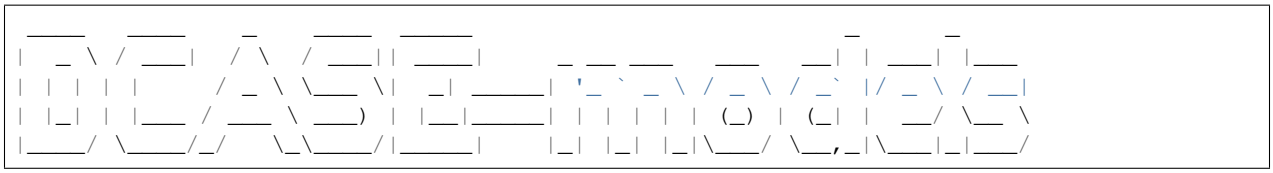
DCASE-models

May 11, 2021

Contents:



DCASE-models is an open-source Python library for rapid prototyping of environmental sound analysis systems, with an emphasis on deep-learning models.

CHAPTER 1

Introduction

DCASE-models is an open-source Python library for rapid prototyping of environmental sound analysis systems, with an emphasis on deep–learning models. The project is on GitHub.

It's main features / design goals are:

- ease of use,
- rapid prototyping of environmental sound analysis systems,
- a simple and lightweight set of basic components that are generally part of a computational environmental audio analysis system,
- a collection of functions for dataset handling, data preparation, feature extraction, and evaluation (most of which rely on existing tools),
- a model interface to standardize the interaction of machine learning methods with the other system components,
- an abstraction layer to make the library independent of the backend used to implement the machine learning model,
- inclusion of reference implementations for several state-of-the-art algorithms.

DCASE-models is a work in progress, thus input is always welcome.

The available documentation is limited for now, but *you can help to improve it*.

CHAPTER 2

Installation instructions

We recommend to install *DCASE-models* in a dedicated virtual environment. For instance, using anaconda:

```
conda create -n dcase python=3.6
conda activate dcase
```

DCASE-models uses SoX for functions related to the datasets. You can install it in your conda environment by:

```
conda install -c conda-forge sox
```

Before installing the library, you must install only one of the Tensorflow variants (CPU-only or GPU):

```
pip install "tensorflow<1.14" # for CPU-only version
pip install "tensorflow-gpu<1.14" # for GPU version
```

Then, you can install the library through the Python Package Index (PyPI) or from the source as explained below.

2.1 pypi

The simplest way to install *DCASE-models* is through the Python Package Index (PyPI). This will ensure that all required dependencies are fulfilled. This can be achieved by executing the following command:

```
pip install dcase_models
```

or:

```
sudo pip install dcase_models
```

to install system-wide, or:

```
pip install -u dcase_models
```

to install just for your own user.

2.2 source

If you've downloaded the archive manually from the releases page, you can install using the *setuptools* script:

```
tar xzf dcase_models-VERSION.tar.gz
cd dcase_models-VERSION/
python setup.py install
```

If you intend to develop *DCASE-models* or make changes to the source code, you can install with *pip install -e* to link to your actively developed source tree:

```
tar xzf dcase_models-VERSION.tar.gz
cd dcase_models-VERSION/
pip install -e .
```

Alternately, the latest development version can be installed via pip:

```
pip install git+https://github.com/pzinemanas/dcase_models
```

2.3 sox

Say something about installing other dependencies such as *sox*.

CHAPTER 3

Tutorial and examples

This is a tutorial introduction to quickly get you up and running with *DCASE-models*

The package of the library includes a set of examples, organized into three different categories, which illustrate the usefulness of *DCASE-models* for carrying out research experiments or developing applications. These examples can also be used as templates to be adapted for implementing specific DCASE methods. The type of examples provided are:

- scripts that perform each step in the typical development pipeline of a DCASE task
- Jupyter Notebooks that replicate some of the experiments reported in the literature
- a web interface for sound classification as an example of a high–level application

The *following section* gives a walk-through of the example scripts provided. Then, the *next section* describes the library organization and exemplifies the use of the most important classes and functionalities.

3.1 Example scripts

A set of Python scripts is provided in the `examples` folder of the package. They perform each step in the typical development pipeline of a DCASE task, i.e downloading a dataset, data augmentation, feature extraction, model training, fine-tuning, and model evaluation. Follow the instructions bellow to know how they are used.

3.1.1 Parameters setting

First, note that the default parameters are stored in the `parameters.json` file at the root folder of the package. You can use other `parameters.json` file by passing its path in the `-p` (or `--path`) argument of each script.

3.1.2 Usage information

In the following, we show examples on how to use these scripts for the typical development pipeline step by step. For further usage information please check each script instructions by typing:

```
python download_dataset.py --help
```

3.1.3 Dataset downloading

First, let's start by downloading a dataset. For instance, to download the ESC-50 dataset just type:

```
python download_dataset.py -d ESC50
```

Note: Note that by default the dataset will be downloaded to the `../datasets/ESC50` folder, following the path set in the `parameters.json` file. You can change the path or the `parameters.json` file. The datasets available are listed in the *Datasets* section.

3.1.4 Data augmentation

If you want to use data augmentation techniques on this dataset, you can run the following script:

```
python data_augmentation.py -d ESC50
```

Note: Note that the name and the parameters of each transformation are defined in the `parameters.json` file. The augmentations implemented so far are pitch-shifting, time-stretching, and white noise addition. Please check the *AugmentedDataset* class for further information.

3.1.5 Feature extraction

Now, you can extract the features for each file in the dataset by typing:

```
python extract_features.py -d ESC50 -f MelSpectrogram
```

Note: Note that you have to specify the features name by the `-f` argument, in this case *MelSpectrogram*. All the features representations available are listed in the *Features* section.

3.1.6 Model training

To train a model is also very straightforward. For instance, to train the *SB_CNN* model on the ESC-50 dataset with the *MelSpectrogram* features extracted before just type:

```
python train_model.py -d ESC50 -f MelSpectrogram -m SB_CNN -fold fold1
```

Note: Note that in this case you have to pass the model name and a fold name as an argument, using `-m` and `-fold`, respectively. This fold is considered to be the fold for testing, meaning that it will not be used during training. All the implemented models available are listed in the *Implemented models* section.

3.1.7 Model evaluation

Once the model is trained, you can evaluate the model in the test set by typing:

```
python evaluate_model.py -d ESC50 -f MelSpectrogram -m SB_CNN -fold fold1
```

Note: Note that the fold specified as an argument is the one used for testing. This scripts prints the results that we get from sed_eval library.

3.1.8 Fine-tuning

Once you have a model trained in some dataset, you can fine-tune the model on another dataset. For instance, to use a pre-trained model on the ESC-50 dataset and fine-tune it on the MAVD dataset just type:

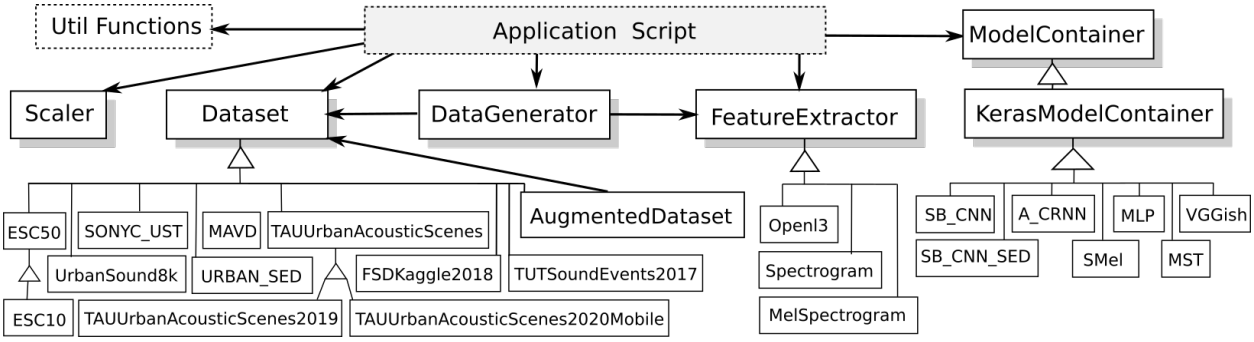
```
python fine_tuning.py -od ESC50 -ofold fold1 -f MelSpectrogram -m SB_CNN -d MAVD -
↪fold test
```

Note: Note that the information of the original dataset is set by the `-od` and `-ofold` arguments. Besides, the `-d` and `-fold` arguments set the new dataset and the test fold, respectively.

3.2 Library organization

A description of the main classes and functionalities of the library is presented in this section, following the order of the typical pipeline: dataset preparation, data augmentation, feature extraction, data loading, data scaling, and model handling. Example code is provided for each step, but please check the documentation of the classes for further information.

The following is a class diagram of *DCASE-models* showing all the base classes and some of the implemented specializations.



3.2.1 Dataset

This is the base class designed to manage a dataset, its paths, and its internal structure. It includes methods to download the data, resample the audio files, and check that both processes succeed.

The library covers several publicly available datasets related to different tasks. Please check the list of currently available datasets in the *Datasets* section.

Each dataset is implemented in the library as a class that inherits from *Dataset*. This design provides a common and simple interface to work with any dataset. For instance, to use the UrbanSound8k dataset, it is enough to initialize its class with the path to the data folder, as follows.

```
dataset = UrbanSound8k(DATASET_PATH)
```

Then, the following methods are used to download the dataset and change its sampling rate (to 22050 Hz).

```
dataset.download()
dataset.change_sampling_rate(22050)
```

Note: Note that most of the datasets devised for research include a fold split and a corresponding evaluation setup (e.g. 5-fold cross-validation). This fold split is generally carefully selected to avoid biases and data contamination. In order to keep the results comparable to those reported in the literature, DCASE-models uses, whenever available, the predefined splits for each dataset. However, the user may define different splits or evaluation setups if needed.

3.2.2 AugmentedDataset

The previously defined `dataset` instance can be expanded using data augmentation techniques. The augmentations implemented so far are pitch-shifting, time-stretching, and white noise addition. The first two are carried out by means of pysox.

An augmented version of a given dataset can be obtained by initializing an instance of the *AugmentedDataset* class with the `dataset` as a parameter, as well as a dictionary containing the name and parameters of each transformation.

```
aug_dataset = AugmentedDataset(dataset,
                               augmentations)
```

After initialization, the following method will perform the actual augmentation and create new audio files for every dataset element according to the type and parameters of each augmentation.

```
aug_dataset.process()
```

Note: Note that the augmented dataset is indeed an instance of *Dataset*, so it can be used as any other dataset in the following steps of the pipeline.

3.2.3 FeatureExtractor

This is the base class to define different types of feature representations. It has methods to load an audio file, extract features, and save them. It can also check if the features were already extracted.

Feature representations are implemented as specializations of the base class *FeatureExtractor*, for instance, *Spectrogram*. Please check the list of currently available features in the *Features* section.

A *FeatureExtractor* is initialized with some parameters. For instance, to define a *MelSpectrogram* feature extractor the parameters are: length and hop in seconds of the feature representation analysis window (`sequence_time` and `sequence_hop_time`); window length and hop size (in samples) for the short-time Fourier Transform (STFT) calculation (`audio_win` and `audio_hop`); and the audio sampling rate (`sr`).

```
features = Spectrogram(sequence_time=1.0, sequence_hop_time=0.5,
                       audio_win=1024, audio_hop=512, sr=22050)
```

After initialization, the following method computes the features for each audio file in the dataset.

```
features.extract(dataset)
```

Once the features are extracted and saved to disk, they can be loaded using *DataGenerator* as explained in the following.

Note: Note that if the audio files are not sampled at the given frequency, they are converted before calculating the features.

3.2.4 DataGenerator

This class uses instances of *Dataset* and *FeatureExtractor* to prepare the data for model training, validation and testing. An instance of this class is created for each one of these processes.

```
data_gen_train = DataGenerator(dataset,
                               features,
                               train=True,
                               folds=['train'])

data_gen_val = DataGenerator(dataset,
                             features,
                             train=False,
                             folds=['val'])
```

At this point of the pipeline, the features and the annotations for training the model can be obtained as follows.

```
X_train, Y_train = data_gen_train.get_data()
```

Note: Note that instances of *DataGenerator* can be used to load data in batches. This feature is especially useful for traininga models on systems with memory limitations.

3.2.5 Scaler

Before feeding data to a model, it is common to normalize the data or scale it to a fixed minimum and maximum value. To do this, the library contains a *Scaler* class, based on scikit-learn preprocessing functions, that includes `fit` and `transform` methods.

```
scaler = Scaler("standard")
scaler.fit(X_train)
X_train = scaler.transform(X_train)
```

In addition, the scaler can be fitted in batches by means of passing the *DataGenerator* instance instead of the data itself.

```
scaler.fit(data_gen_train)
```

It is also possible to scale the data as it is being loaded from the disk, for instance, when training the model. To do so, the *Scaler* can be passed to the *DataGenerator* after its initialization.

```
data_gen_val.set_scaler(scaler)
```

3.2.6 ModelContainer

This class defines an interface to standardize the behavior of machine learning models. It stores the architecture and the parameters of the model. It provides methods to train and evaluate the model, and to save and load its architecture and weights. It also allows the inspection of the output of its intermediate stages (i.e. layers).

The library also provides a container class to define Keras models, namely *KerasModelContainer*, that inherits from *ModelContainer*, and implements its functionality using this specific machine learning backend. Even though the library currently supports only Keras, it is easy to specialize the *ModelContainer* class to integrate other machine learning tools, such as PyTorch.

Each model has its own class that inherits from a specific *ModelContainer*, such as *KerasModelContainer*. Please check the list of currently available features in the *Implemented models* section.

A model's container has to be initialized with some parameters. These parameters vary across models, among which the most important are: input shape, number of classes, and evaluation metrics. Specific parameters may include the number of hidden layers or the number of convolutional layers, among others.

```
model_cont = SB_CNN(**model_params)
```

The *ModelContainer* class has a method to train the model. Training parameters can include, for example, number of epochs, learning rate and batch size.

```
model_cont.train((X_train, Y_train), **train_params)
```

To train the model in batches, the *DataGenerator* object can be passed to the `train` method instead of the pre-loaded data.

```
model_cont.train(data_gen_train, **train_params)
```

Performing model evaluation is also simple. For instance, the following code uses the test set for evaluating the model.

```
data_gen_test = DataGenerator(dataset,
                              features,
                              train=False,
                              folds=['test'])
X_test, Y_test = data_gen_test.get_data()
results = model_cont.evaluate((X_test, Y_test))
```

The results' format depends on which metrics are used. By default, the evaluation is performed using the metrics available from the sed_eval library. Therefore, the results are presented accordingly. Nevertheless, *DCASE-models* enables the use of others evaluating frameworks such as psds_eval, or the use of user-defined metrics in a straightforward way.

When building deep-learning models it is common practice to use fine-tuning and transfer learning techniques. In this way, one can reuse a network that was previously trained on another dataset or for another task, and adapt it to the problem at hand. This type of approach can also be carried out with the *ModelContainer*.

CHAPTER 4

Extending the library

This section includes clear instructions on how to extend different components of *DCASE-models*.

4.1 Datasets

Each dataset is implemented in the library as a class that inherits from *Dataset*.

To include a new dataset in the library you should extend the *Dataset* class and implement:

- `__init__` , where you can define and store arguments related to the dataset.
- `build` , where you define the fold list, label list, paths, etc.
- `generate_file_lists` , where you define the dataset structure.
- `get_annotations` , where you implement the function to get the annotations from a given audio file.
- `download` , where you implement the steps to download and decompress the dataset.

Below we follow all the necessary steps to implement a new dataset. Let's assume that the new dataset has two labels (dog and cat), three folds (train, validate and test), and the audio files are stored in `DATASET_PATH/audio`. Besides the new dataset has the following structure:

```
DATASET_PATH/
|
|- audio/
|   |- train
|   |   |- file1-0-X.wav
|   |   |- file2-1-X.wav
|   |   |- file3-0-X.wav
|   |
|   |- validate
|   |   |- file1-1-Y.wav
|   |   |- file2-0-Y.wav
|   |   |- file3-1-Y.wav
```

(continues on next page)

(continued from previous page)

```
|   |
|   |- test
|   |   |- file1-1-Z.wav
|   |   |- file2-0-Z.wav
|   |   |- file3-0-Z.wav
```

Note that each fold has a folder inside the audio path. Also the file name includes the class label coded after the first dash character (0 for dog, 1 for cat).

