### DCASE2022 Task 4 Baseline for Sound Event Detection in Domestic Environments.

---

## Requirements

The script `conda\_create\_environment.sh` is available to create an environment which runs the

following code (recommended to run line by line in case of problems).

#### Common issues

\*\*Data Download\*\*

`FileNotFoundError: [Errno 2] No such file or directory: 'ffprobe'`

it probably means you have to install ffmpeg on your machine.

A possible installation: `sudo apt install ffmpeg`

\*\*Training\*\*

If training appears too slow, check with `top` and with `nvidia-smi` that you

are effectively using a GPU and not the CPU.

If running `python train\_sed.py` uses by default the CPU you may have \*\*pytorch\*\* installed

without CUDA support.

Check with IPython by running this pytorch line `torch.rand((1)).cuda()`

If you encounter an error install CUDA-enabled pytorch from https://pytorch.org/

Check again till you can run `torch.rand((1)).cuda()` successfully.

## Dataset

You can download the development dataset using the script: `generate\_dcase\_task4\_2022.py`.

The development dataset is composed of two parts:

- real-world data ([DESED dataset][desed]): this part of the dataset is composed of strong labels, weak labels, unlabeled, and validation data which are coming from [Audioset][audioset].

- synthetically generated data: this part of the dataset is composed of synthetically soundscapes, generated using [Scaper][scaper].

### Usage:

Run the command `python generate\_dcase\_task4\_2022.py --basedir="../../data"` to download the dataset (the user can change basedir to the desired data folder.)

If the user already has downloaded part of the dataset, it does not need to re-download the whole set. It is possible to download only part of the full dataset, if needed, using the options:

- \*\*only\_strong\*\* (download only the strong labels of the DESED dataset)

- \*\*only\_real\*\* (download the weak labels, unlabeled and validation data of the DESED dataset)

- \*\*only\_synth\*\* (download only the synthetic part of the dataset)

For example, if the user already has downloaded the real and synthetic part of the set, it can integrate the dataset with the strong labels of the DESED dataset with the following command:

`python generate\_dcase\_task4\_2022.py --only\_strong`

If the user wants to download only the synthetic part of the dataset, it could be done with the following command:

`python generate\_dcase\_task4\_2022.py --only\_synth`

Once the dataset is downloaded, the user should find the folder \*\*missing\_files\*\*, containing the list of files from the real-world dataset (desed\_real) which was not possible to download. You need to download it and \*\*send your missing files to the task

organisers to get the complete dataset\*\* (in priority to Francesca Ronchini and Romain serizel).

### Development dataset

The dataset is composed by 4 different splits of training data:

- Synthetic training set with strong annotations

- Strong labeled training set \*\*(only for the SED Audioset baseline)\*\*

- Weak labeled training set

- Unlabeled in domain training set

#### Synthetic training set with strong annotations

This set is composed of \*\*10000\*\* clips generated with the [Scaper][scaper] soundscape synthesis and augmentation library. The clips are generated such that the distribution per event is close to that of the validation set.

The strong annotations are provided in a tab separated csv file under the following format:

`[filename (string)][tab][onset (in seconds) (float)][tab][offset (in seconds) (float)][tab][event\_label (string)]`

For example: YOTsn73eqbfc\_10.000\_20.000.wav 0.163 0.665 Alarm\_bell\_ringing

#### Strong labeled training set

This set is composed of \*\*3470\*\* audio clips coming from [Audioset][audioset].

\*\*This set is used at training only for the SED Audioset baseline.\*\*

The strong annotations are provided in a tab separated csv file under the following format:

`[filename (string)][tab][onset (in seconds) (float)][tab][offset (in seconds) (float)][tab][event\_label (string)]`

For example: Y07fghylishw\_20.000\_30.000.wav 0.163 0.665 Dog

#### Weak labeled training set

This set contains \*\*1578\*\* clips (2244 class occurrences) for which weak annotations have been manually verified for a small subset of the training set.

The weak annotations are provided in a tab separated csv file under the following format:

`[filename (string)][tab][event\_labels (strings)]`

For example: Y-BJNMHMZDcU\_50.000\_60.000.wav Alarm\_bell\_ringing,Dog

#### Unlabeled in domain training set

This set contains \*\*14412\*\* clips. The clips are selected such that the distribution per class (based on Audioset annotations) is close to the distribution in the labeled set. However, given the uncertainty on Audioset labels, this distribution might not be exactly similar.

