The first dataset is provided in a Matlab format that is accessible through both Matlab and Python (using the scipy.io.loadmat function). The second version of the dataset is provided in the same binary format as the original MNIST dataset as outlined in http://yann.lecun.com/exdb/mnist/

Dataset Summary:

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

EMNIST ByClass:EMNIST814,255 characters. 62 unbalanced classes EMNIST ByMerge: 814,255 characters. 47 unbalanced classes EMNIST Balanced:Balanced131,600 characters. 47 balanced classes. EMNIST Letters:EMNIST145,600 characters. 26 balanced classes. EMNIST Digits:EMNIST280,000 characters. 10 balanced classes. EMNIST MNIST:EMNIST 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset.

Please refer to the EMNIST paper (available at https://arxiv.org/abs/1702.05373v1) for further details of the dataset structure.

How to cite:

Please cite the following paper when using or referencing the dataset:

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

`numpy_datasets.images.fashionmnist.`**`load`**(*path=None*)

Grayscale image classification

Zalando ‘s article image classification. Fashion-MNIST is a dataset of Zalando ‘s article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

`numpy_datasets.images.face_pointing.`**`load`**(*path=None*)

head angle classification The head pose database consists of 15 sets of images. Each set contains of 2 series of 93 images of the same person at different poses. There are 15 people in the database, wearing glasses or not and having various skin color. The pose, or head orientation is determined by 2 angles (h,v), which varies from -90 degrees to +90 degrees. Here is a sample of a serie :

PersonID = {01, . . . , 15}: stands for the number of the person.

Serie = {1, 2} stands for the number of the serie.

Number = {00, 01, . . . , 92} the number of the file in the directory.

VerticalAngle = {-90, -60, -30, -15, 0, +15, +30, +60, +90}

HorizontalAngle = {-90, -75, -60, -45, -30, -15, 0, +15, +30, +45, +60, +75, +90}

All images have been taken using the FAME Platform of the PRIMA Team in INRIA Rhone-Alpes. To obtain different poses, we have put markers in the whole room. Each marker corresponds to a pose (h,v). Post-it are used as markers. The whole set of post-it covers a half-sphere in front of the person.

In order to obtain the face in the center of the image, the person is asked to adjust the chair to see the device in front of him. After this initialization phase, we ask the person to stare successively at 93 post-it notes, without moving his eyes. This second phase just takes a few minutes.

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.rock_paper_scissors.**load**(*path=None*)

The MNIST database of handwritten digits, available from this page has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.dsprites.**load**(*path=None*)

greyscale image classification and disentanglement

This dataset consists of 737,280 images of 2D shapes, procedurally generated from 5 ground truth independent latent factors, controlling the shape, scale, rotation and position of a sprite. This data can be used to assess the disentanglement properties of unsupervised learning methods.

dSprites is a dataset of 2D shapes procedurally generated from 6 ground truth independent latent factors. These factors are color, shape, scale, rotation, x and y positions of a sprite.

All possible combinations of these latents are present exactly once, generating N = 737280 total images.

https://github.com/deepmind/dsprites-dataset

path: str (optional) default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

images: array

latent: array

classes: array

numpy_datasets.images.svhn.**load**(*path=None*)

Street number classification.

The SVHN dataset is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Parameters **path** (*str (optional)*) – default $DATASET_PATH, the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.cifar10.**load**(*path=None*)

Image classification. The **'CIFAR-10 < https: // www.cs.toronto.edu/~kriz/cifar.html >'_** dataset was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.cifar100.**load**(*path=None*)

Image classification.

The ‘CIFAR-100 < https: // www.cs.toronto.edu/~kriz/cifar.html >‘_ dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a “fine” label(the class to which it belongs) and a “coarse” label(the superclass to which it belongs).

numpy_datasets.images.celeb.**load**(*path=None*)

face images with attributes CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset

with more than 200K celebrity images, each with 40 attribute annotations. The

images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations, including - 10,177 number of identities, - 202,599 number of face images, and - 5 landmark locations, 40 binary attributes annotations per image. The dataset can be employed as the training and test sets for the following computer vision tasks: face attribute recognition, face detection, and landmark

(or facial part) localization.