The first step is to create a new class that inherits from *Dataset*, and implement its `__init__()` method. Since the only argument needed for this custom dataset is its path, we simply initialize the `super().__init__()` method. If your dataset needs other arguments from the user, add them here.

```
from dcase_models.data.dataset_base import Dataset


class CustomDataset(Dataset):
    def __init__(self, dataset_path):
        # Don't forget to add this line
        super().__init__(dataset_path)
```

Now implement the *build()* method. You should define here the `audio_path`, `fold_list` and `label_list` attributes. You can also define other attributes for your dataset.

```
def build(self):
    self.audio_path = os.path.join(self.dataset_path, 'audio')
    self.fold_list = ["train", "validate", "test"]
    self.label_list = ["dog", "cat"]
    self.evaluation_mode = 'train-validate-test'
```

The *generate_file_lists()* method defines the structure of the dataset. Basically this structure is defined in the `self.file_lists` dictionary. This dictionary stores the list of the paths to the audio files for each fold in the dataset. Note that you can use the *list_wav_files()* function to list all wav files in a given path.

```
def generate_file_lists(self):
    for fold in self.fold_list:
        audio_folder = os.path.join(self.audio_path, fold)
        self.file_lists[fold] = list_wav_files(audio_folder)
```

Now let's define *get_annotations()*. This method receives three arguments: the path to the audio file, the features representation and the time resolution (used when the annotations are defined following a fix time-grid, e.g see *URBAN_SED*). Note that the first dimension (index sequence) of the annotations and the feature representation coincide. In this example the label of each audio file is coded in its name as explained before.

```
def get_annotations(self, file_name, features, time_resolution):
    y = np.zeros((len(features), len(self.label_list)))
    class_ix = int(os.path.basename(file_name).split('-')[1])
    y[:, class_ix] = 1
    return y
```

The *download()* method defines the steps for downloading the dataset. You can use the *download()* method from the parent *Dataset* to download and decompress all files from zenodo. Also you can use *move_all_files_to_parent()* function to move all files from a subdirectory to the parent.

```
def download(self, force_download=False):
    zenodo_url = "https://zenodo.org/record/1234567/files"
```

(continues on next page)

(continued from previous page)

```
    zenodo_files = ["CustomDataset.tar.gz"]
    downloaded = super().download(
        zenodo_url, zenodo_files, force_download
    )
    if downloaded:
        # mv self.dataset_path/CustomDataset/* self.dataset_path/
        move_all_files_to_parent(self.dataset_path, "CustomDataset")
        # Don't forget this line
        self.set_as_downloaded()
```

Note: If you implement a class for a publicly available dataset that is not present in *Dataset*, consider filing a Github issue or, even better, sending us a pull request.

4.2 Features

Feature representations are implemented as specializations of the base class *FeatureExtractor*.

In order to implement a new feature you should write a class that inherits from *FeatureExtractor*.

The methods you should reimplement are:

- `__init__` , where you can define and store the features arguments.
- `calculate` , where you define the feature calculation process.

For instance, if you want to implement Chroma features:

```
import numpy as np
import librosa
from dcase_models.data.features import FeatureExtractor


class Chroma(FeatureExtractor):
    def __init__(self, sequence_time=1.0, sequence_hop_time=0.5,
                 audio_win=1024, audio_hop=680, sr=22050,
                 n_fft=1024, n_chroma=12, pad_mode='reflect'):

        super().__init__(sequence_time=sequence_time,
                         sequence_hop_time=sequence_hop_time,
                         audio_win=audio_win, audio_hop=audio_hop,
                         sr=sr)

        self.n_fft = n_fft
        self.n_chroma = n_chroma
        self.pad_mode = pad_mode

    def calculate(self, file_name):
        # Load the audio signal
        audio = self.load_audio(file_name)

        # Pad audio signal
        if self.pad_mode is not None:
            audio = librosa.util.fix_length(
                audio,
```

(continues on next page)

(continued from previous page)

```
                audio.shape[0] + librosa.core.frames_to_samples(
                    self.sequence_frames, self.audio_hop, n_fft=self.n_fft),
                axis=0, mode=self.pad_mode
            )

        # Get the spectrogram, shape (n_freqs, n_frames)
        stft = librosa.core.stft(audio, n_fft=self.n_fft,
                                 hop_length=self.audio_hop,
                                 win_length=self.audio_win, center=False)
        # Convert to power
        spectrogram = np.abs(stft)**2

        # Convert to chroma_stft, shape (n_chroma, n_frames)
        chroma = librosa.feature.chroma_stft(
            S=spectrogram, sr=self.sr, n_fft=self.n_fft, n_chroma=self.n_chroma)

        # Transpose time and freq dims, shape (n_frames, n_chroma)
        chroma = chroma.T

        # Windowing, creates sequences
        chroma = np.ascontiguousarray(chroma)
        chroma = librosa.util.frame(
            chroma, self.sequence_frames, self.sequence_hop, axis=0
        )

        return chroma
```

4.3 Models

The models are implemented as specializations of the base class `KerasModelContainer`.

To include a new model in the library you should extend the `KerasModelContainer` class and implement the following methods:

- `__init__` , where you can define and store the model arguments.
- `build` , where you define the model architecture.

Note that you might also reimplement the *`train()`* method. This specially useful for complex models (multiple inputs and outputs, custom loss functions, etc.)

For instance, to implement a simple Convolutional Neural Network:

```
from keras.layers import Input, Lambda, Conv2D, MaxPooling2D
from keras.layers import Dropout, Dense, Flatten
from keras.layers import BatchNormalization
from keras.models import Model
import keras.backend as K
from dcase_models.model.container import KerasModelContainer


class CNN(KerasModelContainer):
    def __init__(self, model=None, model_path=None,
                 metrics=['classification'], n_classes=10,
                 n_frames=64, n_freqs=128):
```

(continues on next page)

(continued from previous page)

```
        self.n_classes = n_classes
        self.n_frames = n_frames
        self.n_freqs = n_freqs

        # Don't forget this line
        super().__init__(model=model, model_path=model_path,
                         model_name='MLP', metrics=metrics)

    def build(self):
        # input
        x = Input(shape=(self.n_frames, self.n_freqs), dtype='float32', name='input')

        # expand dims
        y = Lambda(lambda x: K.expand_dims(x, -1), name='expand_dims')(x)

        # CONV 1
        y = Conv2D(24, (5, 5), padding='valid',
                   activation='relu', name='conv1')(y)
        y = MaxPooling2D(pool_size=(2, 2), strides=None,
                         padding='valid', name='maxpool1')(y)
        y = BatchNormalization(name='batchnorm1')(y)

        # CONV 2
        y = Conv2D(24, (5, 5), padding='valid',
                   activation='relu', name='conv2')(y)
        y = BatchNormalization(name='batchnorm2')(y)

        # Flatten and Dropout
        y = Flatten(name='flatten')(y)
        y = Dropout(0.5, name='dropout1')(y)

        # Dense layer
        y = Dense(self.n_classes, activation='softmax', name='out')(y)

        # Create model
        self.model = Model(inputs=x, outputs=y, name='model')

        # Don't forget this line
        super().build()
```

CHAPTER 5

Development

As an open-source project by researchers for researchers, we highly welcome any contribution!

5.1 What to contribute

5.1.1 Give feedback

To send us general feedback, questions or ideas for improvement, please post on our mailing list.

5.1.2 Report bugs

Please report any bugs at the issue tracker on GitHub. If you are reporting a bug, please include:

- your version of madmom,
- steps to reproduce the bug, ideally reduced to as few commands as possible,
- the results you obtain, and the results you expected instead.

If you are unsure whether the experienced behaviour is intended or a bug, please just ask on our mailing list first.

5.1.3 Fix bugs

Look for anything tagged with "bug" on the issue tracker on GitHub and fix it.

5.1.4 Features

Please do not hesitate to propose any ideas at the issue tracker on GitHub. Think about posting them on our mailing list first, so we can discuss it and/or guide you through the implementation.

Alternatively, you can look for anything tagged with "feature request" or "enhancement" on the issue tracker on GitHub.

5.1.5 Write documentation

Whenever you find something not explained well, misleading or just wrong, please update it! The *Edit on GitHub* link on the top right of every documentation page and the *[source]* link for every documented entity in the API reference will help you to quickly locate the origin of any text.

5.2 How to contribute

5.2.1 Edit on GitHub

As a very easy way of just fixing issues in the documentation, use the *Edit on GitHub* link on the top right of a documentation page or the *[source]* link of an entity in the API reference to open the corresponding source file in GitHub, then click the *Edit this file* link to edit the file in your browser and send us a Pull Request.

For any more substantial changes, please follow the steps below.

5.2.2 Fork the project

First, fork the project on GitHub.

Then, follow the *general installation instructions* and, more specifically, the *installation from source*. Please note that you should clone from your fork instead.

5.2.3 Documentation

The documentation is generated with Sphinx. To build it locally, run the following commands:

```
cd docs
make html
```

Afterwards, open `docs/_build/html/index.html` to view the documentation as it would appear on readthedocs. If you changed a lot and seem to get misleading error messages or warnings, run `make clean html` to force Sphinx to recreate all files from scratch.

When writing docstrings, follow existing documentation as much as possible to ensure consistency throughout the library. For additional information on the syntax and conventions used, please refer to the following documents:

- reStructuredText Primer
- Sphinx reST markup constructs
- A Guide to NumPy/SciPy Documentation

CHAPTER 6

Citation

If you use *DCASE-models* in your work, please consider citing it:

```
@inproceedings{DCASE-models,
   Title = {{DCASE-models: a Python library for Computational Environmental Sound␣
↪Analysis using deep-learning models}},
   Author = {Zinemanas, Pablo and Hounie, Ignacio and Cancela, Pablo and Font,␣
↪Frederic and Rocamora, Martín and Serra, Xavier},
   Booktitle = {Proceedings of the Detection and Classification of Acoustic Scenes␣
↪and Events (DCASE) Workshop},
   Month = {11},
   Year = {2020},
   Pages = {??--??},
   Address = {Tokyo, Japan},
   Doi = {10.00/00000.0000}
}
```

CHAPTER 7

Data

7.1 Datasets

Datasets are implemented as specializations of the base class Dataset.

Dataset(dataset_path)	Abstract base class to load and manage DCASE datasets.
UrbanSound8k(dataset_path)	UrbanSound8k dataset.
ESC50(dataset_path)	ESC-50 dataset.
ESC10(dataset_path)	ESC-10 dataset.
URBAN_SED(dataset_path)	URBAN-SED dataset.
SONYC_UST(dataset_path)	SONYC-UST dataset.
TAUUrbanAcousticScenes2019(dataset_path)	TAU Urban Acoustic Scenes 2019 dataset.
TAUUrbanAcousticScenes2020Mobile(dataset_path)	TAU Urban Acoustic Scenes 2019 dataset.
TUTSoundEvents2017(dataset_path)	TUT Sound Events 2017 dataset.
FSDKaggle2018(dataset_path)	FSDKaggle2018 dataset.
MAVD(dataset_path)	MAVD-traffic dataset.

7.1.1 dcase_models.data.Dataset

class dcase_models.data.**Dataset**(*dataset_path*)

Bases: object

Abstract base class to load and manage DCASE datasets.

Descendants of this class are defined to manage specific DCASE databases (see UrbanSound8k, ESC50)

Parameters

dataset_path [str] Path to the dataset fold. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/UrbanSound8k).

Examples

To create a new dataset, it is necessary to define a class that inherits from Dataset. Then is required to define the build, generate_file_lists, get_annotations and download (if online available) methods.

```
>>> from dcase_models.util import list_wav_files
```

```
>>> class TestDataset(Dataset):
>>>     def __init__(self, dataset_path):
>>>         super().__init__(dataset_path)
```

```
>>>     def build(self):
>>>         self.audio_path = os.path.join(self.dataset_path, 'audio')
>>>         self.fold_list = ["train", "validate", "test"]
>>>         self.label_list = ["cat", "dog"]
```

```
>>>     def generate_file_lists(self):
>>>         for fold in self.fold_list:
>>>             audio_folder = os.path.join(self.audio_path, fold)
>>>             self.file_lists[fold] = list_wav_files(audio_folder)
```

```
>>>     def get_annotations(self, file_path, features):
>>>         y = np.zeros((len(features), len(self.label_list)))
>>>         class_ix = int(os.path.basename(file_path).split('-')[1])
>>>         y[:, class_ix] = 1
>>>         return y
```

```
>>>     def download(self, force_download=False):
>>>         zenodo_url = "https://zenodo.org/record/123456/files"
>>>         zenodo_files = ["TestData.tar.gz"]
>>>         downloaded = super().download(
>>>             zenodo_url, zenodo_files, force_download
>>>         )
>>>         if downloaded:
>>>             move_all_files_to_parent(self.dataset_path, "TestData")
>>>             self.set_as_downloaded()
```

Attributes

file_lists [dict] This dictionary stores the list of files for each fold. Dict of form: {fold_name : list_of_files}. e.g. {'fold1' : [file1, file2 . . .], 'fold2' : [file3, file4 . . .],}.

audio_path [str] Path to the audio folder, i.e {self.dataset_path}/audio. This attribute is defined in build()

fold_list [list] List of the pre-defined fold names. e.g. ['fold1', 'fold2', 'fold3', . . .]

label_list [list] List of class labels. e.g. ['dog', 'siren', . . .]

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download(zenodo_url, zenodo_files[, ...])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_path, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*zenodo_url*, *zenodo_files*, *force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_path*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.2 dcase_models.data.UrbanSound8k

class dcase_models.data.**UrbanSound8k**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

UrbanSound8k dataset.

This class inherits all functionality from Dataset and defines specific attributs and methods for UrbanSound8k.

Url: https://urbansounddataset.weebly.com/urbansound8k.html

J. Salamon, C. Jacoby, and J. P. Bello "A dataset and taxonomy for urban sound research," 22st ACM International Conference on Multimedia (ACM-MM'14) Orlando, FL, USA, November 2014

Parameters

dataset_path [str] Path to the dataset fold. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/UrbanSound8k).

Examples

To work with UrbanSound8k dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import UrbanSound8k
>>> dataset = UrbanSound8k('../datasets/UrbanSound8K')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.3 dcase_models.data.ESC50

class dcase_models.data.**ESC50**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

ESC-50 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for ESC-50.

Url: https://github.com/karolpiczak/ESC-50

K. J. Piczak "Esc: Dataset for environmental sound classification," Proceedings of the 23rd ACM international conference on Multimedia Brisbane, Australia, October, 2015.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/ESC50).

Examples

To work with ESC50 dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import ESC50
>>> dataset = ESC50('../datasets/ESC50')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset

Continued on next page

Table 4 – continued from previous page

build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
get_basename_wav(filename)	
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

get_basename_wav(*filename*)

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.4 dcase_models.data.ESC10

class dcase_models.data.**ESC10**(*dataset_path*)
Bases: dcase_models.data.datasets.ESC50

ESC-10 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for ESC-10.

ESC-10 is a subsampled version of ESC-50.

Url: https://github.com/karolpiczak/ESC-50

K. J. Piczak "Esc: Dataset for environmental sound classification," Proceedings of the 23rd ACM international conference on Multimedia Brisbane, Australia, October, 2015.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/ESC50).

Examples

To work with ESC10 dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import ESC10
>>> dataset = ESC10('../datasets/ESC50')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
get_basename_wav(filename)	
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path

and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

get_basename_wav(*filename*)

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.5 dcase_models.data.URBAN_SED

class dcase_models.data.**URBAN_SED**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

URBAN-SED dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for URBAN-SED.

Url: http://urbansed.weebly.com/

J. Salamon, D. MacConnell, M. Cartwright, P. Li, and J. P.Bello. "Scaper: A library for soundscape synthesis and augmentation". IEEE Workshop on Applications of Signal Processing to Audio and Acoustics New York, USA, October 2017.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/URBAN_SED).

Examples

To work with URBAN_SED dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import URBAN_SED
>>> dataset = URBAN_SED('../datasets/URBAN_SED')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)

Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()

Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)

Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)

Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()

Saves a download.txt file in dataset_path as a downloaded flag.

7.1.6 dcase_models.data.SONYC_UST

class dcase_models.data.**SONYC_UST**(*dataset_path*)

Bases: dcase_models.data.dataset_base.Dataset

SONYC-UST dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for SONYC-UST.

Version: 2.1.0

Url: https://zenodo.org/record/3693077

M. Cartwright, et al. "SONYC Urban Sound Tagging (SONYC-UST): A Multilabel Dataset from an Urban Acoustic Sensor Network". Proceedings of the Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE), 2019.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/SONYC_UST).

Examples

To work with SONYC_UST dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import SONYC_UST
>>> dataset = SONYC_UST('../datasets/SONYC_UST')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path

and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.7 dcase_models.data.TAUUrbanAcousticScenes2019

class dcase_models.data.**TAUUrbanAcousticScenes2019**(*dataset_path*)
Bases: dcase_models.data.datasets._TAUUrbanAcousticScenes

TAU Urban Acoustic Scenes 2019 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for TAU Urban Acoustic Scenes 2019.

Url: https://zenodo.org/record/2589280

A. Mesaros, T. Heittola, and T. Virtanen. "A multi-devicedataset for urban acoustic scene classification". Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE 2018). November 2018.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/TAUUrbanAcousticScenes2019).

Examples

To work with TAUUrbanAcousticScenes2019 dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import TAUUrbanAcousticScenes2019
>>> dataset = TAUUrbanAcousticScenes2019(
    '../datasets/TAUUrbanAcousticScenes2019')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.8 dcase_models.data.TAUUrbanAcousticScenes2020Mobile

class dcase_models.data.**TAUUrbanAcousticScenes2020Mobile**(*dataset_path*)
Bases: dcase_models.data.datasets._TAUUrbanAcousticScenes

TAU Urban Acoustic Scenes 2019 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for TAU Urban Acoustic Scenes 2020 Mobile.

Url: https://zenodo.org/record/3819968

T. Heittola, A. Mesaros, and T. Virtanen. “Acoustic sceneclassification in DCASE 2020 challenge: generalizationacross devices and low complexity solutions”. Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE 2020). 2020

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/TAUUrbanAcousticScenes2020Mobile).

Examples

To work with TAUUrbanAcousticScenes2020Mobile dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import TAUUrbanAcousticScenes2020Mobile
>>> dataset = TAUUrbanAcousticScenes2020Mobile(
    '../datasets/TAUUrbanAcousticScenes2020Mobile')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It’s mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)

Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.9 dcase_models.data.TUTSoundEvents2017

class dcase_models.data.**TUTSoundEvents2017**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

TUT Sound Events 2017 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for TUT Sound Events 2017.