The dataset uses [FUSS][fuss\_git], [FSD50K][FSD50K], [desed\_soundbank][desed] and [desed\_real][desed].

For more information regarding the dataset, please refer to the [previous year DCASE Challenge website][dcase\_21\_dataset].

## Training

We provide \*\*three\*\* baselines for the task:

- SED baseline

- baseline using pre-trained embedding extractor DNN.

- baseline using Audioset data (real-world strong-label data)

For now, only the SED baseline is available (the missing baseline will be published soon).

### How to run the Baseline systems

The \*\*SED baseline\*\* can be run from scratch using the following command:

`python train\_sed.py`

---

\*\*NOTE: Currently multi-GPUs is not supported\*\*

\*\*note\*\*: `python train\_sed.py --gpus 0` will use the CPU. GPU indexes start from 1 here.

\*\*Common issues\*\*

If you encounter:

`pytorch\_lightning.utilities.exceptions.MisconfigurationException: You requested GPUs: [0]

But your machine only has: [] (edited) `

or

`OSError: libc10\_cuda.so: cannot open shared object file: No such file or directory`

It probably means you have installed CPU-only version of Pytorch or have installed the incorrect

\*\*cudatoolkit\*\* version.

Please install the correct version from https://pytorch.org/

---

Note that the default training config will use GPU 0.

Alternatively, we provide a [pre-trained checkpoint][zenodo\_pretrained\_models] along with tensorboard logs. The baseline can be tested on the development set of the dataset using the following command:

`python train\_sed.py --test\_from\_checkpoint /path/to/downloaded.ckpt`

The tensorboard logs can be tested using the command `tensorboard --logdir="path/to/exp\_folder"`.

## \*\*(NEW!)\*\* Energy Consumption

In this year DCASE Task 4 Challenge, we also use energy consumption (kWh)

via [CodeCarbon](https://github.com/mlco2/codecarbon) as an additional metric to rank the submitted systems.

We encourage the participants to provide, for each submitted system (or at least the best one), the following energy consumption figures in kWh using [CodeCarbon](https://github.com/mlco2/codecarbon):

1) whole system training

2) devtest inference

3) evaluation set inference

You can refer to [Codecarbon](https://github.com/mlco2/codecarbon) on how to do this (super simple! 😉 )

or to this baseline code see `local/sed\_trained.py` for some hints on

how we are doing this for the baseline system.

\*\*Important\*\*

In addition to this, we kindly suggest the participants to

provide the energy consumption in kWh (using the same hardware used for 2) and 3)) of:

1) devtest inference for baseline system using:

`python train\_sed.py --test\_from\_checkpoint /path/to/downloaded.ckpt`

You can find the energy consumed in kWh in `./exp/2022\_baseline/devtest\_codecarbon/devtest\_tot\_kwh.txt`

2) evaluation set inference for baseline system using:

`python train\_sed.py --eval\_from\_checkpoint /path/to/downloaded.ckpt`

You can find the energy consumed in kWh in `./exp/2022\_baseline/evaluation\_codecarbon/eval\_tot\_kwh.txt`

\*\*Why we require this ?\*\*

Energy consumption depends on hardware and each participant uses

different hardware.

To obviate for this difference we use the baseline inference kWh energy consumption

as a common reference. Because of this, it is important that the

inference energy consumption figures for both submitted system

and baseline are computed on same hardware under similar loading.

#### Results:

Dataset | \*\*PSDS-scenario1\*\* | \*\*PSDS-scenario2\*\* | \*Intersection-based F1\* | \*Collar-based F1\*

--------|--------------------|--------------------|-------------------------|-----------------

Dev-test| \*\*0.336\*\* | \*\*0.536\*\* | 64.1% | 40.1%

\*\*Energy Consumption\*\* (GPU: NVIDIA A100 40Gb)

Dataset | Training | Dev-Test |

--------|-----------|--------------------

\*\*kWh\*\* | \*\*1.717\*\* | \*\*0.030\*\*

Collar-based = event-based. More information about the metrics in the DCASE Challenge [webpage][dcase22\_webpage].

The results are from the \*\*student\*\* predictions.

\*\*NOTES\*\*:

All baselines scripts assume that your data is in `../../data` folder in DESED\_task directory.

If your data is in another folder, you will have to change the paths of your data in the corresponding `data` keys in YAML configuration file in `conf/sed.yaml`.

Note that `train\_sed.py` will create (at its very first run) additional folders with resampled data (from 44kHz to 16kHz)

so the user need to have write permissions on the folder where your data are saved.