Note: CelebA dataset may contain potential bias. The fairness indicators *https://github.com/tensorflow/fairness-indicators/blob/master/fairness_indicators/documentation/examples/Fairness_Indicators_TFCO_CelebA_Case_Study.ipynb* goes into detail about several considerations to keep in mind while using the CelebA dataset.

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.ibeans.**load**(*path=None*)

Plant images classification.

This dataset is of leaf images taken in the field in different districts in Uganda by the Makerere AI lab in collaboration with the National Crops Resources Research Institute (NaCRRI), the national body in charge of research in agriculture in Uganda.

The goal is to build a robust machine learning model that is able to distinguish between diseases in the Bean plants. Beans are an important cereal food crop for Africa grown by many small-holder farmers - they are a significant source of proteins for school-age going children in East Africa.

The data is of leaf images representing 3 classes: the healthy class of images, and two disease classes including Angular Leaf Spot and Bean Rust diseases. The model should be able to distinguish between these 3 classes with high accuracy. The end goal is to build a robust, model that can be deployed on a mobile device and used in the field by a farmer.

The data includes leaf images taken in the field. The figure above depicts examples of the types of images per class. Images were taken from the field/garden a basic smartphone.

The images were then annotated by experts from NaCRRI who determined for each image which disease was manifested. The experts were part of the data collection team and images were annotated directly during the data collection process in the field.

Class Examples Healthy class 428 Angular Leaf Spot 432 Bean Rust 436 Total: 1,296

Data Released 20-January-2020 License MIT Credits Makerere AI Lab

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.images.stl10.**load**(*path=None*)

Image classification with extra unlabeled images.

The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior. We also expect that the higher resolution of this dataset (96x96) will make it a challenging benchmark for developing more scalable unsupervised learning methods.

Parameters **path** (*str (optional)*) – the path to look for the data and where it will be downloaded if not present

Returns

- **train_images** (*array*) – the training images
- **train_labels** (*array*) – the training labels
- **test_images** (*array*) – the test images
- **test_labels** (*array*) – the test labels
- **extra_images** (*array*) – the unlabeled additional images

numpy_datasets.images.tinyimagenet.**load**(*path=None*)

Tiny Imagenet has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. We have released the training and validation sets with images and annotations. We provide both class labels an bounding boxes as annotations; however, you are asked only to predict the class label of each image without localizing the objects. The test set is released without labels. You can download the whole tiny ImageNet dataset here.

2.5 numpy_datasets.timeseries

numpy_datasets.timeseries.audiomnist.load([path])	digit recognition
numpy_datasets.timeseries.univariate_timeseries.load([path])	**param path** default ($DATASET_PATH), the path to look for the data and

Continued on next page

Table 2 – continued from previous page

numpy_datasets.timeseries.dcase_2019_task4.load([path])	synthetic data for polyphonic event detection
numpy_datasets.timeseries.groove_MIDI.load([path])	The Groove MIDI Dataset (GMD) is composed of 13.6 hours of aligned MIDI and (synthesized) audio of human-performed, tempo-aligned expressive drumming.
numpy_datasets.timeseries.speech_commands.load([path])	
numpy_datasets.timeseries.picidae.load([path])	**param path** default ($DATASET_PATH), the path to look for the data and
numpy_datasets.timeseries.esc.load([path])	ESC-10/50: Environmental Sound Classification
numpy_datasets.timeseries.warblr.load([path])	Binary audio classification, presence or absence of a bird.
numpy_datasets.timeseries.gtzan.load([path])	music genre classification
numpy_datasets.timeseries.irmas.load([path])	music instrument classification
numpy_datasets.timeseries.vocalset.load([path])	singer/technique/vowel of singing voices
numpy_datasets.timeseries.freefield1010.load([path])	Audio binary classification, presence or absence of bird songs.
numpy_datasets.timeseries.birdvox_70k.load([path])	a dataset for avian flight call detection in half-second clips
numpy_datasets.timeseries.birdvox_dcase_20k.load([path])	Binary bird detection classification
numpy_datasets.timeseries.seizures_neonatal.load([path])	A dataset of neonatal EEG recordings with seizures annotations
numpy_datasets.timeseries.sonycust.load([path])	multilabel urban sound classification
numpy_datasets.timeseries.gtzan.load([path])	music genre classification
numpy_datasets.timeseries.TUTacousticscenes2017.load([path])	Acoustic Scene classification