Url: https://zenodo.org/record/814831

A. Mesaros et al. DCASE 2017 challenge setup: tasks, datasets and baseline system. Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017), 85–92. November 2017.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/TUTSoundEvents2017).

Examples

To work with TUTSoundEvents2017 dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import TUTSoundEvents2017
>>> dataset = TUTSoundEvents2017('../datasets/TUTSoundEvents2017')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.10 dcase_models.data.FSDKaggle2018

class dcase_models.data.**FSDKaggle2018**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

FSDKaggle2018 dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for FSDKaggle2018.

Url: https://zenodo.org/record/2552860

Eduardo Fonseca et al. "General-purpose Tagging of Freesound Audio with AudioSet Labels: Task Description, Dataset, and Baseline". Proceedings of the DCASE 2018 Workshop. 2018.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/FSDKaggle2018).

Examples

To work with FSDKaggle2018 dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import FSDKaggle2018
>>> dataset = FSDKaggle2018('../datasets/FSDKaggle2018')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.1.11 dcase_models.data.MAVD

class dcase_models.data.**MAVD**(*dataset_path*)
Bases: dcase_models.data.dataset_base.Dataset

MAVD-traffic dataset.

This class inherits all functionality from Dataset and defines specific attributes and methods for MAVD-traffic.

Url: https://zenodo.org/record/3338727

P. Zinemanas, P. Cancela, and M. Rocamora. "MAVD: a dataset for sound event detection in urban environments" Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE 2019). October, 2019.

Parameters

dataset_path [str] Path to the dataset folder. This is the path to the folder where the complete dataset will be downloaded, decompressed and handled. It is expected to use a folder name that represents the dataset unambiguously (e.g. ../datasets/MAVD).

Examples

To work with MAVD dataset, just initialize this class with the path to the dataset.

```
>>> from dcase_models.data.datasets import MAVD
>>> dataset = MAVD('../datasets/MAVD')
```

Then, you can download the dataset and change the sampling rate.

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

__init__(*dataset_path*)
Init Dataset

Methods

__init__(dataset_path)	Init Dataset
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download([force_download])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Creates file_lists, a dict that includes a list of files per fold.
get_annotations(file_name, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns paths to the audio folder.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Creates file_lists, a dict that includes a list of files per fold.

Each dataset has a different way of organizing the files. This function defines the dataset structure.

get_annotations(*file_name*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns paths to the audio folder.

If sr is None, return audio_path. Else, return {audio_path}{sr}.

Parameters

sr [int or None, optional] Sampling rate.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. Important when use AugmentedDataset. e.g. ['{DATASET_PATH}/audio/original']

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.2 Features

Features are implemented as specializations of the base class FeatureExtractor.

FeatureExtractor([sequence_time, ...])	Abstract base class for feature extraction.
Spectrogram([sequence_time, ...])	Spectrogram feature extractor.
MelSpectrogram([sequence_time, ...])	MelSpectrogram feature extractor.

Continued on next page

Table 13 – continued from previous page

Openl3([sequence_time, sequence_hop_time, ...])	Openl3 feature extractor.
RawAudio([sequence_time, sequence_hop_time, ...])	RawAudio feature extractor.
FramesAudio([sequence_time, ...])	FramesAudio feature extractor.

7.2.1 dcase_models.data.FeatureExtractor

class dcase_models.data.**FeatureExtractor**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, ***kwargs*)

Bases: object

Abstract base class for feature extraction.

Includes methods to load audio files, calculate features and prepare sequences.

Inherit this class to define custom features (e.g. features.MelSpectrogram, features.Openl3).

Parameters

sequence_time [float, default=1.0] Length (in seconds) of the feature representation analysis windows (model's input).

sequence_hop_time [float, default=0.5] Hop time (in seconds) of the feature representation analysis windows.

audio_win [int, default=1024] Window length (in samples) for the short-time audio processing (e.g short-time Fourier Transform (STFT))

audio_hop [int, default=680] Hop length (in samples) for the short-time audio processing (e.g short-time Fourier Transform (STFT))

sr [int, default=22050] Sampling rate of the audio signals. If the original audio is not sampled at this rate, it is re-sampled before feature extraction.

Examples

To create a new feature representation, it is necessary to define a class that inherits from FeatureExtractor. It is required to define the calculate() method.:

```
from dcase_models.data.feature_extractor import FeatureExtractor
class Chroma(FeatureExtractor):
    def __init__(self, sequence_time=1.0, sequence_hop_time=0.5,
                       audio_win=1024, audio_hop=512, sr=44100,
                       # Add here your custom parameters
                       n_fft=1024, n_chroma=12):
        # Don't forget this line
        super().__init__(sequence_time=sequence_time,
                         sequence_hop_time=sequence_hop_time,
                         audio_win=audio_win,
                         audio_hop=audio_hop, sr=sr)

        self.sequence_samples = int(librosa.core.frames_to_samples(
            self.sequence_frames,
            self.audio_hop,
            n_fft=self.n_fft
        ))
    def calculate(self, file_name):
```

(continues on next page)

(continued from previous page)

```
        # Here define your function to calculate the chroma features
        # Load the audio signal
        audio = self.load_audio(file_name)
        # Pad audio signal
        audio = librosa.util.fix_length(
            audio,
            audio.shape[0] + self.sequence_samples,
            axis=0, mode='constant'
        )
        # Get the chroma features
        chroma = librosa.feature.chroma_stft(y=audio,
                                             sr=self.sr,
                                             n_fft=self.n_fft,
                                             hop_length=audio_hop,
                                             win_length=audio_win
                                             )
        # Convert to sequences
        chroma = np.ascontiguousarray(chroma)
        chroma = librosa.util.frame(chroma,
                                    self.sequence_frames,
                                    self.sequence_hop,
                                    axis=0
                                    )
        return chroma
```

Attributes

sequence_frames [int] Number of frames equivalent to the sequence_time.

sequence_hop [int] Number of frames equivalent to the sequence_hop_time.

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, ***kwargs*)
Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)
Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)
Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)
Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)
Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)
Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)
Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)
Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.2.2 dcase_models.data.Spectrogram

class dcase_models.data.**Spectrogram**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *pad_mode='reflect'*)
Bases: dcase_models.data.feature_extractor.FeatureExtractor

Spectrogram feature extractor.

Extracts the log-scaled spectrogram of the audio signals. The spectrogram is calculated over the whole audio signal and then is separated in overlapped sequences (frames)

Parameters

n_fft [int, default=1024] Number of samples used for FFT calculation. Refer to librosa.core.stft for further information.

pad_mode [str or None, default='reflect'] Mode of padding applied to the audio signal. This argument is passed to librosa.util.fix_length for padding the signal. If pad_mode is None, no padding is applied.

See also:

FeatureExtractor FeatureExtractor base class.

MelSpectrogram MelSpectrogram feature extractor.

Notes

Based in librosa.core.stft function.

Examples

Extract features of a given file

```
>>> from dcase_models.data.features import Spectrogram
>>> from dcase_models.util.files import example_audio_file
>>> features = Spectrogram()
>>> features_shape = features.get_shape()
>>> print(features_shape)
    (21, 32, 513)
>>> file_name = example_audio_file()
>>> spectrogram = features.calculate(file_name)
>>> print(spectrogram.shape)
    (3, 32, 513)
```

Extract features for each file in a given dataset.

```
>>> from dcase_models.data.datasets import ESC50
>>> dataset = ESC50('../datasets/ESC50')
>>> features.extract(dataset)
```

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *pad_mode='reflect'*)
Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)
Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)

Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)
Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)
Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)
Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)
Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)
Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.2.3 dcase_models.data.MelSpectrogram

class dcase_models.data.**MelSpectrogram**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *mel_bands=64*, *pad_mode='reflect'*, ***kwargs*)

Bases: dcase_models.data.feature_extractor.FeatureExtractor

MelSpectrogram feature extractor.

Extracts the log-scaled mel-spectrogram of the audio signals. The mel-spectrogram is calculated over the whole audio signal and then is separated in overlapped sequences (frames).

Parameters

n_fft [int, default=1024] Number of samples used for FFT calculation. Refer to *librosa.core.stft* for further information.

mel_bands [int, default=64] Number of mel bands.

pad_mode [str or None, default='reflect'] Mode of padding applied to the audio signal. This argument is passed to librosa.util.fix_length for padding the signal. If pad_mode is None, no padding is applied.

kwargs Additional keyword arguments to *librosa.filters.mel*.

See also:

FeatureExtractor FeatureExtractor base class

Spectrogram Spectrogram features

Notes

Based in *librosa.core.stft* and *librosa.filters.mel* functions.

Examples

Extract features of a given file.

```
>>> from dcase_models.data.features import MelSpectrogram
>>> from dcase_models.util.files import example_audio_file
>>> features = MelSpectrogram()
>>> features_shape = features.get_shape()
```

(continues on next page)

(continued from previous page)

```
>>> print(features_shape)
    (21, 32, 64)
>>> file_name = example_audio_file()
>>> mel_spectrogram = features.calculate(file_name)
>>> print(mel_spectrogram.shape)
    (3, 32, 64)
```

Extract features for each file in a given dataset.

```
>>> from dcase_models.data.datasets import ESC50
>>> dataset = ESC50('../datasets/ESC50')
>>> features.extract(dataset)
```

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *mel_bands=64*, *pad_mode='reflect'*, ***kwargs*)
Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)
Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)
Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)
Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)
Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)
Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)
Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)
Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.2.4 dcase_models.data.Openl3

class dcase_models.data.**Openl3**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *content_type='env'*, *input_repr='mel256'*, *embedding_size=512*)
Bases: dcase_models.data.feature_extractor.FeatureExtractor

Openl3 feature extractor.

Based in openl3 library.

Parameters

content_type [{'music' or 'env'}, default='env'] Type of content used to train the embedding model. Refer to openl3.core.get_audio_embedding.

input_repr [{'linear', 'mel128', or 'mel256'}] Spectrogram representation used for model. Refer to openl3.core.get_audio_embedding.

embedding_size [{6144 or 512}, default=512] Embedding dimensionality. Refer to openl3.core.get_audio_embedding.

pad_mode [str or None, default='reflect'] Mode of padding applied to the audio signal. This argument is passed to librosa.util.fix_length for padding the signal. If pad_mode is None, no padding is applied.

See also:

FeatureExtractor FeatureExtractor base class

Spectrogram Spectrogram features

Examples

Extract features of a given file.

```
>>> from dcase_models.data.features import Openl3
>>> from dcase_models.util.files import example_audio_file
>>> features = Openl3()
>>> features_shape = features.get_shape()
>>> print(features_shape)
    (20, 512)
>>> file_name = example_audio_file()
>>> mel_spectrogram = features.calculate(file_name)
>>> print(mel_spectrogram.shape)
    (3, 512)
```

Extract features for each file in a given dataset.

```
>>> from dcase_models.data.datasets import ESC50
>>> dataset = ESC50('../datasets/ESC50')
>>> features.extract(dataset)
```

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *content_type='env'*, *input_repr='mel256'*, *embedding_size=512*)
Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)
Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)
Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)

Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)

Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)

Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)

Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)

Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.2.5 dcase_models.data.RawAudio

class dcase_models.data.**RawAudio**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *pad_mode='reflect'*)

Bases: dcase_models.data.feature_extractor.FeatureExtractor

RawAudio feature extractor.

Load the audio signal and create sequences (overlapped windows)

Parameters

pad_mode [str or None, default='reflect'] Mode of padding applied to the audio signal. This argument is passed to librosa.util.fix_length for padding the signal. If pad_mode is None, no padding is applied.

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *pad_mode='reflect'*)

Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)

Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)

Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)
Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)
Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)
Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)
Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)
Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.2.6 dcase_models.data.FramesAudio

class dcase_models.data.**FramesAudio**(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *pad_mode='reflect'*)
Bases: dcase_models.data.feature_extractor.FeatureExtractor

FramesAudio feature extractor.

Load the audio signal, convert it into time-short frames, and create sequences (overlapped windows).

Parameters

pad_mode [str or None, default='reflect'] Mode of padding applied to the audio signal. This argument is passed to librosa.util.fix_length for padding the signal. If pad_mode is None, no padding is applied.

__init__(*sequence_time=1.0*, *sequence_hop_time=0.5*, *audio_win=1024*, *audio_hop=680*, *sr=22050*, *n_fft=1024*, *pad_mode='reflect'*)
Initialize the FeatureExtractor

Methods

__init__([sequence_time, sequence_hop_time, ...])	Initialize the FeatureExtractor
calculate(file_name)	Loads an audio file and calculates features
check_if_extracted(dataset)	Checks if the features of each file in dataset was calculated.
check_if_extracted_path(path)	Checks if the features saved in path were calculated.
convert_to_sequences(audio_representation)	
extract(dataset)	Extracts features for each file in dataset.
get_features_path(dataset)	Returns the path to the features folder.
get_shape([length_sec])	Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.
load_audio(file_name[, mono, ...])	Loads an audio signal and converts it to mono if needed
pad_audio(audio)	
set_as_extracted(path)	Saves a json file with self.__dict__.

calculate(*file_name*)
Loads an audio file and calculates features

Parameters

file_name [str] Path to the audio file

Returns

ndarray feature representation of the audio signal

check_if_extracted(*dataset*)
Checks if the features of each file in dataset was calculated.

Calls check_if_extracted_path for each path in the dataset.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

check_if_extracted_path(*path*)
Checks if the features saved in path were calculated.

Compare if the features were calculated with the same parameters of self.__dict__.

Parameters

path [str] Path to the features folder

Returns

bool True if the features were already extracted.

convert_to_sequences(*audio_representation*)

extract(*dataset*)
Extracts features for each file in dataset.

Call calculate() for each file in dataset and save the result into the features path.

Parameters

dataset [Dataset] Instance of the dataset.

get_features_path(*dataset*)
Returns the path to the features folder.

Parameters

dataset [Dataset] Instance of the dataset.

Returns

features_path [str] Path to the features folder.

get_shape(*length_sec=10.0*)
Calls calculate() with a dummy signal of length length_sec and returns the shape of the feature representation.

Parameters

length_sec [float] Duration in seconds of the test signal

Returns

tuple Shape of the feature representation

load_audio(*file_name*, *mono=True*, *change_sampling_rate=True*)
Loads an audio signal and converts it to mono if needed

Parameters

file_name [str] Path to the audio file

mono [bool] if True, only returns left channel

change_sampling_rate [bool] if True, the audio signal is re-sampled to self.sr

Returns

array audio signal

pad_audio(*audio*)

set_as_extracted(*path*)
Saves a json file with self.__dict__.

Useful for checking if the features files were calculated with same parameters.

Parameters

path [str] Path to the JSON file

7.3 Augmentation

AugmentedDataset(dataset, sr, augmentations_list)	Class that manage data augmentation.
WhiteNoise(snr)	Implements white noise augmentation.

7.3.1 dcase_models.data.AugmentedDataset

class dcase_models.data.**AugmentedDataset**(*dataset*, *sr*, *augmentations_list*)
Bases: dcase_models.data.dataset_base.Dataset

Class that manage data augmentation.

Basically, it takes an instance of Dataset and generates an augmented one. Includes methods to generate data augmented versions of the audio files in an existing Dataset.

Parameters

dataset [Dataset] Instance of Dataset to be augmented.

augmentations_list [list] List of augmentation types and their parameters. Dict of form: [{'type' : aug_type, 'param1': param1 ...} ...]. e.g.:

```
[
    {'type': 'pitch_shift', 'n_semitones': -1},
    {'type': 'time_stretching', 'factor': 1.05}
]
```

sr [int] Sampling rate

Examples

Define an instance of UrbanSound8k and convert it into an augmented instance of the dataset. Note that the actual augmentation is performed when process() method is called.

```
>>> from dcase_models.data.datasets import UrbanSound8k
>>> from dcase_models.data.data_augmentation import AugmentedDataset
>>> dataset = UrbanSound8k('../datasets/UrbanSound8K')
>>> augmentations = [
```

(continues on next page)

(continued from previous page)

```
        {"type": "pitch_shift", "n_semitones": -1},
        {"type": "time_stretching", "factor": 1.05},
        {"type": "white_noise", "snr": 60}
    ]
>>> aug_dataset = AugmentedDataset(dataset, augmentations)
>>> aug_dataset.process()
```

__init__ (*dataset*, *sr*, *augmentations_list*)
Initialize the AugmentedDataset.

Initialize sox Transformers for each type of augmentation.

Methods

__init__(dataset, sr, augmentations_list)	Initialize the AugmentedDataset.
build()	Builds the dataset.
change_sampling_rate(new_sr)	Changes the sampling rate of each wav file in audio_path.
check_if_downloaded()	Checks if the dataset was downloaded.
check_sampling_rate(sr)	Checks if dataset was resampled before.
convert_to_wav([remove_original])	Converts each file in the dataset to wav format.
download(zenodo_url, zenodo_files[, ...])	Downloads and decompresses the dataset from zenodo.
generate_file_lists()	Create self.file_lists, a dict that includes a list of files per fold.
get_annotations(file_path, features, ...)	Returns the annotations of the file in file_path.
get_audio_paths([sr])	Returns a list of paths to the folders that include the dataset augmented files.
process()	Generate augmentated data for each file in dataset.
set_as_downloaded()	Saves a download.txt file in dataset_path as a downloaded flag.

build()
Builds the dataset.

Define specific attributes of the dataset. It's mandatory to define audio_path, fold_list and label_list. Other attributes may be defined here (url, authors, etc.).

change_sampling_rate(*new_sr*)
Changes the sampling rate of each wav file in audio_path.

Creates a new folder named audio_path{new_sr} (i.e audio22050) and converts each wav file in audio_path and save the result in the new folder.

Parameters

sr [int] Sampling rate.

check_if_downloaded()
Checks if the dataset was downloaded.

Just checks if exists download.txt file.

Further checks in the future.

check_sampling_rate(*sr*)
Checks if dataset was resampled before.

For now, only checks if the folder {audio_path}{sr} exists and each wav file present in audio_path is also present in {audio_path}{sr}.

Parameters

sr [int] Sampling rate.

Returns

bool True if the dataset was resampled before.

convert_to_wav(*remove_original=False*)
Converts each file in the dataset to wav format.

If remove_original is False, the original files will be deleted

Parameters

remove_original [bool] Remove original files.

download(*zenodo_url*, *zenodo_files*, *force_download=False*)
Downloads and decompresses the dataset from zenodo.

Parameters

zenodo_url [str] URL with the zenodo files. e.g. 'https://zenodo.org/record/12345/files'

zenodo_files [list of str] List of files. e.g. ['file1.tar.gz', 'file2.tar.gz', 'file3.tar.gz']

force_download [bool] If True, download the dataset even if was downloaded before.

Returns

bool True if the downloading process was successful.

generate_file_lists()
Create self.file_lists, a dict that includes a list of files per fold.

Just call dataset.generate_file_lists() and copy the attribute.

get_annotations(*file_path*, *features*, *time_resolution*)
Returns the annotations of the file in file_path.

Parameters

file_path [str] Path to the file

features [ndarray] nD array with the features of file_path

time_resolution [float] Time resolution of the features

Returns

ndarray Annotations of the file file_path Expected output shape: (features.shape[0], len(self.label_list))

get_audio_paths(*sr=None*)
Returns a list of paths to the folders that include the dataset augmented files.

The folder of each augmentation is defined using its name and parameter values.

e.g. {DATASET_PATH}/audio/pitch_shift_1 where 1 is the 'n_semitones' parameter.

Parameters

sr [int or None, optional] Sampling rate (optional). We keep this parameter to keep compatibility with Dataset.get_audio_paths() method.

Returns

audio_path [str] Path to the root audio folder. e.g. DATASET_PATH/audio

subfolders [list of str] List of subfolders include in audio folder. e.g.:

```
[
    '{DATASET_PATH}/audio/original',
    '{DATASET_PATH}/audio/pitch_shift_1',
    '{DATASET_PATH}/audio/time_stretching_1.1',
]
```

process()
Generate augmentated data for each file in dataset.

Replicate the folder structure of {DATASET_PATH}/audio/original into the folder of each augmentation folder.

set_as_downloaded()
Saves a download.txt file in dataset_path as a downloaded flag.

7.3.2 dcase_models.data.WhiteNoise

class dcase_models.data.**WhiteNoise**(*snr*)
Bases: object

Implements white noise augmentation.

The structure is similar to sox.Transformer in order to keep compatibility with sox.

Parameters

snr [float] Signal to noise ratio.

__init__(*snr*)
Initialize the white noise.

Methods

__init__(snr)	Initialize the white noise.
build(file_origin, file_destination)	Add noise to the file_origin and save the result in file_destination.

build(*file_origin*, *file_destination*)
Add noise to the file_origin and save the result in file_destination.

Parameters

file_origin [str] Path to the source file.

file_destination [str] Path to the destination file.

7.4 DataGenerator

DataGenerator(dataset, inputs, folds[, ...])	Includes methods to load features files from DCASE datasets.
KerasDataGenerator(data_generator)	

7.4.1 dcase_models.data.DataGenerator

class dcase_models.data.**DataGenerator**(*dataset*, *inputs*, *folds*, *outputs='annotations'*, *batch_size=32*, *shuffle=True*, *train=True*, *scaler=None*, *scaler_outputs=None*)

Bases: object

Includes methods to load features files from DCASE datasets.

Parameters

dataset [Dataset] Instance of the Dataset used to load the data. Note that the dataset has to be downloaded before initializing the DataGenerator. Refer to dcase-models/data/datasets.py for a complete list of available datasets.

inputs [instance of FeatureExtractor or list of FeatureExtractor instances] Instance(s) of FeatureExtractor. These are the feature extractor(s) used to generate the features. For multi-input, pass a list of FeatureExtractor instances.

folds [list of str] List of folds to be loaded. Each fold has to be in dataset.fold_list. Note that since the folds used at each stage of the pipeline (training, validation, evaluation) are different, an instance of DataGenerator for each stage has to be created. e.g. ['fold1', 'fold2', 'fold3', ...]

outputs [str, FeatureExtractor or list, default='annotations'] Instance(s) of FeatureExtractor used to generate the outputs. To use the annotations obtained from Dataset, use a string. For multi-output, use a list of FeatureExtractor and/or strings.

batch_size [int, default=32] Number of files loaded when call get_data_batch(). Note that the meaning of batch_size here is slightly different from the one in machine learning libraries like keras. In these libraries batch_size means the number of instances (sequences in DCASE-models) used in each training step. Here batch_size is the number of files, and therefore, the number of sequences varies in each batch.

shuffle: bool, default=True When training a model, it is typical to shuffle the dataset at the end of each epoch. If shuffle is True (default), then the audio file list is shuffled when the class is initialized and when shuffle_list() method is called.

train [bool, default True] When training, it is typical to feed the model with a numpy array that contains all the data concatenated. For validation and testing it is necessary to have the features of each file separate in order to do a file-wise evaluation. Therefore, if train is True, the loaded data is concatenated and converted to a numpy array. If train is False get_data() and get_data_batch() return a list, whose elements are the features of each file in the audio_file_list.

scaler [Scaler or None, default=None] If is not None, the Scaler object is used to scale the data after loading.

scaler_outputs [Scaler or None, default=None] Same as scaler but for the system outputs.

See also:

Dataset Dataset class

FeatureExtractor FeatureExtractor class

Examples

Create instances of Dataset and FeatureExtractor with default parameters

```
>>> from dcase_models.data.datasets import UrbanSound8k
>>> from dcase_models.data.features import MelSpectrogram
>>> from dcase_models.data.data_generator import DataGenerator
>>> dataset = UrbanSound8k('../datasets/UrbanSound8k')
>>> features = MelSpectrogram()
```

Assuming that the dataset was downloaded and features were extracted already, we can initialize the data generators. This example uses fold1 and fold2 for training and fold3 for validation.

```
>>> data_gen_train = DataGenerator(
    dataset, features, ['fold1', 'fold2'], train=True)
>>> data_gen_val = DataGenerator(
    dataset, features, ['fold3'], train=False)
```

```
>>> X_train, Y_train = data_gen_train.get_data_batch(0)
>>> print(X_train.shape, Y_train.shape)
    (212, 43, 64) (212, 10)
```

```
>>> X_val, Y_val = data_gen_val.get_data_batch(0)
>>> print(len(X_val), len(Y_val))
    32 32
>>> print(X_val[0].shape, Y_val[0].shape)
    (7, 43, 64) (7, 10)
```

```
>>> X_train, Y_train = data_gen_train.get_data()
>>> print(X_train.shape, Y_train.shape)
    (11095, 43, 64) (11095, 10)
```

```
>>> X_val, Y_val = data_gen_val.get_data()
>>> print(len(X_val), len(Y_val))
    925 925
>>> print(X_val[0].shape, Y_val[0].shape)
    (7, 43, 64) (7, 10)
```

Attributes

audio_file_list [list of dict] List of audio files from which the features will be loaded. Each element in the list includes information of the original audio file (important to get the annotations) and the subfolder where is the resampled (and maybe augmented) audio file. e.g.:

audio_file_list = [{'file_original': 'audio/1.wav', 'sub_folder': 'original'}, {'file_original': 'audio/1.wav', 'sub_folder': 'pitch_shift_1'}, {'file_original': 'audio/2.wav', 'sub_folder': 'original'}, ...

]

__init__(*dataset*, *inputs*, *folds*, *outputs='annotations'*, *batch_size=32*, *shuffle=True*, *train=True*, *scaler=None*, *scaler_outputs=None*)

Initialize the DataGenerator.

Generates the audio_file_list by concatenating all the files from the folds passed as an argument.

Methods

__init__(dataset, inputs, folds[, outputs, ...])	Initialize the DataGenerator.
convert_audio_path_to_features_path(..., ...])	Converts audio path(s) to features path(s).
convert_features_path_to_audio_path(..., sr])	Converts features path(s) to audio path(s).
get_data()	Return all data from the selected folds.
get_data_batch(index)	Return the data from the batch given by argument.
get_data_from_file(file_index)	Returns the data from the file index given by argument.
paths_remove_aug_subfolder(path)	Removes the subfolder string related to augmentation from a path.
set_scaler(scaler)	Set scaler object.
set_scaler_outputs(scaler_outputs)	Set scaler object.
shuffle_list()	Shuffles features_file_list.

convert_audio_path_to_features_path(*audio_file*, *features_path*, *subfolder=''*)
Converts audio path(s) to features path(s).

Parameters

audio_file [str or list of str] Path(s) to the audio file(s).

Returns

features_file [str or list of str] Path(s) to the features file(s).

convert_features_path_to_audio_path(*features_file*, *features_path*, *sr=None*)
Converts features path(s) to audio path(s).

Parameters

features_file [str or list of str] Path(s) to the features file(s).

Returns

audio_file [str or list of str] Path(s) to the audio file(s).

get_data()
Return all data from the selected folds.

If train were set as True, the output is concatenated and converted to a numpy array. Otherwise the outputs are lists whose elements are the features of each file.

Returns

X [list or ndarray] List or array of features for each file.

Y [list or ndarray] List or array of annotations for each file.

get_data_batch(*index*)
Return the data from the batch given by argument.

If train were set as True, the output is concatenated and converted to a numpy array. Otherwise the outputs are lists whose elements are the features of each file.

Returns

X [list or ndarray] List or array of features for each file.

Y [list or ndarray] List or array of annotations for each file.

get_data_from_file(*file_index*)
Returns the data from the file index given by argument.

Returns

X [ndarray] Array of features for each file.

Y [ndarray] Array of annotations for each file.

paths_remove_aug_subfolder(*path*)
Removes the subfolder string related to augmentation from a path.

Converts DATASET_PATH/audio/original/... into DATASET_PATH/audio/...

Parameters

path [str or list of str] Path to be converted.

Returns

features_file [str or list of str] Path(s) to the features file(s).

set_scaler(*scaler*)
Set scaler object.

set_scaler_outputs(*scaler_outputs*)
Set scaler object.

shuffle_list()
Shuffles features_file_list.

Notes

Only shuffle the list if shuffle is True.

7.4.2 dcase_models.data.KerasDataGenerator

class dcase_models.data.**KerasDataGenerator**(*data_generator*)
Bases: tensorflow.python.keras.utils.data_utils.Sequence

__init__(*data_generator*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(data_generator)	Initialize self.
on_epoch_end()	Updates indexes after each epoch

on_epoch_end()
Updates indexes after each epoch

7.5 Scaler

Scaler([normalizer])	Scaler object to normalize or scale the data.

7.5.1 dcase_models.data.Scaler

class dcase_models.data.**Scaler**(*normalizer='standard'*)

Bases: object

Scaler object to normalize or scale the data.

Parameters

normalizer [{'standard' or 'minmax'}, default='standard'] Type of normalizer.

See also:

DataGenerator data generator class

Examples

```
>>> from dcase_models.data.scaler import Scaler
>>> import numpy as np
>>> scaler = Scaler('minmax')
>>> X = 3 * np.random.rand(10, 150)
>>> print(np.amin(X), np.amax(X))
```

```
>>> scaler.fit(X)
>>> X = scaler.transform(X)
>>> print(np.amin(X), np.amax(X))
```

Attributes

scaler [sklearn.preprocessing.StandardScaler or list] Scaler object for standard normalizer or list for minmax scaler.

__init__(*normalizer='standard'*)

Initialize the Scaler.

If normalizer is 'standard', initialize the sklearn object.

Methods

__init__([normalizer])	Initialize the Scaler.
fit(X[, inputs])	Fit the Scaler.
inverse_transform(X)	Invert transformation.
partial_fit(X)	Fit the Scaler in one batch.
transform(X)	Scale X using the scaler.

fit(*X*, *inputs=True*)

Fit the Scaler.

Parameters

X [ndarray or DataGenerator] Data to be used in the fitting process.

inverse_transform(*X*)
Invert transformation.

Parameters

X [ndarray] Data to be scaled.

Returns

ndarray Scaled data. The shape of the output is the same of the input.

partial_fit(*X*)
Fit the Scaler in one batch.

Parameters

X [ndarray] Data to be used in the fitting process.

transform(*X*)
Scale X using the scaler.

Parameters

X [ndarray] Data to be scaled.

Returns

ndarray Scaled data. The shape of the output is the same of the input.

CHAPTER 8

Models

8.1 ModelContainer

A ModelContainer defines an interface to standardize the behavior of machine learning models. It stores the architecture and the parameters of the model. It provides methods to train and evaluate the model, and to save and load its architecture and weights.

ModelContainer([model, model_path, ...])	Abstract base class to store and manage models.
KerasModelContainer([model, model_path, ...])	ModelContainer for keras models.

8.1.1 dcase_models.model.ModelContainer

class dcase_models.model.**ModelContainer**(*model=None, model_path=None, model_name='ModelContainer', metrics=['classification']*)

Bases: object

Abstract base class to store and manage models.

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

__init__(*model=None, model_path=None, model_name='ModelContainer', metrics=['classification']*)

Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

__init__([model, model_path, model_name, ...])	Initialize ModelContainer
build()	Missing docstring here
check_if_model_exists(folder, **kwargs)	Missing docstring here
evaluate(X_test, Y_test[, scaler])	Missing docstring here
get_available_intermediate_outputs()	Missing docstring here
get_intermediate_output(output_ix_name)	Missing docstring here
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Missing docstring here
load_model_weights(weights_folder)	Missing docstring here
save_model_json(folder)	Missing docstring here
save_model_weights(weights_folder)	Missing docstring here
train()	Missing docstring here

build()
Missing docstring here

check_if_model_exists(*folder*, ***kwargs*)
Missing docstring here

evaluate(*X_test*, *Y_test*, *scaler=None*)
Missing docstring here

get_available_intermediate_outputs()
Missing docstring here

get_intermediate_output(*output_ix_name*)
Missing docstring here

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Missing docstring here

load_model_weights(*weights_folder*)
Missing docstring here

save_model_json(*folder*)
Missing docstring here

save_model_weights(*weights_folder*)
Missing docstring here

train()
Missing docstring here

8.1.2 dcase_models.model.KerasModelContainer

class dcase_models.model.**KerasModelContainer**(*model=None, model_path=None, model_name='DCASEModelContainer', metrics=['classification'], **kwargs*)

Bases: dcase_models.model.container.ModelContainer

ModelContainer for keras models.

A class that contains a keras model, the methods to train, evaluate, save and load the model. Descendants of this class can be specialized for specific models (i.e see SB_CNN class)

Parameters

model [keras.models.Model or None, default=None] If model is None the model is created with *build()*.

model_path [str or None, default=None] Path to the model. If it is not None, the model loaded from this path.

model_name [str, default=DCASEModelContainer] Model name.

metrics [list of str, default=['classification']] List of metrics used for evaluation. See *dcase_models.utils.metrics*.

kwargs Additional keyword arguments to *load_model_from_json()*.

__init__(*model=None, model_path=None, model_name='DCASEModelContainer', metrics=['classification'], **kwargs*)

Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

__init__([model, model_path, model_name, ...])	Initialize ModelContainer
build()	Define your model here
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.

Continued on next page

Table 3 – continued from previous page

load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()
Define your model here

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)

Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2 Implemented models

Each implemented model has its own class that inherits from a specific ModelContainer, such as KerasModelContainer.

MLP([model, model_path, metrics, n_classes, ...])	KerasModelContainer for a generic MLP model.
SB_CNN([model, model_path, metrics, ...])	KerasModelContainer for SB_CNN model.
SB_CNN_SED([model, model_path, metrics, ...])	KerasModelContainer for SB_CNN_SED model.
A_CRNN([model, model_path, metrics, ...])	KerasModelContainer for A_CRNN model.
VGGish([model, model_path, metrics, ...])	KerasModelContainer for VGGish model
SMel([model, model_path, metrics, ...])	KerasModelContainer for SMel model.
MST([model, model_path, metrics, mel_bands, ...])	KerasModelContainer for MST model.

8.2.1 dcase_models.model.MLP

class dcase_models.model.**MLP**(*model=None, model_path=None, metrics=['classification'], n_classes=10, n_frames=64, n_freqs=12, hidden_layers_size=[128, 64], dropout_rates=[0.5, 0.5], hidden_activation='relu', l2_reg=1e-05, final_activation='softmax', temporal_integration='mean', **kwargs*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for a generic MLP model.

Parameters

n_classes [int, default=10] Number of classes (dimmension output).

n_frames [int or None, default=64] Length of the input (number of frames of each sequence). Use None to not use frame-level input and output. In this case the input has shape (None, n_freqs).

n_freqs [int, default=12] Number of frequency bins. The model's input has shape (n_frames, n_freqs).

hidden_layers_size [list of int, default=[128, 64]] Dimmension of each hidden layer. Note that the length of this list defines the number of hidden layers.

dropout_rates [list of float, default=[0.5, 0.5]] List of dropout rate use after each hidden layer. The length of this list must be equal to the length of hidden_layers_size. Use 0.0 (or negative) to not use dropout.

hidden_activation [str, default='relu'] Activation for hidden layers.

l2_reg [float, default=1e-5] Weight of the l2 regularizers. Use 0.0 to not use regularization.

final_activation [str, default='softmax'] Activation of the last layer.

temporal_integration [{'mean', 'sum', 'autopool'}, default='mean'] Temporal integration operation used after last layer.

kwargs Additional keyword arguments to *Dense layers*.