2.5.1 Detailed description

numpy_datasets.timeseries.audiomnist.**load**(*path=None*)

digit recognition https://github.com/soerenab/AudioMNIST

A simple audio/speech dataset consisting of recordings of spoken digits in wav files at 48kHz. The recordings are trimmed so that they have near minimal silence at the beginnings and ends.

FSDD is an open dataset, which means it will grow over time as data is contributed. In order to enable reproducibility and accurate citation the dataset is versioned using Zenodo DOI as well as git tags.

Current status

4 speakers 2,000 recordings (50 of each digit per speaker) English pronunciations

numpy_datasets.timeseries.univariate_timeseries.**load**(*path=None*)

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **train_images** (*array*)
- **train_labels** (*array*)
- **valid_images** (*array*)
- **valid_labels** (*array*)
- **test_images** (*array*)
- **test_labels** (*array*)

numpy_datasets.timeseries.speech_commands.**load**(*path=None*)

numpy_datasets.timeseries.picidae.**load**(*path=None*)

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **wavs** (*array*) – the waveforms in the time amplitude domain
- **labels** (*array*) – binary values representing the presence or not of an avian
- **flag** (*array*) – the Xeno-Canto ID

numpy_datasets.timeseries.esc.**load**(*path=None*)

ESC-10/50: Environmental Sound Classification

https://github.com/karolpiczak/ESC-50#download

The ESC-50 dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification.

The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class) loosely arranged into 5 major categories:

> Animals Natural soundscapes & water sounds Human, non-speech sounds Interior/domestic sounds Exterior/urban noises

Clips in this dataset have been manually extracted from public field recordings gathered by the Freesound.org project. The dataset has been prearranged into 5 folds for comparable cross-validation, making sure that fragments from the same original source file are contained in a single fold.

ESC 50.

https://github.com/karolpiczak/ESC-50#download

Parameters **path** (*str (optional)*) – default $DATASET_path), the path to look for the data and where the data will be downloaded if not present

Returns

- **wavs** (*array*) – the wavs as a numpy array (matrix) with first dimension the data and second dimension time
- **fine_labels** (*array*) – the labels of the final classes (50 different ones) as a integer vector
- **coarse_labels** (*array*) – the labels of the classes big cateogry (5 of them)

- **folds** (*array*) – the fold as an integer from 1 to 5 specifying how to split the data one should not split a fold into train and set as it would make the same recording (but different subparts) be present in train and test, biasing optimistically the results.
- **esc10** (*array*) – the boolean vector specifying if the corresponding datum (wav, label, ...) is in the ESC-10 dataset or not. That is, to load the ESC-10 dataset simply load ESC-50 and use this boolean vector to extract only the ESC-10 data.

numpy_datasets.timeseries.warblr.**load**(*path=None*)

Binary audio classification, presence or absence of a bird.

Warblr comes from a UK bird-sound crowdsourcing research spinout called Warblr. From this initiative we have 10,000 ten-second smartphone audio recordings from around the UK. The audio totals around 44 hours duration. The audio will be published by Warblr under a Creative Commons licence. The audio covers a wide distribution of UK locations and environments, and includes weather noise, traffic noise, human speech and even human bird imitations. It is directly representative of the data that is collected from a mobile crowdsourcing initiative. Load the data given a path

numpy_datasets.timeseries.gtzan.**load**(*path=None*)

music genre classification

This dataset was used for the well known paper in genre classification "Musical genre classification of audio signals" by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.

Unfortunately the database was collected gradually and very early on in my research so I have no titles (and obviously no copyright permission etc). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions. Nevetheless I have been providing it to researchers upon request mainly for comparison purposes etc. Please contact George Tzanetakis (gtzan@cs.uvic.ca) if you intend to publish experimental results using this dataset.