Examples

```
>>> from dcase_models.model.models import MLP
>>> model_container = MLP()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 64, 12)            0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 64, 128)           1664
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 128)           0
_________________________________________________________________
time_distributed_2 (TimeDist (None, 64, 64)            8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 64)            0
_________________________________________________________________
time_distributed_3 (TimeDist (None, 64, 10)            650
_________________________________________________________________
temporal_integration (Lambda (None, 10)                0
=================================================================
Total params: 10,570
Trainable params: 10,570
Non-trainable params: 0
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['classification'], n_classes=10, n_frames=64, n_freqs=12, hidden_layers_size=[128, 64], dropout_rates=[0.5, 0.5], hidden_activation='relu', l2_reg=1e-05, final_activation='softmax', temporal_integration='mean', **kwargs*)

Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

__init__([model, model_path, metrics, ...])	Initialize ModelContainer
build()	Missing docstring here
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Load pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()

Missing docstring here

check_if_model_exists(*folder*, ***kwargs*)

Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)

Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)

Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)

Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()

Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)

Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.2 dcase_models.model.SB_CNN

class dcase_models.model.**SB_CNN**(*model=None, model_path=None, metrics=['classification'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, filter_size_cnn=(5, 5), pool_size_cnn=(2, 2), n_dense_cnn=64, n_channels=0*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for SB_CNN model.

J. Salamon and J. P. Bello. "Deep Convolutional Neural Networks and Data Augmentation For Environmental Sound Classification". IEEE Signal Processing Letters, 24(3), pages 279 - 283. 2017.

Parameters

n_classes [int, default=10] Number of classes (dimmension output).

n_frames_cnn [int or None, default=64] Length of the input (number of frames of each sequence).

n_freq_cnn [int, default=128] Number of frequency bins. The model's input has shape (n_frames, n_freqs).

filter_size_cnn [tuple, default=(5,5)] Kernel dimmension for convolutional layers.

pool_size_cnn [tuple, default=(2,2)] Pooling dimmension for maxpooling layers.

n_dense_cnn [int, default=64] Dimmension of penultimate dense layer.

n_channels [int, default=0] Number of input channels

0 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn)

1 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn, 1)

2 [stereo signals.] Input shape = (n_frames_cnn, n_freq_cnn, 2)

n > 2 [multi-representations.] Input shape = (n_frames_cnn, n_freq_cnn, n_channels)

Notes

Code based on Salamon's implementation https://github.com/justinsalamon/scaper_waspaa2017

Examples

```
>>> from dcase_models.model.models import SB_CNN
>>> model_container = SB_CNN()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 64, 128)           0
```

(continues on next page)

(continued from previous page)

```
_________________________________________________________________
lambda (Lambda)              (None, 64, 128, 1)        0
_________________________________________________________________
conv1 (Conv2D)               (None, 60, 124, 24)       624
_________________________________________________________________
maxpool1 (MaxPooling2D)      (None, 30, 62, 24)        0
_________________________________________________________________
batchnorm1 (BatchNormalizati (None, 30, 62, 24)        96
_________________________________________________________________
conv2 (Conv2D)               (None, 26, 58, 48)        28848
_________________________________________________________________
maxpool2 (MaxPooling2D)      (None, 6, 29, 48)         0
_________________________________________________________________
batchnorm2 (BatchNormalizati (None, 6, 29, 48)         192
_________________________________________________________________
conv3 (Conv2D)               (None, 2, 25, 48)         57648
_________________________________________________________________
batchnorm3 (BatchNormalizati (None, 2, 25, 48)         192
_________________________________________________________________
flatten (Flatten)            (None, 2400)              0
_________________________________________________________________
dropout1 (Dropout)           (None, 2400)              0
_________________________________________________________________
dense1 (Dense)               (None, 64)                153664
_________________________________________________________________
dropout2 (Dropout)           (None, 64)                0
_________________________________________________________________
out (Dense)                  (None, 10)                650
=================================================================
Total params: 241,914
Trainable params: 241,674
Non-trainable params: 240
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['classification'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, filter_size_cnn=(5, 5), pool_size_cnn=(2, 2), n_dense_cnn=64, n_channels=0*)
Initialization of the SB-CNN model.

Methods

__init__([model, model_path, metrics, ...])	Initialization of the SB-CNN model.
build()	Builds the CNN Keras model according to the initialized parameters.
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.

Continued on next page

Table 6 – continued from previous page

get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
sub_model()	Missing docstring here
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()
Builds the CNN Keras model according to the initialized parameters.

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)

Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()

Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)

Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()

Missing docstring here

load_model_from_json(*folder*, ***kwargs*)

Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)

Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)

Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)

Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

sub_model()
Missing docstring here

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.3 dcase_models.model.SB_CNN_SED

class dcase_models.model.**SB_CNN_SED**(*model=None, model_path=None, metrics=['sed'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, filter_size_cnn=(5, 5), pool_size_cnn=(2, 2), large_cnn=False, n_dense_cnn=64, n_filters_cnn=64, n_chanels=0*)
Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for SB_CNN_SED model.

J. Salamon, D. MacConnell, M. Cartwright, P. Li, and J. P. Bello. "Scaper: A Library for Soundscape Synthesis and Augmentation". IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA). New Paltz, NY, USA, Oct. 2017

Parameters

n_classes [int, default=10] Number of classes (dimmension output).

n_frames_cnn [int or None, default=64] Length of the input (number of frames of each sequence).

n_freq_cnn [int, default=128] Number of frequency bins. The model's input has shape (n_frames, n_freqs).

filter_size_cnn [tuple, default=(5,5)] Kernel dimmension for convolutional layers.

pool_size_cnn [tuple, default=(2,2)] Pooling dimmension for maxpooling layers.

large_cnn [bool, default=False] If large_cnn is true, add other dense layer after penultimate layer.

n_dense_cnn [int, default=64] Dimmension of penultimate dense layer.

n_channels [int, default=0] Number of input channels.

0 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn)

1 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn, 1)

2 [stereo signals.] Input shape = (n_frames_cnn, n_freq_cnn, 2)

n > 2 [multi-representations.] Input shape = (n_frames_cnn, n_freq_cnn, n_channels)

Notes

Code based on Salamon's implementation https://github.com/justinsalamon/scaper_waspaa2017

Examples

```
>>> from dcase_models.model.models import SB_CNN_SED
>>> model_container = SB_CNN_SED()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 128)           0
_________________________________________________________________
lambda_1 (Lambda)            (None, 64, 128, 1)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 124, 64)       1664
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 62, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 62, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 58, 64)        102464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 29, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 29, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 25, 64)         102464
_________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
batch_normalization_3 (Batch (None, 9, 25, 64)         256
_________________________________________________________________
flatten_1 (Flatten)          (None, 14400)             0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14400)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                921664
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 1,129,674
Trainable params: 1,129,290
Non-trainable params: 384
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['sed'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, filter_size_cnn=(5, 5), pool_size_cnn=(2, 2), large_cnn=False, n_dense_cnn=64, n_filters_cnn=64, n_chanels=0*)
Initialization of the SB-CNN-SED model.

Methods

__init__([model, model_path, metrics, ...])	Initialization of the SB-CNN-SED model.
build()	Missing docstring here
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()
Missing docstring here

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.4 dcase_models.model.A_CRNN

class dcase_models.model.**A_CRNN**(*model=None, model_path=None, metrics=['sed'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, cnn_nb_filt=128, cnn_pool_size=[5, 2, 2], rnn_nb=[32, 32], fc_nb=[32], dropout_rate=0.5, n_channels=0, final_activation='softmax', sed=False, bidirectional=False*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for A_CRNN model.

S. Adavanne, P. Pertilä, T. Virtanen "Sound event detection using spatial features and convolutional recurrent neural network" International Conference on Acoustics, Speech, and Signal Processing. 2017. https://arxiv.org/pdf/1706.02291.pdf

Parameters

n_classes [int, default=10] Number of classes (dimmension output).

n_frames_cnn [int or None, default=64] Length of the input (number of frames of each sequence).

n_freq_cnn [int, default=128] Number of frequency bins. The model's input has shape (n_frames, n_freqs).

cnn_nb_filt [int, default=128] Number of filters used in convolutional layers.

cnn_pool_size [tuple, default=(5, 2, 2)] Pooling dimmension for maxpooling layers.

rnn_nb [list, default=[32, 32]] Number of units in each recursive layer.

fc_nb [list, default=[32]] Number of units in each dense layer.

dropout_rate [float, default=0.5] Dropout rate.

n_channels [int, default=0] Number of input channels

0 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn)

1 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn, 1)

2 [stereo signals.] Input shape = (n_frames_cnn, n_freq_cnn, 2)

n > 2 [multi-representations.] Input shape = (n_frames_cnn, n_freq_cnn, n_channels)

final_activation [str, default='softmax'] Activation of the last layer.

sed [bool, default=False] If sed is True, the output is frame-level. If False the output is time averaged.

bidirectional [bool, default=False] If bidirectional is True, the recursive layers are bidirectional.

Notes

Code based on Adavanne's implementation https://github.com/sharathadavanne/sed-crnn

Examples

```
>>> from dcase_models.model.models import A_CRNN
>>> model_container = A_CRNN()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 64, 128)           0
_________________________________________________________________
lambda (Lambda)              (None, 64, 128, 1)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 128, 128)      1280
_________________________________________________________________
batch_normalization_7 (Batch (None, 64, 128, 128)      512
_________________________________________________________________
activation_4 (Activation)    (None, 64, 128, 128)      0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 64, 25, 128)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 64, 25, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 25, 128)       147584
_________________________________________________________________
batch_normalization_8 (Batch (None, 64, 25, 128)       100
_________________________________________________________________
activation_5 (Activation)    (None, 64, 25, 128)       0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 64, 12, 128)       0
_________________________________________________________________
dropout_10 (Dropout)         (None, 64, 12, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 64, 12, 128)       147584
_________________________________________________________________
batch_normalization_9 (Batch (None, 64, 12, 128)       48
_________________________________________________________________
activation_6 (Activation)    (None, 64, 12, 128)       0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 64, 6, 128)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 64, 6, 128)        0
_________________________________________________________________
reshape_2 (Reshape)          (None, 64, 768)           0
_________________________________________________________________
gru_3 (GRU)                  (None, 64, 32)            76896
_________________________________________________________________
```

(continues on next page)

(continued from previous page)

```
gru_4 (GRU)                  (None, 64, 32)            6240
_________________________________________________________________
time_distributed_6 (TimeDist (None, 64, 32)            1056
_________________________________________________________________
dropout_12 (Dropout)         (None, 64, 32)            0
_________________________________________________________________
time_distributed_7 (TimeDist (None, 64, 10)            330
_________________________________________________________________
mean (Lambda)                (None, 10)                0
_________________________________________________________________
strong_out (Activation)      (None, 10)                0
=================================================================
Total params: 381,630
Trainable params: 381,300
Non-trainable params: 330
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['sed'], n_classes=10, n_frames_cnn=64, n_freq_cnn=128, cnn_nb_filt=128, cnn_pool_size=[5, 2, 2], rnn_nb=[32, 32], fc_nb=[32], dropout_rate=0.5, n_channels=0, final_activation='softmax', sed=False, bidirectional=False*)

Methods

__init__([model, model_path, metrics, ...])	
build()	Builds the CRNN Keras model.
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()

Builds the CRNN Keras model.

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.5 dcase_models.model.VGGish

class dcase_models.model.**VGGish**(*model=None, model_path=None, metrics=['classification'], n_frames_cnn=96, n_freq_cnn=64, n_classes=10, n_channels=0, embedding_size=128, pooling='avg', include_top=False, compress=False*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for VGGish model

Jort F. Gemmeke et al. Audio Set: An ontology and human-labeled dataset for audio events International Conference on Acoustics, Speech, and Signal Processing. New Orleans, LA, 2017.

Parameters

n_frames_cnn [int or None, default=96] Length of the input (number of frames of each sequence).

n_freq_cnn [int, default=64] Number of frequency bins. The model's input has shape (n_frames, n_freqs).

n_classes [int, default=10] Number of classes (dimmension output).

n_channels [int, default=0] Number of input channels

0 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn)

1 [mono signals.] Input shape = (n_frames_cnn, n_freq_cnn, 1)

2 [stereo signals.] Input shape = (n_frames_cnn, n_freq_cnn, 2)

n > 2 [multi-representations.] Input shape = (n_frames_cnn, n_freq_cnn, n_channels)

embedding_size [int, default=128] Number of units in the embeddings layer.

pooling [{'avg', max}, default='avg'] Use AveragePooling or Maxpooling.

include_top [bool, default=False] Include fully-connected layers.

compress [bool, default=False] Apply PCA.

Notes

https://research.google.com/audioset/ Based on vggish-keras https://pypi.org/project/vggish-keras/