There are some practical and conceptual issues with this dataset, described in "The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use" by B. Sturm on arXiv 2013.

numpy_datasets.timeseries.irmas.**load**(*path=None*)

music instrument classification

ref https://zenodo.org/record/1290750#.WzCwSRyxXMU

This dataset includes musical audio excerpts with annotations of the predominant instrument(s) present. It was used for the evaluation in the following article:

Bosch, J. J., Janer, J., Fuhrmann, F., & Herrera, P. "A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals", in Proc. ISMIR (pp. 559-564), 2012

Please Acknowledge IRMAS in Academic Research

IRMAS is intended to be used for training and testing methods for the automatic recognition of predominant instruments in musical audio. The instruments considered are: cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human singing voice. This dataset is derived from the one compiled by Ferdinand Fuhrmann in his PhD thesis, with the difference that we provide audio data in stereo format, the annotations in the testing dataset are limited to specific pitched instruments, and there is a different amount and lenght of excerpts.

numpy_datasets.timeseries.vocalset.**load**(*path=None*)

singer/technique/vowel of singing voices

source: https://zenodo.org/record/1442513#.W7OaFBNKjx4

We present VocalSet, a singing voice dataset consisting of 10.1 hours of monophonic recorded audio of professional singers demonstrating both standard and extended vocal techniques on all 5 vowels. Existing singing voice datasets aim to capture a focused subset of singing voice characteristics, and generally consist of just a few singers. VocalSet contains recordings from 20 different singers (9 male, 11 female) and a range of voice types.

VocalSet aims to improve the state of existing singing voice datasets and singing voice research by capturing not only a range of vowels, but also a diverse set of voices on many different vocal techniques, sung in contexts of scales, arpeggios, long tones, and excerpts. :param path: a string where to load the data and download if not present :type path: str (optional)

Returns

- **singers** (*list*) – the list of singers as strings, 11 males and 9 females as in male1, male2, ...
- **genders** (*list*) – the list of genders of the singers as in male, male, female, ...
- **vowels** (*list*) – the vowels being pronunced
- **data** (*list*) – the list of waveforms, not all equal length

numpy_datasets.timeseries.freefield1010.**load**(*path=None*)

Audio binary classification, presence or absence of bird songs. freefield1010. is a collection of over 7,000 excerpts from field recordings around the world, gathered by the FreeSound project, and then standardised for research. This collection is very diverse in location and environment, and for the BAD Challenge we have newly annotated it for the presence/absence of birds.

numpy_datasets.timeseries.birdvox_70k.**load**(*path=None*)

a dataset for avian flight call detection in half-second clips

Version 1.0, April 2018.

Created By

Vincent Lostanlen (1, 2, 3), Justin Salamon (2, 3), Andrew Farnsworth (1), Steve Kelling (1), and Juan Pablo Bello (2, 3).

(1): Cornell Lab of Ornithology (CLO) (2): Center for Urban Science and Progress, New York University (3): Music and Audio Research Lab, New York University

https://wp.nyu.edu/birdvox

Description

The BirdVox-70k dataset contains 70k half-second clips from 6 audio recordings in the BirdVox-full-night dataset, each about ten hours in duration. These recordings come from ROBIN autonomous recording units, placed near Ithaca, NY, USA during the fall 2015. They were captured on the night of September 23rd, 2015, by six different sensors, originally numbered 1, 2, 3, 5, 7, and 10.

Andrew Farnsworth used the Raven software to pinpoint every avian flight call in time and frequency. He found 35402 flight calls in total. He estimates that about 25 different species of passerines (thrushes, warblers, and sparrows) are present in this recording. Species are not labeled in BirdVox-70k, but it is possible to tell apart thrushes from warblers and sparrows by looking at the center frequencies of their calls. The annotation process took 102 hours.

The dataset can be used, among other things, for the research,development and testing of bioacoustic classification models, including the reproduction of the results reported in [1].

For details on the hardware of ROBIN recording units, we refer the reader to [2].