Examples

```
>>> from dcase_models.model.models import VGGish
>>> model_container = VGGish()
>>> model_container.model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 96, 64)            0
_________________________________________________________________
lambda (Lambda)              (None, 96, 64, 1)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 96, 64, 64)        640
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 48, 32, 64)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 48, 32, 128)       73856
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 24, 16, 128)       0
_________________________________________________________________
conv3/conv3_1 (Conv2D)       (None, 24, 16, 256)       295168
_________________________________________________________________
conv3/conv3_2 (Conv2D)       (None, 24, 16, 256)       590080
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 12, 8, 256)        0
_________________________________________________________________
conv4/conv4_1 (Conv2D)       (None, 12, 8, 512)        1180160
_________________________________________________________________
conv4/conv4_2 (Conv2D)       (None, 12, 8, 512)        2359808
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 6, 4, 512)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
=================================================================
Total params: 4,499,712
Trainable params: 4,499,712
Non-trainable params: 0
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['classification'], n_frames_cnn=96, n_freq_cnn=64, n_classes=10, n_channels=0, embedding_size=128, pooling='avg', include_top=False, compress=False*)

Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

`__init__`([model, model_path, metrics, ...])	Initialize ModelContainer
`build`()	Builds the VGGish Keras model.
`check_if_model_exists`(folder, **kwargs)	Checks if the model already exits in the path.
`cut_network`(layer_where_to_cut)	Cuts the network at the layer passed as argument.
`download_pretrained_weights`([weights_folder])	Download pretrained weights from: https://github.com/DTaoo/VGGish https://drive.google.com/file/d/1mhqXZ8CANgHyepum7N4yrjiyIg6qaMe6/view
`evaluate`(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
`fine_tuning`(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
`get_available_intermediate_outputs`()	Return a list of available intermediate outputs.
`get_intermediate_output`(output_ix_name, inputs)	Return the output of the model in a given layer.
`get_number_of_parameters`()	Missing docstring here
`load_model_from_json`(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
`load_model_weights`(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
`load_pretrained_model_weights`([weights_folder])	Load pretrained weights to self.model weights.
`save_model_json`(folder)	Saves the model to a model.json file in the given folder path.
`save_model_weights`(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
`train`(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

class Postprocess(*output_shape=None*, ***kw*)

Bases: `tensorflow.python.keras.engine.base_layer.Layer`

Keras layer that applies PCA and quantizes the ouput.

Based on vggish-keras https://pypi.org/project/vggish-keras/

activity_regularizer

Optional regularizer function for the output of this layer.

add_loss(*losses*, ***kwargs*)

Add loss tensor(s), potentially dependent on layer inputs.

Some losses (for instance, activity regularization losses) may be dependent on the inputs passed when calling a layer. Hence, when reusing the same layer on different inputs *a* and *b*, some entries in *layer.losses* may be dependent on *a* and some on *b*. This method automatically keeps track of dependencies.

This method can be used inside a subclassed layer or model's *call* function, in which case *losses* should be a Tensor or list of Tensors.

Example:

‘‘‘python class MyLayer(tf.keras.layers.Layer):
def call(self, inputs): self.add_loss(tf.abs(tf.reduce_mean(inputs))) return inputs
‘‘‘

This method can also be called directly on a Functional Model during construction. In this case, any loss Tensors passed to this Model must be symbolic and be able to be traced back to the model's *Input's. These losses become part of the model's topology and are tracked in 'get_config*.

Example:

`python inputs = tf.keras.Input(shape=(10,)) x = tf.keras.layers.Dense(10)(inputs) outputs = tf.keras.layers.Dense(1)(x) model = tf.keras.Model(inputs, outputs) # Activity regularization. model.add_loss(tf.abs(tf.reduce_mean(x))) `

If this is not the case for your loss (if, for example, your loss references a *Variable* of one of the model's layers), you can wrap your loss in a zero-argument lambda. These losses are not tracked as part of the model's topology since they can't be serialized.

Example:

`python inputs = tf.keras.Input(shape=(10,)) d = tf.keras.layers.Dense(10) x = d(inputs) outputs = tf.keras.layers.Dense(1)(x) model = tf.keras.Model(inputs, outputs) # Weight regularization. model.add_loss(lambda: tf.reduce_mean(d.kernel)) `

Arguments:

losses: Loss tensor, or list/tuple of tensors. Rather than tensors, losses may also be zero-argument callables which create a loss tensor.

****kwargs: Additional keyword arguments for backward compatibility.**

Accepted values: inputs - Deprecated, will be automatically inferred.

add_metric(*value*, *name=None*, ***kwargs*)

Adds metric tensor to the layer.

This method can be used inside the *call()* method of a subclassed layer or model.

‘‘‘python class MyMetricLayer(tf.keras.layers.Layer):

def __init__(self): super(MyMetricLayer, self).__init__(name='my_metric_layer') self.mean = tf.keras.metrics.Mean(name='metric_1')

def call(self, inputs): self.add_metric(self.mean(x)) self.add_metric(tf.reduce_sum(x), name='metric_2') return inputs

‘‘‘

This method can also be called directly on a Functional Model during construction. In this case, any tensor passed to this Model must be symbolic and be able to be traced back to the model's *Input‘s. These metrics become part of the model's topology and are tracked when you save the model via ‘save()*.

`python inputs = tf.keras.Input(shape=(10,)) x = tf.keras.layers.Dense(10)(inputs) outputs = tf.keras.layers.Dense(1)(x) model = tf.keras.Model(inputs, outputs) model.add_metric(math_ops.reduce_sum(x), name='metric_1') `

Note: Calling *add_metric()* with the result of a metric object on a Functional Model, as shown in the example below, is not supported. This is because we cannot trace the metric result tensor back to the model's inputs.

`python inputs = tf.keras.Input(shape=(10,)) x = tf.keras.layers.Dense(10)(inputs) outputs = tf.keras.layers.Dense(1)(x) model = tf.keras.Model(inputs, outputs) model.add_metric(tf.keras.metrics.Mean()(x), name='metric_1') `

Args: value: Metric tensor. name: String metric name. ****kwargs: Additional keyword arguments for backward compatibility.

Accepted values: *aggregation* - When the *value* tensor provided is not the result of calling a *keras.Metric* instance, it will be aggregated by default using a *keras.Metric.Mean*.

add_update(*updates*, *inputs=None*)

Add update op(s), potentially dependent on layer inputs.

Weight updates (for instance, the updates of the moving mean and variance in a BatchNormalization layer) may be dependent on the inputs passed when calling a layer. Hence, when reusing the same layer on different inputs *a* and *b*, some entries in *layer.updates* may be dependent on *a* and some on *b*. This method automatically keeps track of dependencies.

This call is ignored when eager execution is enabled (in that case, variable updates are run on the fly and thus do not need to be tracked for later execution).

Arguments:

updates: Update op, or list/tuple of update ops, or zero-arg callable that returns an update op. A zero-arg callable should be passed in order to disable running the updates by setting *trainable=False* on this Layer, when executing in Eager mode.

inputs: Deprecated, will be automatically inferred.

add_variable(**args*, ***kwargs*)

Deprecated, do NOT use! Alias for *add_weight*.

add_weight(*name=None*, *shape=None*, *dtype=None*, *initializer=None*, *regularizer=None*, *trainable=None*, *constraint=None*, *use_resource=None*, *synchronization=<VariableSynchronization.AUTO: 0>*, *aggregation=<VariableAggregation.NONE: 0>*, ***kwargs*)

Adds a new variable to the layer.

Arguments: name: Variable name. shape: Variable shape. Defaults to scalar if unspecified. dtype: The type of the variable. Defaults to *self.dtype*. initializer: Initializer instance (callable). regularizer: Regularizer instance (callable). trainable: Boolean, whether the variable should be part of the layer's

"trainable_variables" (e.g. variables, biases) or "non_trainable_variables" (e.g. BatchNorm mean and variance). Note that *trainable* cannot be *True* if *synchronization* is set to *ON_READ*.

constraint: Constraint instance (callable). use_resource: Whether to use *ResourceVariable*. synchronization: Indicates when a distributed a variable will be

aggregated. Accepted values are constants defined in the class *tf.VariableSynchronization*. By default the synchronization is set to *AUTO* and the current *DistributionStrategy* chooses when to synchronize. If *synchronization* is set to *ON_READ*, *trainable* must not be set to *True*.

aggregation: Indicates how a distributed variable will be aggregated. Accepted values are constants defined in the class *tf.VariableAggregation*.

****kwargs: Additional keyword arguments. Accepted values are *getter*,** *collections*, *experimental_autocast* and *caching_device*.

Returns: The variable created.

Raises:

ValueError: When giving unsupported dtype and no initializer or when trainable has been set to True with synchronization set as *ON_READ*.

apply(*inputs*, **args*, ***kwargs*)

Deprecated, do NOT use!

This is an alias of *self.__call__*.

Arguments: inputs: Input tensor(s). *args: additional positional arguments to be passed to *self.call*. **kwargs: additional keyword arguments to be passed to *self.call*.

Returns: Output tensor(s).

build(*input_shape*)

Creates the variables of the layer (optional, for subclass implementers).

This is a method that implementers of subclasses of *Layer* or *Model* can override if they need a state-creation step in-between layer instantiation and layer call.

This is typically used to create the weights of *Layer* subclasses.

Arguments:

input_shape: Instance of *TensorShape*, or list of instances of *TensorShape* if the layer expects a list of inputs (one instance per input).

call(*x*)

This is where the layer's logic lives.

Note here that *call()* method in *tf.keras* is little bit different from *keras* API. In *keras* API, you can pass support masking for layers as additional arguments. Whereas *tf.keras* has *compute_mask()* method to support masking.

Arguments: inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword arguments. Currently unused.

Returns: A tensor or list/tuple of tensors.

compute_dtype

The dtype of the layer's computations.

This is equivalent to *Layer.dtype_policy.compute_dtype*. Unless mixed precision is used, this is the same as *Layer.dtype*, the dtype of the weights.

Layers automatically cast their inputs to the compute dtype, which causes computations and the output to be in the compute dtype as well. This is done by the base Layer class in *Layer.__call__*, so you do not have to insert these casts if implementing your own layer.

Layers often perform certain internal computations in higher precision when *compute_dtype* is float16 or bfloat16 for numeric stability. The output will still typically be float16 or bfloat16 in such cases.

Returns: The layer's compute dtype.

compute_mask(*inputs*, *mask=None*)

Computes an output mask tensor.

Arguments: inputs: Tensor or list of tensors. mask: Tensor or list of tensors.

Returns:

None or a tensor (or list of tensors, one per output tensor of the layer).

compute_output_shape(*input_shape*)

Computes the output shape of the layer.

If the layer has not been built, this method will call *build* on the layer. This assumes that the layer will later be used with inputs that match the input shape provided here.

Arguments:

input_shape: Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns: An input shape tuple.

compute_output_signature(*input_signature*)

Compute the output tensor signature of the layer based on the inputs.

Unlike a TensorShape object, a TensorSpec object contains both shape and dtype information for a tensor. This method allows layers to provide output dtype information if it is different from the input dtype. For any layer that doesn't implement this function, the framework will fall back to use *compute_output_shape*, and will assume that the output dtype matches the input dtype.

Args:

input_signature: Single TensorSpec or nested structure of TensorSpec objects, describing a candidate input for the layer.

Returns:

Single TensorSpec or nested structure of TensorSpec objects, describing how the layer would transform the provided input.

Raises: TypeError: If input_signature contains a non-TensorSpec object.

count_params()
Count the total number of scalars composing the weights.
Returns: An integer count.
Raises:
ValueError: if the layer isn't yet built (in which case its weights aren't yet defined).

dtype
The dtype of the layer weights.

This is equivalent to *Layer.dtype_policy.variable_dtype*. Unless mixed precision is used, this is the same as *Layer.compute_dtype*, the dtype of the layer's computations.

dtype_policy
The dtype policy associated with this layer.

This is an instance of a *tf.keras.mixed_precision.Policy*.

dynamic
Whether the layer is dynamic (eager-only); set in the constructor.

classmethod from_config(*config*)
Creates a layer from its config.

This method is the reverse of *get_config*, capable of instantiating the same layer from the config dictionary. It does not handle layer connectivity (handled by Network), nor weights (handled by *set_weights*).
Arguments:
config: A Python dictionary, typically the output of get_config.
Returns: A layer instance.

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).
Returns: Python dictionary.

get_input_at(*node_index*)
Retrieves the input tensor(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.
Returns: A tensor (or list of tensors if the layer has multiple inputs).
Raises: RuntimeError: If called in Eager mode.

get_input_mask_at(*node_index*)
Retrieves the input mask tensor(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.
Returns: A mask tensor (or list of tensors if the layer has multiple inputs).

get_input_shape_at(*node_index*)
Retrieves the input shape(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.

Returns: A shape tuple (or list of shape tuples if the layer has multiple inputs).
Raises: RuntimeError: If called in Eager mode.

get_losses_for(*inputs*)
Deprecated, do NOT use!

Retrieves losses relevant to a specific set of inputs.
Arguments: inputs: Input tensor or list/tuple of input tensors.
Returns: List of loss tensors of the layer that depend on *inputs*.

get_output_at(*node_index*)
Retrieves the output tensor(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.
Returns: A tensor (or list of tensors if the layer has multiple outputs).
Raises: RuntimeError: If called in Eager mode.

get_output_mask_at(*node_index*)
Retrieves the output mask tensor(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.
Returns: A mask tensor (or list of tensors if the layer has multiple outputs).

get_output_shape_at(*node_index*)
Retrieves the output shape(s) of a layer at a given node.
Arguments:
node_index: Integer, index of the node from which to retrieve the attribute. E.g. *node_index=0* will correspond to the first time the layer was called.
Returns: A shape tuple (or list of shape tuples if the layer has multiple outputs).
Raises: RuntimeError: If called in Eager mode.

get_updates_for(*inputs*)
Deprecated, do NOT use!

Retrieves updates relevant to a specific set of inputs.
Arguments: inputs: Input tensor or list/tuple of input tensors.
Returns: List of update ops of the layer that depend on *inputs*.

get_weights()
Returns the current weights of the layer.

The weights of a layer represent the state of the layer. This function returns both trainable and non-trainable weight values associated with this layer as a list of Numpy arrays, which can in turn be used to load state into similarly parameterized layers.

For example, a Dense layer returns a list of two values– per-output weights and the bias value. These can be used to set the weights of another Dense layer:

```
>>> a = tf.keras.layers.Dense(1,
...   kernel_initializer=tf.constant_initializer(1.))
>>> a_out = a(tf.convert_to_tensor([[1., 2., 3.]]))
>>> a.get_weights()
[array([[1.],
       [1.],
       [1.]], dtype=float32), array([0.], dtype=float32)]
>>> b = tf.keras.layers.Dense(1,
...   kernel_initializer=tf.constant_initializer(2.))
```

(continues on next page)

(continued from previous page)

```
>>> b_out = b(tf.convert_to_tensor([[10., 20., 30.]]))
>>> b.get_weights()
[array([[2.],
       [2.],
       [2.]], dtype=float32), array([0.], dtype=float32)]
>>> b.set_weights(a.get_weights())
>>> b.get_weights()
[array([[1.],
       [1.],
       [1.]], dtype=float32), array([0.], dtype=float32)]
```

Returns: Weights values as a list of numpy arrays.

inbound_nodes
Deprecated, do NOT use! Only for compatibility with external Keras.

input
Retrieves the input tensor(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer.
Returns: Input tensor or list of input tensors.
Raises: RuntimeError: If called in Eager mode. AttributeError: If no inbound nodes are found.

input_mask
Retrieves the input mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.
Returns: Input mask tensor (potentially None) or list of input mask tensors.
Raises: AttributeError: if the layer is connected to more than one incoming layers.

input_shape
Retrieves the input shape(s) of a layer.

Only applicable if the layer has exactly one input, i.e. if it is connected to one incoming layer, or if all inputs have the same shape.
Returns: Input shape, as an integer shape tuple (or list of shape tuples, one tuple per input tensor).
Raises: AttributeError: if the layer has no defined input_shape. RuntimeError: if called in Eager mode.

input_spec
InputSpec instance(s) describing the input format for this layer.

When you create a layer subclass, you can set *self.input_spec* to enable the layer to run input compatibility checks when it is called. Consider a *Conv2D* layer: it can only be called on a single input tensor of rank 4. As such, you can set, in *__init__()*:

`` `python self.input_spec = tf.keras.layers.InputSpec(ndim=4) ` ``

Now, if you try to call the layer on an input that isn't rank 4 (for instance, an input of shape *(2,)*, it will raise a nicely-formatted error:

```
` ValueError:  Input 0 of layer conv2d is incompatible with the
layer:  expected ndim=4, found ndim=1. Full shape received:  [2] `
```

Input checks that can be specified via *input_spec* include: - Structure (e.g. a single input, a list of 2 inputs, etc) - Shape - Rank (ndim) - Dtype

For more information, see *tf.keras.layers.InputSpec*.
Returns: A *tf.keras.layers.InputSpec* instance, or nested structure thereof.

losses

List of losses added using the *add_loss()* API.

Variable regularization tensors are created when this property is accessed, so it is eager safe: accessing *losses* under a *tf.GradientTape* will propagate gradients back to the corresponding variables.

Examples:

```
>>> class MyLayer(tf.keras.layers.Layer):
...   def call(self, inputs):
...     self.add_loss(tf.abs(tf.reduce_mean(inputs)))
...     return inputs
>>> l = MyLayer()
>>> l(np.ones((10, 1)))
>>> l.losses
[1.0]
```

```
>>> inputs = tf.keras.Input(shape=(10,))
>>> x = tf.keras.layers.Dense(10)(inputs)
>>> outputs = tf.keras.layers.Dense(1)(x)
>>> model = tf.keras.Model(inputs, outputs)
>>> # Activity regularization.
>>> len(model.losses)
0
>>> model.add_loss(tf.abs(tf.reduce_mean(x)))
>>> len(model.losses)
1
```

```
>>> inputs = tf.keras.Input(shape=(10,))
>>> d = tf.keras.layers.Dense(10, kernel_initializer='ones')
>>> x = d(inputs)
>>> outputs = tf.keras.layers.Dense(1)(x)
>>> model = tf.keras.Model(inputs, outputs)
>>> # Weight regularization.
>>> model.add_loss(lambda: tf.reduce_mean(d.kernel))
>>> model.losses
[<tf.Tensor: shape=(), dtype=float32, numpy=1.0>]
```

Returns: A list of tensors.

metrics

List of metrics added using the *add_metric()* API.

Example:

```
>>> input = tf.keras.layers.Input(shape=(3,))
>>> d = tf.keras.layers.Dense(2)
>>> output = d(input)
>>> d.add_metric(tf.reduce_max(output), name='max')
>>> d.add_metric(tf.reduce_min(output), name='min')
>>> [m.name for m in d.metrics]
['max', 'min']
```

Returns: A list of *Metric* objects.

name

Name of the layer (string), set in the constructor.

name_scope

Returns a *tf.name_scope* instance for this class.

non_trainable_variables

non_trainable_weights
List of all non-trainable weights tracked by this layer.

Non-trainable weights are *not* updated during training. They are expected to be updated manually in *call()*.

Note: This will not track the weights of nested *tf.Modules* that are not themselves Keras layers.
Returns: A list of non-trainable variables.

outbound_nodes
Deprecated, do NOT use! Only for compatibility with external Keras.

output
Retrieves the output tensor(s) of a layer.

Only applicable if the layer has exactly one output, i.e. if it is connected to one incoming layer.
Returns: Output tensor or list of output tensors.
Raises:
AttributeError: if the layer is connected to more than one incoming layers.
RuntimeError: if called in Eager mode.

output_mask
Retrieves the output mask tensor(s) of a layer.

Only applicable if the layer has exactly one inbound node, i.e. if it is connected to one incoming layer.
Returns: Output mask tensor (potentially None) or list of output mask tensors.
Raises: AttributeError: if the layer is connected to more than one incoming layers.

output_shape
Retrieves the output shape(s) of a layer.

Only applicable if the layer has one output, or if all outputs have the same shape.
Returns: Output shape, as an integer shape tuple (or list of shape tuples, one tuple per output tensor).
Raises: AttributeError: if the layer has no defined output shape. RuntimeError: if called in Eager mode.

set_weights(*weights*)
Sets the weights of the layer, from Numpy arrays.

The weights of a layer represent the state of the layer. This function sets the weight values from numpy arrays. The weight values should be passed in the order they are created by the layer. Note that the layer's weights must be instantiated before calling this function by calling the layer.

For example, a Dense layer returns a list of two values– per-output weights and the bias value. These can be used to set the weights of another Dense layer:

```
>>> a = tf.keras.layers.Dense(1,
...   kernel_initializer=tf.constant_initializer(1.))
>>> a_out = a(tf.convert_to_tensor([[1., 2., 3.]]))
>>> a.get_weights()
[array([[1.],
       [1.],
       [1.]], dtype=float32), array([0.], dtype=float32)]
>>> b = tf.keras.layers.Dense(1,
...   kernel_initializer=tf.constant_initializer(2.))
>>> b_out = b(tf.convert_to_tensor([[10., 20., 30.]]))
>>> b.get_weights()
[array([[2.],
       [2.],
```

(continues on next page)

(continued from previous page)

```
       [2.]], dtype=float32), array([0.], dtype=float32)]
>>> b.set_weights(a.get_weights())
>>> b.get_weights()
[array([[1.],
       [1.],
       [1.]], dtype=float32), array([0.], dtype=float32)]
```

Arguments:

weights: a list of Numpy arrays. The number of arrays and their shape must match number of the dimensions of the weights of the layer (i.e. it should match the output of *get_weights*).

Raises:

ValueError: If the provided weights list does not match the layer's specifications.

stateful

submodules

Sequence of all sub-modules.

Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).

```
>>> a = tf.Module()
>>> b = tf.Module()
>>> c = tf.Module()
>>> a.b = b
>>> b.c = c
>>> list(a.submodules) == [b, c]
True
>>> list(b.submodules) == [c]
True
>>> list(c.submodules) == []
True
```

Returns: A sequence of all submodules.

supports_masking

Whether this layer supports computing a mask using *compute_mask*.

trainable

trainable_variables

Sequence of trainable variables owned by this module and its submodules.

Note: this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

Returns: A sequence of variables for the current module (sorted by attribute name) followed by variables from all submodules recursively (breadth first).

trainable_weights

List of all trainable weights tracked by this layer.

Trainable weights are updated via gradient descent during training.

Note: This will not track the weights of nested *tf.Modules* that are not themselves Keras layers.

Returns: A list of trainable variables.

updates

variable_dtype
Alias of *Layer.dtype*, the dtype of the weights.

variables
Returns the list of all layer variables/weights.

Alias of *self.weights*.

Note: This will not track the weights of nested *tf.Modules* that are not themselves Keras layers.
Returns: A list of variables.

weights
Returns the list of all layer variables/weights.

Note: This will not track the weights of nested *tf.Modules* that are not themselves Keras layers.
Returns: A list of variables.

classmethod with_name_scope(*method*)
Decorator to automatically enter the module name scope.

```
>>> class MyModule(tf.Module):
...   @tf.Module.with_name_scope
...   def __call__(self, x):
...     if not hasattr(self, 'w'):
...       self.w = tf.Variable(tf.random.normal([x.shape[1], 3]))
...     return tf.matmul(x, self.w)
```

Using the above module would produce ‘tf.Variable‘s and ‘tf.Tensor‘s whose names included the module name:

```
>>> mod = MyModule()
>>> mod(tf.ones([1, 2]))
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=..., dtype=float32)>
>>> mod.w
<tf.Variable 'my_module/Variable:0' shape=(2, 3) dtype=float32,
numpy=..., dtype=float32)>
```

Args: method: The method to wrap.
Returns: The original method wrapped such that it enters the module's name scope.

build()
Builds the VGGish Keras model.

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

download_pretrained_weights(*weights_folder='./pretrained_weights'*)
Download pretrained weights from: https://github.com/DTaoo/VGGish https://drive.google.com/file/d/1mhqXZ8CANgHyepum7N4yrjiyIg6qaMe6/view

Code based on: https://github.com/beasteers/VGGish/blob/master/vggish_keras/download_helpers/download_weights.py

Parameters

weights_folder [str] Path to save the weights file

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.6 dcase_models.model.SMel

class dcase_models.model.**SMel**(*model=None, model_path=None, metrics=['mean_squared_error'], mel_bands=128, n_seqs=64, audio_win=1024, audio_hop=512, alpha=1, scaler=None, amin=1e-10*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for SMel model.

P. Zinemanas, P. Cancela, M. Rocamora. "End–to–end Convolutional Neural Networks for Sound Event Detection in Urban Environments" Proceedings of the 24th Conference of Open Innovations Association FRUCT, 3rd IEEE FRUCT International Workshop on Semantic Audio and the Internet of Things. Moscow, Russia, April 2019.

Parameters

mel_bands [int, default=128] Number of mel bands.

n_seqs [int, default=64] Time dimmension of the input.

audio_win [int, default=1024] Length of the audio window (number of samples of each frame).

audio_hop [int, default=512] Length of the hop size (in samples).

alpha [int, default=1] Multiply factor before apply log (compression factor).

scaler [tuple, list or None] If scaler is not None, this is used before output.

amin [float, default=1e-10 (-100 dB)] Minimum value for db calculation.

Examples

```
>>> from dcase_models.model.models import SMel
>>> model_container = SMel()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 1024)          0
_________________________________________________________________
lambda (Lambda)              (None, 64, 1024, 1)       0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 64, 64, 128)       131200
_________________________________________________________________
lambda_1 (Lambda)            (None, 64, 64, 128)       0
_________________________________________________________________
lambda_2 (Lambda)            (None, 64, 128)           0
_________________________________________________________________
lambda_3 (Lambda)            (None, 64, 128)           0
```

(continues on next page)

(continued from previous page)

```
=================================================================
Total params: 131,200
Trainable params: 131,200
Non-trainable params: 0
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__ (*model=None, model_path=None, metrics=['mean_squared_error'], mel_bands=128, n_seqs=64, audio_win=1024, audio_hop=512, alpha=1, scaler=None, amin=1e-10*)
Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

__init__([model, model_path, metrics, ...])	Initialize ModelContainer
build()	Builds the SMel Keras model.
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.
train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()
Builds the SMel Keras model.

check_if_model_exists (*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

8.2.7 dcase_models.model.MST

class dcase_models.model.**MST**(*model=None, model_path=None, metrics=['mean_squared_error'], mel_bands=128, sequence_samples=22050, audio_win=1024, audio_hop=512*)

Bases: dcase_models.model.container.KerasModelContainer

KerasModelContainer for MST model.

T. M. S. Tax, J. L. D. Antich, H. Purwins, and L. Maaløe. "Utilizing domain knowledge in end-to-end audio processing" 31st Conference on Neural Information Processing Systems (NIPS). Long Beach, CA, USA, 2017.

Parameters

mel_bands [int, default=128] Number of mel bands.

sequence_samples [int, default=22050] Number of samples of each input.

audio_win [int, default=1024] Length of the audio window (number of samples of each frame).

audio_hop [int, default=512] Length of the hop size (in samples).

Examples

```
>>> from dcase_models.model.models import SMel
>>> model_container = SMel()
>>> model_container.model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 22050)             0
_________________________________________________________________
lambda (Lambda)              (None, 22050, 1)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 44, 512)           524800
_________________________________________________________________
batch_normalization_1 (Batch (None, 44, 512)           2048
_________________________________________________________________
activation_1 (Activation)    (None, 44, 512)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 44, 256)           393472
_________________________________________________________________
batch_normalization_2 (Batch (None, 44, 256)           1024
```

(continues on next page)

(continued from previous page)

```
_________________________________________________________________
activation_2 (Activation)    (None, 44, 256)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 44, 128)           98432
_________________________________________________________________
batch_normalization_3 (Batch (None, 44, 128)           512
_________________________________________________________________
activation_3 (Activation)    (None, 44, 128)           0
=================================================================
Total params: 1,020,288
Trainable params: 1,018,496
Non-trainable params: 1,792
_________________________________________________________________
```

Attributes

model [keras.models.Model] Keras model.

__init__(*model=None, model_path=None, metrics=['mean_squared_error'], mel_bands=128, sequence_samples=22050, audio_win=1024, audio_hop=512*)

Initialize ModelContainer

Parameters

model [keras model or similar] Object that defines the model (i.e keras.models.Model)

model_path [str] Path to the model file

model_name [str] Model name

metrics [list of str] List of metrics used for evaluation

Methods

__init__([model, model_path, metrics, ...])	Initialize ModelContainer
build()	Builds the MST Keras model.
check_if_model_exists(folder, **kwargs)	Checks if the model already exits in the path.
cut_network(layer_where_to_cut)	Cuts the network at the layer passed as argument.
evaluate(data_test, **kwargs)	Evaluates the keras model using X_test and Y_test.
fine_tuning(layer_where_to_cut[, ...])	Create a new model for fine-tuning.
get_available_intermediate_outputs()	Return a list of available intermediate outputs.
get_intermediate_output(output_ix_name, inputs)	Return the output of the model in a given layer.
get_number_of_parameters()	Missing docstring here
load_model_from_json(folder, **kwargs)	Loads a model from a model.json file in the path given by folder.
load_model_weights(weights_folder)	Loads self.model weights in weights_folder/best_weights.hdf5.
load_pretrained_model_weights([weights_folder])	Loads pretrained weights to self.model weights.
save_model_json(folder)	Saves the model to a model.json file in the given folder path.
save_model_weights(weights_folder)	Saves self.model weights in weights_folder/best_weights.hdf5.

Continued on next page

Table 11 – continued from previous page

train(data_train, data_val[, weights_path, ...])	Trains the keras model using the data and paramaters of arguments.

build()
Builds the MST Keras model.

check_if_model_exists(*folder*, ***kwargs*)
Checks if the model already exits in the path.

Check if the folder/model.json file exists and includes the same model as self.model.

Parameters

folder [str] Path to the folder to check.

cut_network(*layer_where_to_cut*)
Cuts the network at the layer passed as argument.

Parameters

layer_where_to_cut [str or int] Layer name (str) or index (int) where cut the model.

Returns

keras.models.Model Cutted model.

evaluate(*data_test*, ***kwargs*)
Evaluates the keras model using X_test and Y_test.

Parameters

X_test [ndarray] 3D array with mel-spectrograms of test set. Shape = (N_instances, N_hops, N_mel_bands)

Y_test [ndarray] 2D array with the annotations of test set (one hot encoding). Shape (N_instances, N_classes)

scaler [Scaler, optional] Scaler objet to be applied if is not None.

Returns

float evaluation's accuracy

list list of annotations (ground_truth)

list list of model predictions

fine_tuning(*layer_where_to_cut*, *new_number_of_classes=10*, *new_activation='softmax'*, *freeze_source_model=True*, *new_model=None*)
Create a new model for fine-tuning.

Cut the model in the layer_where_to_cut layer and add a new fully-connected layer.

Parameters

layer_where_to_cut [str or int] Name (str) of index (int) of the layer where cut the model. This layer is included in the new model.

new_number_of_classes [int] Number of units in the new fully-connected layer (number of classes).

new_activation [str] Activation of the new fully-connected layer.

freeze_source_model [bool] If True, the source model is set to not be trainable.

new_model [Keras Model] If is not None, this model is added after the cut model. This is useful if you want add more than a fully-connected layer.

get_available_intermediate_outputs()
Return a list of available intermediate outputs.

Return a list of model's layers.

Returns

list of str List of layers names.

get_intermediate_output(*output_ix_name*, *inputs*)
Return the output of the model in a given layer.

Cut the model in the given layer and predict the output for the given inputs.

Returns

ndarray Output of the model in the given layer.

get_number_of_parameters()
Missing docstring here

load_model_from_json(*folder*, ***kwargs*)
Loads a model from a model.json file in the path given by folder. The model is load in self.model attribute.

Parameters

folder [str] Path to the folder that contains model.json file

load_model_weights(*weights_folder*)
Loads self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file.

load_pretrained_model_weights(*weights_folder='./pretrained_weights'*)
Loads pretrained weights to self.model weights.

Parameters

weights_folder [str] Path to load the weights file

save_model_json(*folder*)
Saves the model to a model.json file in the given folder path.

Parameters

folder [str] Path to the folder to save model.json file

save_model_weights(*weights_folder*)
Saves self.model weights in weights_folder/best_weights.hdf5.

Parameters

weights_folder [str] Path to save the weights file

train(*data_train, data_val, weights_path='./', optimizer='Adam', learning_rate=0.001, early_stopping=100, considered_improvement=0.01, losses='categorical_crossentropy', loss_weights=[1], sequence_time_sec=0.5, metric_resolution_sec=1.0, label_list=[], shuffle=True, **kwargs_keras_fit*)
Trains the keras model using the data and paramaters of arguments.

Parameters

X_train [ndarray] 3D array with mel-spectrograms of train set. Shape = (N_instances, N_hops, N_mel_bands)

Y_train [ndarray] 2D array with the annotations of train set (one hot encoding). Shape (N_instances, N_classes)

X_val [ndarray] 3D array with mel-spectrograms of validation set. Shape = (N_instances, N_hops, N_mel_bands)

Y_val [ndarray] 2D array with the annotations of validation set (one hot encoding). Shape (N_instances, N_classes)

weights_path [str] Path where to save the best weights of the model in the training process

weights_path [str] Path where to save log of the training process

loss_weights [list] List of weights for each loss function ('categorical_crossentropy', 'mean_squared_error', 'prototype_loss')

optimizer [str] Optimizer used to train the model

learning_rate [float] Learning rate used to train the model

batch_size [int] Batch size used in the training process

epochs [int] Number of training epochs

fit_verbose [int] Verbose mode for fit method of Keras model

CHAPTER 9

Utilities

9.1 Metric functions

predictions_temporal_integration(Y_predicted)	Integrate temporal dimension.
evaluate_metrics(model, data, metrics, **kwargs)	Calculate metrics over files with different length
sed(Y_val, Y_predicted[, sequence_time_sec, ...])	Calculate metrics for Sound Event Detection
classification(Y_val, Y_predicted[, label_list])	Calculate metrics for Audio Classification
tagging(Y_val, Y_predicted[, label_list])	Calculate metrics for Audio Tagging
accuracy	
ER	
F1	

9.1.1 dcase_models.util.predictions_temporal_integration

dcase_models.util.**predictions_temporal_integration**(*Y_predicted*, *type='sum'*)

Integrate temporal dimension.

Parameters

Y_predicted [ndarray] Signal to be integrated. e.g. shape (N_times, N_classes)

type [str] Type of integration ('sum', 'mean', 'autopool')

Returns

array Integrated signal. e.g. shape (N_classes,)

9.1.2 dcase_models.util.evaluate_metrics

dcase_models.util.**evaluate_metrics**(*model*, *data*, *metrics*, ***kwargs*)

Calculate metrics over files with different length

Parameters

model [keras Model] model to get the predictions

data [tuple or KerasDataGenerator] Validation data for model evaluation (X_val, Y_val) or KerasDataGenerator

X_val [list of ndarray] Each element in list is a 3D array with the mel-spectrograms of one file. Shape of each element: (N_windows, N_hops, N_mel_bands) N_windows can be different in each file (element)

Y_val [list ndarray] Each element in the list is a 1D array with the annotations (one hot encoding). Shape of each element (N_classes,)

metrics [list] List of metrics to apply. Each element can be a metric name or a function.

Returns

dict Dict with the results information.

{'annotations' [[Y0, Y1, . . .],] 'predictions' : [Yp0, Yp1, . . .], metrics[0]: 0.1, metrics[1]: 0.54}

9.1.3 dcase_models.util.sed

dcase_models.util.**sed**(*Y_val*, *Y_predicted*, *sequence_time_sec=0.5*, *metric_resolution_sec=1.0*, *label_list=[]*)

Calculate metrics for Sound Event Detection

Parameters

Y_val [list of ndarray] 2D array with the ground-truth event roll shape: (N_times, N_classes)

Y_predicted [list of ndarray] 2D array with the predicted event roll shape: (N_times, N_classes)

sequence_time_sec [float] Resolution of Y_val and Y_predicted.

metric_resolution_sec [float] Resolution of the metrics.

label_list: Label list.

Returns

sef_eval.sound_events.SegmentBasedMetrics Object with the SED results

9.1.4 dcase_models.util.classification

dcase_models.util.**classification**(*Y_val*, *Y_predicted*, *label_list=[]*)

Calculate metrics for Audio Classification

Parameters

Y_val [listy of ndarray] 2D array with the ground-truth event roll shape: (N_times, N_classes)

Y_predicted [list of ndarray] 2D array with the predicted event roll shape: (N_times, N_classes)

label_list: Label list.

Returns

sef_eval.scenes.SceneClassificationMetrics Object with the classification results

9.1.5 dcase_models.util.tagging

dcase_models.util.**tagging**(*Y_val*, *Y_predicted*, *label_list=[]*)

Calculate metrics for Audio Tagging

Parameters

Y_val [list of ndarray] 2D array with the ground-truth event roll shape: (N_times, N_classes)

Y_predicted [list of ndarray] 2D array with the predicted event roll shape: (N_times, N_classes)

label_list: Label list.

Returns

sef_eval.scenes.AudioTaggingMetrics Object with the tagging results

9.2 Data functions

get_fold_val(fold_test, fold_list)	Get the validation fold given the test fold.
evaluation_setup(fold_test, folds, . . . [, . . .])	Return a evaluation setup given by the evaluation_mode.

9.2.1 dcase_models.util.get_fold_val

dcase_models.util.**get_fold_val**(*fold_test*, *fold_list*)

Get the validation fold given the test fold.

Useful for cross-validation evaluation mode.

Return the next fold in a circular way. e.g. if the fold_list is ['fold1', 'fold2',. . . ,'fold10'] and the fold_test is 'fold1', then return 'fold2'. If the fold_test is 'fold10', return 'fold1'.

Parameters

fold_test [str] Fold used for model testing.

fold_list [list of str] Fold list.

Returns

str Validation fold.

9.2.2 dcase_models.util.evaluation_setup

dcase_models.util.**evaluation_setup**(*fold_test*, *folds*, *evaluation_mode*, *use_validate_set=True*)

Return a evaluation setup given by the evaluation_mode.

Return fold list for training, validatin and testing the model.

Each evaluation_mode return different lists.

Parameters

fold_test [str] Fold used for model testing.

folds [list of str] Fold list.

evaluation_mode [str] Evaluation mode ('cross-validation', 'train-validate-test', 'cross-validation-with-test', 'train-test')

use_validate_set [bool] If not, the validation set is the same as the train set.

Returns

list List of folds for training

list List of folds for validating

list List of folds for testing

9.3 Events functions

contiguous_regions(act)	
evaluation_setup(fold_test, folds, . . . [, . . .])	Return a evaluation setup given by the evaluation_mode.
event_roll_to_event_list(event_roll, . . .)	Convert a event roll matrix to a event list.
tag_probabilities_to_tag_list(. . . [, threshold])	Convert a tag probabilites matrix to a tag list.

9.3.1 dcase_models.util.contiguous_regions

dcase_models.util.**contiguous_regions**(*act*)

9.3.2 dcase_models.util.event_roll_to_event_list

dcase_models.util.**event_roll_to_event_list**(*event_roll*, *event_label_list*, *time_resolution*)

Convert a event roll matrix to a event list.

Parameters

event_roll [ndarray] Shape (N_times, N_classes)

event_label_list [list of str] Label list

time_resolution [float] Time resolution of the event_roll.

Returns

list List of dicts with events information. e.g.