[1] V. Lostanlen, J. Salamon, A. Farnsworth, S. Kelling, J. Bello. BirdVox-full-night: a dataset and benchmark for avian flight call detection. Proc. IEEE ICASSP, 2018.

[2] J. Salamon, J. P. Bello, A. Farnsworth, M. Robbins, S. Keen, H. Klinck, and S. Kelling. Towards the Automatic Classification of Avian Flight Calls for Bioacoustic Monitoring. PLoS One, 2016.

@inproceedings{lostanlen2018icassp, title = {BirdVox-full-night: a dataset and benchmark for avian flight call detection}, author = {Lostanlen, Vincent and Salamon, Justin and Farnsworth, Andrew and Kelling, Steve and Bello, Juan Pablo}, booktitle = {Proc. IEEE ICASSP}, year = {2018}, published = {IEEE}, venue = {Calgary, Canada}, month = {April}, }

Parameters **path** (*str (optional)*) – default ($DATASET_PATH), the path to look for the data and where the data will be downloaded if not present

Returns

- **wavs** (*array(70804, 12000)*) – the waveforms in the time amplitude domain
- **labels** (*array(70804,)*) – binary values representing the presence or not of an avian
- **recording** (*array(70804,)*) – the file number from which the sample has been extracted

numpy_datasets.timeseries.birdvox_dcase_20k.**load**(*path=None*)

Binary bird detection classification

Dataset is 16.5Go compressed.

BirdVox-DCASE-20k: a dataset for bird audio detection in 10-second clips

Version 2.0, March 2018.

link

Description

The BirdVox-DCASE-20k dataset contains 20,000 ten-second audio recordings. These recordings come from ROBIN autonomous recording units, placed near Ithaca, NY, USA during the fall 2015. They were captured on the night of September 23rd, 2015, by six different sensors, originally numbered 1, 2, 3, 5, 7, and 10.

Out of these 20,000 recording, 10,017 (50.09%) contain at least one bird vocalization (either song, call, or chatter).

The dataset is a derivative work of the BirdVox-full-night dataset [1], containing almost as much data but formatted into ten-second excerpts rather than ten-hour full night recordings.

In addition, the BirdVox-DCASE-20k dataset is provided as a development set in the context of the "Bird Audio Detection" challenge, organized by DCASE (Detection and Classification of Acoustic Scenes and Events) and the IEEE Signal Processing Society.

The dataset can be used, among other things, for the development and evaluation of bioacoustic classification models.

We refer the reader to [1] for details on the distribution of the data and [2] for details on the hardware of ROBIN recording units.

[1] V. Lostanlen, J. Salamon, A. Farnsworth, S. Kelling, J.P. Bello. "BirdVox-full-night: a dataset and benchmark for avian flight call detection", Proc. IEEE ICASSP, 2018.

[2] J. Salamon, J. P. Bello, A. Farnsworth, M. Robbins, S. Keen, H. Klinck, and S. Kelling. Towards the Automatic Classification of Avian Flight Calls for Bioacoustic Monitoring. PLoS One, 2016.

Data Files

The wav folder contains the recordings as WAV files, sampled at 44,1 kHz, with a single channel (mono). The original sample rate was 24 kHz.

The name of each wav file is a random 128-bit UUID (Universal Unique IDentifier) string, which is randomized with respect to the origin of the recording in BirdVox-full-night, both in terms of time (UTC hour at the start of the excerpt) and space (location of the sensor).

The origin of each 10-second excerpt is known by the challenge organizers, but not disclosed to the participants.

Please Acknowledge BirdVox-DCASE-20k in Academic Research

When BirdVox-70k is used for academic research, we would highly appreciate it if scientific publications of works partly based on this dataset cite the following publication:

V. Lostanlen, J. Salamon, A. Farnsworth, S. Kelling, J. Bello. "BirdVox-full-night: a dataset and benchmark for avian flight call detection", Proc. IEEE ICASSP, 2018.

The creation of this dataset was supported by NSF grants 1125098 (BIRDCAST) and 1633259 (BIRDVOX), a Google Faculty Award, the Leon Levy Foundation, and two anonymous donors. :param path: default ($DATASET_PATH), the path to look for the data and

where the data will be downloaded if not present

Returns

- **wavs** (*array*) – the waveforms in the time amplitude domain
- **labels** (*array*) – binary values representing the presence or not of an avian
- **recording** (*array*) – the file number from which the sample has been extracted

numpy_datasets.timeseries.seizures_neonatal.**load**(*path=None*)

A dataset of neonatal EEG recordings with seizures annotations

source: https://zenodo.org/record/2547147

Neonatal seizures are a common emergency inthe neonatal intensive care unit (NICU). There are many questions yet to be answered regarding the temporal/spatial characteristics of seizures from different pathologies, response to medication, effects on neurodevelopment and optimal detection. This dataset contains EEG recordings from human neonates and the visual interpretation of the EEG by the human expert. Multi-channel EEG was recorded from 79 term neonates admitted to the neonatal intensive care unit (NICU) at the Helsinki University Hospital. The median recording duration was 74 minutes (IQR: 64 to 96 minutes). EEGs were annotated by three experts for the presence of seizures. An average of 460 seizures were annotated per expert in the dataset, 39 neonates had seizures by consensus and 22 were seizure free by consensus. The dataset can be used as a reference set of neonatal seizures, for the development of automated methods of seizure detection and other EEG analysis, as well as for the analysis of inter-observer agreement. :param path: a string where to load the data and download if not present :type path: str (optional)

Returns

- **annotations** (*list*) – the list of multichannel binary vectors representing the presence or absence of seizure, 3 channels due to 3 expert annotations
- **waveforms** (*list*) – list of (channels, TIME) multichannel EEGs

numpy_datasets.timeseries.sonycust.**load**(*path=None*)

multilabel urban sound classification

Reference at https://zenodo.org/record/3233082

Description

SONYC Urban Sound Tagging (SONYC-UST) is a dataset for the development and evaluation of machine listening systems for realistic urban noise monitoring. The audio was recorded from the SONYC acoustic sensor network. Volunteers on the Zooniverse citizen science platform tagged the presence of 23 classes that were chosen in consultation with the New York City Department of Environmental Protection. These 23 fine-grained classes can be grouped into 8 coarse-grained classes. The recordings are split into three subsets: training, validation, and test. These sets are disjoint with respect to the sensor from which each recording came. For increased reliability, three volunteers annotated each recording, and members of the SONYC team subsequently created a set of ground-truth tags for the validation set using a two-stage annotation procedure in which two annotators independently tagged and then collectively resolved any disagreements. For more details on the motivation and creation of this dataset see the DCASE 2019 Urban Sound Tagging Task website.

Audio data

The provided audio has been acquired using the SONYC acoustic sensor network for urban noise pollution monitoring. Over 50 different sensors have been deployed in New York City, and these sensors have collectively gathered the equivalent of 37 years of audio data, of which we provide a small subset. The data was sampled by selecting the nearest neighbors on VGGish features of recordings known to have classes of interest. All recordings are 10 seconds and were recorded with identical microphones at identical gain settings. To maintain privacy, the recordings in this release have been distributed in time and location, and the time and location of the recordings are not included in the metadata.

Labels

there are fine and coarse labels engine 1: small-sounding-engine 2: medium-sounding-engine 3: large-sounding-engine X: engine-of-uncertain-size machinery-impact 1: rock-drill 2: jackhammer 3: hoe-ram 4: pile-driver X: other-unknown-impact-machinery non-machinery-impact 1: non-machinery-impact powered-saw 1: chainsaw 2: small-medium-rotating-saw 3: large-rotating-saw X: other-unknown-powered-saw alert-signal 1: car-horn 2: car-alarm 3: siren 4: reverse-beeper X: other-unknown-alert-signal music 1: stationary-music 2: mobile-music 3: ice-cream-truck X: music-from-uncertain-source human-voice 1: person-or-small-group-talking 2: person-or-small-group-shouting 3: large-crowd 4: amplified-speech X: other-unknown-human-voice dog 1: dog-barking-whining

numpy_datasets.timeseries.gtzan.**load**(*path=None*)

music genre classification

This dataset was used for the well known paper in genre classification "Musical genre classification of audio signals" by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.