[{'event_onset': 0.1, 'event_offset': 1.5, 'event_label' : 'dog'}, . . .]

9.3.3 dcase_models.util.tag_probabilities_to_tag_list

dcase_models.util.**tag_probabilities_to_tag_list**(*tag_probabilities*, *label_list*, *threshold=0.5*)

Convert a tag probabilites matrix to a tag list.

Parameters

tag_probabilities [ndarray] Shape (N_times, N_classes)

label_list [list of str] Label list

threshold [float] Threshold to decide if a tag is present.

Returns

list List of tags. e.g. ['dog', 'cat', . . .]

9.4 Files functions

save_json(path, json_string)	Save a json file in the location given by path.
load_json(path)	Load a json file from path.
save_pickle(X, path)	Save a pickle object in the location given by path.
load_pickle(path)	Load a pickle object from path.
list_all_files(path)	List all files in the path including subfolders.
list_wav_files(path)	List all wav files in the path including subfolders.
load_training_log(weights_folder)	Load the training log files of keras.
mkdir_if_not_exists(path[, parents])	Make dir if does not exists.
download_files_and_unzip(dataset_folder, ...)	Download files from zenodo and decompress them.
move_all_files_to(source, destination)	Move all files from source to destination
move_all_files_to_parent(parent, child)	Move all files in parent/child to the parent/
duplicate_folder_structure(origin_path, ...)	Duplicate the folder structure from the origin to the destination.
example_audio_file([index])	Get path to an example audio file

9.4.1 dcase_models.util.save_json

dcase_models.util.**save_json**(*path*, *json_string*)

Save a json file in the location given by path.

Parameters

path [str] Path to json file.

json_string [str] JSON string to be saved.

9.4.2 dcase_models.util.load_json

dcase_models.util.**load_json**(*path*)

Load a json file from path.

Parameters

path [str] Path to json file.

Returns

dict Data from the json file.

9.4.3 dcase_models.util.save_pickle

dcase_models.util.**save_pickle**(*X*, *path*)

Save a pickle object in the location given by path.

Parameters

X [pickle object] Object to be saved.

path [str] Path to pickle file.

9.4.4 dcase_models.util.load_pickle

dcase_models.util.**load_pickle**(*path*)
Load a pickle object from path.

Parameters

path [str] Path to pickle file.

Returns

pickle object Loaded pickle object.

9.4.5 dcase_models.util.list_all_files

dcase_models.util.**list_all_files**(*path*)
List all files in the path including subfolders.

Parameters

path [str] Path to files.

Returns

list List of paths to the files.

9.4.6 dcase_models.util.list_wav_files

dcase_models.util.**list_wav_files**(*path*)
List all wav files in the path including subfolders.

Parameters

path [str] Path to wav files.

Returns

list List of paths to the wav files.

9.4.7 dcase_models.util.load_training_log

dcase_models.util.**load_training_log**(*weights_folder*)
Load the training log files of keras.

Parameters

weights_folder [str] Path to training log folder.

Returns

dict Dict with the log information. Each key in the dict includes information of some variable.
e.g. {'loss': [0.1, . . .], 'accuracy': [80.1, . . .]}

9.4.8 dcase_models.util.mkdir_if_not_exists

dcase_models.util.**mkdir_if_not_exists**(*path*, *parents=False*)
Make dir if does not exists.

If parents is True, also creates all parents needed.

Parameters

path [str] Path to folder to be created.

parents [bool, optional] If True, also creates all parents needed.

9.4.9 dcase_models.util.download_files_and_unzip

dcase_models.util.**download_files_and_unzip**(*dataset_folder*, *zenodo_url*, *zenodo_files*)
Download files from zenodo and decompress them.

Parameters

dataset_folder [str] Path to the folder where download the files.

zenodo_url [str] Url to the zenodo repository.

zenodo_files [list of str] List of file names to download.

9.4.10 dcase_models.util.move_all_files_to

dcase_models.util.**move_all_files_to**(*source*, *destination*)
Move all files from source to destination

Parameters

source [str] Path to the source folder.

destination [str] Folder to the destination folder.

9.4.11 dcase_models.util.move_all_files_to_parent

dcase_models.util.**move_all_files_to_parent**(*parent*, *child*)
Move all files in parent/child to the parent/

Parameters

parent [str] Path to the parent folder.

child [str] Folder name of the child folder.

9.4.12 dcase_models.util.duplicate_folder_structure

dcase_models.util.**duplicate_folder_structure**(*origin_path*, *destination_path*)
Duplicate the folder structure from the origin to the destination.

Parameters

origin_path [str] Origin path.

destination_path [str] Destination path.

9.4.13 dcase_models.util.example_audio_file

dcase_models.util.**example_audio_file**(*index=0*)
Get path to an example audio file

Parameters

index [int, default=0] Index of the audio file

Returns

path [str] Path to the example audio file

9.5 Callback functions

ClassificationCallback(data[, file_weights, ...])	Keras callback to calculate acc after each epoch and save file with the weights if the evaluation improves
SEDCallback(data[, file_weights, best_F1, ...])	Keras callback to calculate F1 and ER after each epoch and save file with the weights if the evaluation improves.
TaggingCallback(data[, file_weights, ...])	Keras callback to calculate acc after each epoch and save file with the weights if the evaluation improves
F1ERCallback	

9.5.1 dcase_models.util.ClassificationCallback

class dcase_models.util.**ClassificationCallback**(*data*, *file_weights=None*, *best_acc=0*, *early_stopping=0*, *considered_improvement=0.01*, *label_list=[]*)
Bases: tensorflow.python.keras.callbacks.Callback

Keras callback to calculate acc after each epoch and save file with the weights if the evaluation improves

__init__(*data*, *file_weights=None*, *best_acc=0*, *early_stopping=0*, *considered_improvement=0.01*, *label_list=[]*)
Initialize the keras callback

Parameters

data [tuple or KerasDataGenerator] Validation data for model evaluation (X_val, Y_val) or KerasDataGenerator

file_weights [string] Path to the file with the weights

best_acc [float] Last accuracy value, only if continue

early_stopping [int] Number of epochs for cut the training if not improves if 0, do not use it

Methods

__init__(data[, file_weights, best_acc, ...])	Initialize the keras callback
on_batch_begin(batch[, logs])	A backwards compatibility alias for *on_train_batch_begin*.

Continued on next page

Table 6 – continued from previous page

on_batch_end(batch[, logs])	A backwards compatibility alias for *on_train_batch_end*.
on_epoch_begin(epoch[, logs])	Called at the start of an epoch.
on_epoch_end(epoch[, logs])	This function is run when each epoch ends.
on_predict_batch_begin(batch[, logs])	Called at the beginning of a batch in *predict* methods.
on_predict_batch_end(batch[, logs])	Called at the end of a batch in *predict* methods.
on_predict_begin([logs])	Called at the beginning of prediction.
on_predict_end([logs])	Called at the end of prediction.
on_test_batch_begin(batch[, logs])	Called at the beginning of a batch in *evaluate* methods.
on_test_batch_end(batch[, logs])	Called at the end of a batch in *evaluate* methods.
on_test_begin([logs])	Called at the beginning of evaluation or validation.
on_test_end([logs])	Called at the end of evaluation or validation.
on_train_batch_begin(batch[, logs])	Called at the beginning of a training batch in *fit* methods.
on_train_batch_end(batch[, logs])	Called at the end of a training batch in *fit* methods.
on_train_begin([logs])	Called at the beginning of training.
on_train_end([logs])	Called at the end of training.
set_model(model)	
set_params(params)	

on_batch_begin(*batch*, *logs=None*)
A backwards compatibility alias for *on_train_batch_begin*.

on_batch_end(*batch*, *logs=None*)
A backwards compatibility alias for *on_train_batch_end*.

on_epoch_begin(*epoch*, *logs=None*)
Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Arguments: epoch: Integer, index of epoch. logs: Dict. Currently no data is passed to this argument for this method

but that may change in the future.

on_epoch_end(*epoch*, *logs={}*)
This function is run when each epoch ends. The metrics are calculated, printed and saved to the log file.

Parameters

epoch [int] number of epoch (from Callback class)

logs [dict] log data (from Callback class)

on_predict_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.predict_step*,

it typically returns a dict with a key 'outputs' containing the model's outputs.

on_predict_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_predict_begin(*logs=None*)
Called at the beginning of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_predict_end(*logs=None*)
Called at the end of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.test_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_test_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_test_begin(*logs=None*)
Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_end(*logs=None*)
Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_test_batch_end()* is passed to this argument for this method but that may change in the future.

on_train_batch_begin(*batch*, *logs=None*)
Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.train_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_train_batch_end(*batch*, *logs=None*)
Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_train_begin(*logs=None*)
Called at the beginning of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_train_end(*logs=None*)
Called at the end of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_epoch_end()* is passed to this argument for this method but that may change in the future.

set_model(*model*)

set_params(*params*)

9.5.2 dcase_models.util.SEDCallback

class dcase_models.util.**SEDCallback**(*data*, *file_weights=None*, *best_F1=0*, *early_stopping=0*, *considered_improvement=0.01*, *sequence_time_sec=0.5*, *metric_resolution_sec=1.0*, *label_list=[]*)

Bases: tensorflow.python.keras.callbacks.Callback

Keras callback to calculate F1 and ER after each epoch and save file with the weights if the evaluation improves.

Use sed_eval library.

__init__(*data*, *file_weights=None*, *best_F1=0*, *early_stopping=0*, *considered_improvement=0.01*, *sequence_time_sec=0.5*, *metric_resolution_sec=1.0*, *label_list=[]*)

Initialize the keras callback

Parameters

data [tuple or KerasDataGenerator] Validation data for model evaluation (X_val, Y_val) or KerasDataGenerator

file_weights [string] Path to the file with the weights

best_acc [float] Last accuracy value, only if continue

early_stopping [int] Number of epochs for cut the training if not improves if 0, do not use it

Methods

__init__(data[, file_weights, best_F1, ...])	Initialize the keras callback
on_batch_begin(batch[, logs])	A backwards compatibility alias for *on_train_batch_begin*.
on_batch_end(batch[, logs])	A backwards compatibility alias for *on_train_batch_end*.
on_epoch_begin(epoch[, logs])	Called at the start of an epoch.
on_epoch_end(epoch[, logs])	This function is run when each epoch ends.
on_predict_batch_begin(batch[, logs])	Called at the beginning of a batch in *predict* methods.
on_predict_batch_end(batch[, logs])	Called at the end of a batch in *predict* methods.
on_predict_begin([logs])	Called at the beginning of prediction.
on_predict_end([logs])	Called at the end of prediction.
on_test_batch_begin(batch[, logs])	Called at the beginning of a batch in *evaluate* methods.
on_test_batch_end(batch[, logs])	Called at the end of a batch in *evaluate* methods.
on_test_begin([logs])	Called at the beginning of evaluation or validation.
on_test_end([logs])	Called at the end of evaluation or validation.
on_train_batch_begin(batch[, logs])	Called at the beginning of a training batch in *fit* methods.
on_train_batch_end(batch[, logs])	Called at the end of a training batch in *fit* methods.
on_train_begin([logs])	Called at the beginning of training.
on_train_end([logs])	Called at the end of training.
set_model(model)	
set_params(params)	

on_batch_begin(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_begin*.

on_batch_end(*batch*, *logs=None*)
A backwards compatibility alias for *on_train_batch_end*.

on_epoch_begin(*epoch*, *logs=None*)
Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Arguments: epoch: Integer, index of epoch. logs: Dict. Currently no data is passed to this argument for this method

but that may change in the future.

on_epoch_end(*epoch*, *logs={}*)
This function is run when each epoch ends. The metrics are calculated, printed and saved to the log file.

Parameters

epoch [int] number of epoch (from Callback class)

logs [dict] log data (from Callback class)

on_predict_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.predict_step*,

it typically returns a dict with a key 'outputs' containing the model's outputs.

on_predict_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_predict_begin(*logs=None*)
Called at the beginning of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_predict_end(*logs=None*)
Called at the end of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.test_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_test_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_test_begin(*logs=None*)
Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_end(*logs=None*)
Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_test_batch_end()* is passed to this argument for this method but that may change in the future.

on_train_batch_begin(*batch*, *logs=None*)
Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.train_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_train_batch_end(*batch*, *logs=None*)
Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_train_begin(*logs=None*)
Called at the beginning of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_train_end(*logs=None*)
Called at the end of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_epoch_end()* is passed to this argument for this method but that may change in the future.

set_model(*model*)

set_params(*params*)

9.5.3 dcase_models.util.TaggingCallback

class dcase_models.util.**TaggingCallback**(*data*, *file_weights=None*, *best_F1=0*, *early_stopping=0*, *considered_improvement=0.01*, *label_list=[]*)
Bases: tensorflow.python.keras.callbacks.Callback

Keras callback to calculate acc after each epoch and save file with the weights if the evaluation improves

__init__(*data*, *file_weights=None*, *best_F1=0*, *early_stopping=0*, *considered_improvement=0.01*, *label_list=[]*)
Initialize the keras callback

Parameters

data [tuple or KerasDataGenerator] Validation data for model evaluation (X_val, Y_val) or KerasDataGenerator

file_weights [string] Path to the file with the weights

best_acc [float] Last accuracy value, only if continue

early_stopping [int] Number of epochs for cut the training if not improves if 0, do not use it

Methods

__init__(data[, file_weights, best_F1, ...])	Initialize the keras callback
on_batch_begin(batch[, logs])	A backwards compatibility alias for *on_train_batch_begin*.

Continued on next page

Table 8 – continued from previous page

on_batch_end(batch[, logs])	A backwards compatibility alias for *on_train_batch_end*.
on_epoch_begin(epoch[, logs])	Called at the start of an epoch.
on_epoch_end(epoch[, logs])	This function is run when each epoch ends.
on_predict_batch_begin(batch[, logs])	Called at the beginning of a batch in *predict* methods.
on_predict_batch_end(batch[, logs])	Called at the end of a batch in *predict* methods.
on_predict_begin([logs])	Called at the beginning of prediction.
on_predict_end([logs])	Called at the end of prediction.
on_test_batch_begin(batch[, logs])	Called at the beginning of a batch in *evaluate* methods.
on_test_batch_end(batch[, logs])	Called at the end of a batch in *evaluate* methods.
on_test_begin([logs])	Called at the beginning of evaluation or validation.
on_test_end([logs])	Called at the end of evaluation or validation.
on_train_batch_begin(batch[, logs])	Called at the beginning of a training batch in *fit* methods.
on_train_batch_end(batch[, logs])	Called at the end of a training batch in *fit* methods.
on_train_begin([logs])	Called at the beginning of training.
on_train_end([logs])	Called at the end of training.
set_model(model)	
set_params(params)	

on_batch_begin(*batch*, *logs=None*)
A backwards compatibility alias for *on_train_batch_begin*.

on_batch_end(*batch*, *logs=None*)
A backwards compatibility alias for *on_train_batch_end*.

on_epoch_begin(*epoch*, *logs=None*)
Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Arguments: epoch: Integer, index of epoch. logs: Dict. Currently no data is passed to this argument for this method

but that may change in the future.

on_epoch_end(*epoch*, *logs={}*)
This function is run when each epoch ends. The metrics are calculated, printed and saved to the log file.

Parameters

epoch [int] number of epoch (from Callback class)

logs [dict] log data (from Callback class)

on_predict_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.predict_step*,

it typically returns a dict with a key 'outputs' containing the model's outputs.

on_predict_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_predict_begin(*logs=None*)
Called at the beginning of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_predict_end(*logs=None*)
Called at the end of prediction.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_batch_begin(*batch*, *logs=None*)
Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.test_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_test_batch_end(*batch*, *logs=None*)
Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_test_begin(*logs=None*)
Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_test_end(*logs=None*)
Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_test_batch_end()* is passed to this argument for this method but that may change in the future.

on_train_batch_begin(*batch*, *logs=None*)
Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict, contains the return value of *model.train_step*. Typically,

the values of the *Model*'s metrics are returned. Example: *{'loss': 0.2, 'accuracy': 0.7}*.

on_train_batch_end(*batch*, *logs=None*)
Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Arguments: batch: Integer, index of batch within the current epoch. logs: Dict. Aggregated metric results up until this batch.

on_train_begin(*logs=None*)
Called at the beginning of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently no data is passed to this argument for this method but that may change in the future.

on_train_end(*logs=None*)
Called at the end of training.

Subclasses should override for any actions to run.

Arguments:

logs: Dict. Currently the output of the last call to *on_epoch_end()* is passed to this argument for this method but that may change in the future.

set_model(*model*)

set_params(*params*)

9.6 GUI functions

encode_audio(data, sr)	Encode an audio signal for web applications.

9.6.1 dcase_models.util.encode_audio

dcase_models.util.**encode_audio**(*data*, *sr*)

Encode an audio signal for web applications.

Parameters

data [array] Audio signal.

sr [int] Sampling rate

Returns

str Encoded audio signal.

9.7 UI functions

progressbar(it[, prefix, size, file])	Iterable progress bar.

9.7.1 dcase_models.util.progressbar

dcase_models.util.**progressbar**(*it*, *prefix=''*, *size=60*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Iterable progress bar.

9.8 Miscellaneous functions

get_class_by_name(classes_dict, class_name, ...)	Get a class given its name.

9.8.1 dcase_models.util.get_class_by_name

dcase_models.util.**get_class_by_name**(*classes_dict*, *class_name*, *default*)

Get a class given its name.

Parameters

classes_dict [dict] Dict with the form {class_name: class}

class_name [str] Class name.

default: class Class to be used if class_name is not in classes_dict.

Returns

Class Class with name class_name

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

V

W