Unfortunately the database was collected gradually and very early on in my research so I have no titles (and obviously no copyright permission etc). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions. Nevetheless I have been providing it to researchers upon request mainly for comparison purposes etc. Please contact George Tzanetakis (gtzan@cs.uvic.ca) if you intend to publish experimental results using this dataset.

There are some practical and conceptual issues with this dataset, described in "The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use" by B. Sturm on arXiv 2013.

2.6 numpy_datasets.utils

numpy_datasets.utils.patchify_1d(x, ...)	extract patches from a numpy array
numpy_datasets.utils.patchify_2d(x, ...)	
numpy_datasets.utils.train_test_split(*args)	split given data into two non overlapping sets
numpy_datasets.utils.batchify(*args, batch_size)	
numpy_datasets.utils.resample_images(images, ...)	
numpy_datasets.utils.download_dataset(path, ...)	dataset downlading utility
numpy_datasets.utils.extract_file(filename, ...)	

2.6.1 Detailed description

numpy_datasets.utils.**patchify_1d**(*x*, *window_length*, *stride*)
extract patches from a numpy array

Parameters

- **x** (*array-like*) – the input data to extract patches from, any shape, the last dimension is the one being patched
- **window_length** (*int*) – the length of the patches
- **stride** (*int*) – the amount of stride (bins separating two consecutive patches

Returns **x_patches** – the number of patches is put in the pre-last dimension (-2)

Return type array-like

numpy_datasets.utils.**patchify_2d**(*x*, *window_length*, *stride*)

numpy_datasets.utils.**train_test_split**(**args*, *train_size=0.8*, *stratify=None*, *seed=None*)
split given data into two non overlapping sets

Parameters

- ***args** (*inputs*) – the sets to be split by the function
- **train_size** (*scalar*) – the amount of data to put in the first set, either an integer value being the actual number of data to keep, or a ratio (0 to 1 number)
- **stratify** (*array (optional)*) – the optimal stratify guide to spit the array s.t. the same proportion based on the stratify array is kep in both set based on the proportion of the split
- **seed** (*integer (optional)*) – the seed for the random number generator for reproducibility

Returns

- **train_set** (*list*) – returns the train data, the list has the members of *args split
- **test_set** (*list*) – returns the test data, the list has the members of *args split

Example

```
x = numpy.random.randn(100, 4)
y = numpy.random.randn(100)

train, test = train_test_split(x, y, train_size=0.5)
print(train[0].shape, train[1].shape)
# (50, 4) (50,)
print(test[0].shape, test[1].shape)
# (50, 4) (50,)
```

class numpy_datasets.utils.**batchify**(**args*, *batch_size*, *option='random'*, *load_func=None*, *extra_process=0*, *n_batches=None*)

numpy_datasets.utils.**resample_images**(*images*, *target_shape*, *ratio='same'*, *order=1*, *mode='nearest'*, *data_format='channels_first'*)

numpy_datasets.utils.**download_dataset**(*path*, *dataset*, *urls_names*, *baseurl=''*, *extract=False*)
dataset downlading utility

Args:

path: string the path where the dataset should be download

dataset: string the name of the dataset, used as the folder name

urls_names: dict dictionnary mapping urls to filename. If the urls have a common root, then it can be omited from this variable and put into the baseurl argument

baseurl: string the common url to prepend onto each url in urls_names

numpy_datasets.utils.**extract_file**(*filename*, *target*)

2.7 numpy_datasets.preprocess

numpy_datasets.preprocess.ZCAWhitening([...])
numpy_datasets.preprocess.Standardize([eps, ...])

2.7.1 Detailed description

class numpy_datasets.preprocess.**ZCAWhitening**(*eps=0.0001*, *name=''*)

class numpy_datasets.preprocess.**Standardize**(*eps=1e-06, axis=[0], name=''*)

Python Module Index

n

Index

T

Z