\*\*Hyperparameters\*\* can be changed in the YAML file (e.g. lower or higher batch size).

A different configuration YAML (for example sed\_2.yaml) can be used in each run using `--conf\_file="confs/sed\_2.yaml` argument.

The default directory for checkpoints and logging can be changed using `--log\_dir="./exp/2021\_baseline`.

Training can be resumed using the following command:

`python train\_sed.py --resume\_from\_checkpoint /path/to/file.ckpt`

In order to make a "fast" run, which could be useful for development and debugging, you can use the following command:

`python train\_sed.py --fast\_dev\_run`

It uses very few batches and epochs so it won't give any meaningful result.

\*\*Architecture\*\*

The baseline is the same as the [DCASE 2021 Task 4 baseline][dcase\_21\_repo], based on a Mean-Teacher model [1].

The baseline uses a Mean-Teacher model which is a combination of two models: a student model and a

teacher model, having the same architecture. The student model is the one used at inference while the goal of the teacher is to help the student model during training. The teacher's weight are the exponential average of the student model's weights. The models are a combination of a convolutional neural network (CNN) and a recurrent neural network (RNN) followed by an attention layer. The output of the RNN gives strong predictions while the output of the attention layer gives the weak predictions [2].

Figure 1 shows an illustration of the baseline model.

| ![This is an image](./img/mean\_teacher.png) |

|:--:|

| \*Figure 1: baseline Mean-teacher model. Adapted from [2].\* |

Mixup is used as data augmentation technique for weak and synthetic data by mixing data in a batch (50% chance of applying it) [3].

For more information regarding the baseline model, the reader is referred to [1] and [2].

### SED baseline using Audioset data (real-world strong-label data)

The SED baseline using the strongly annotated part of Audioset can be run from scratch using the following command:

`python train\_sed.py --strong\_real`

The command will automatically considered the strong labels recorded data coming from Audioset in the training process.

Alternatively, also in this case, we provide a [pre-trained checkpoint][zenodo\_pretrained\_audioset\_models]. The baseline can be tested on the development set of the dataset using the following command:

`python train\_sed.py --test\_from\_checkpoint /path/to/downloaded.ckpt`

#### Results:

Dataset | \*\*PSDS-scenario1\*\* | \*\*PSDS-scenario2\*\* | \*Intersection-based F1\* | \*Collar-based F1\*

--------|--------------------|--------------------|-------------------------|-----------------

Dev-test| \*\*0.351\*\* | \*\*0.552\*\* | 64.3% | 42.9%

\*\*Energy Consumption\*\* (GPU: NVIDIA A100 40Gb)

Dataset | Training | Dev-Test |

--------|-----------|--------------------

\*\*kWh\*\* | \*\*2.418\*\* | \*\*0.027\*\*

Collar-based = event-based. More information about the metrics in the DCASE Challenge [webpage][dcase22\_webpage].

The results are computed from the \*\*student\*\* predictions.

All the comments related to the possibility of resuming the training and the fast development run in the [SED baseline][sed\_baseline] are valid also in this case.

## \*\*(NEW)\*\* baseline using pre-trained embeddings from models (SEC/Tagging) trained on Audioset

We added a new baseline which exploits pre-trained models such as [PANNs](https://arxiv.org/abs/1912.10211) and [AST](https://arxiv.org/abs/2104.01778) to increase the performance.

to increase the performance.

In this baseline the frame-level or whole-clip level features are used in a late-fusion fashion

with the existing CRNN baseline classifier.

See `desed\_task/nnet/CRNN.py` for details. The whole-clip features are concatenated with CNN extracted features in the baseline

CRNN classifier.

Regarding he frame-level features, since they have different sequence length w.r.t. CNN features

we use a trainable RNN-based encoder to encode those to a fixed dim output (obtaining again a whole-clip level embedding).

This embedding is then concatenated in the same way as the whole-clip features.

\*\*We provide different ways to integrate such pre-trained models.\*\*

See the configuration file: `./confs/pretrained.yaml`:

```yaml

pretrained:

model: ast

e2e: False

freezed: True

url: https://zenodo.org/record/3987831/files/Cnn14\_16k\_mAP%3D0.438.pth?download=1

dest: ./pretrained\_models/Cnn14\_16k\_mAP%3D0.438.pth

extracted\_embeddings\_dir: ./embeddings

```

You can choose \*\*ast\*\* or \*\*panns\*\*.

You can choose whether to keep the pre-trained model \*\*freezed\*\* or train it along with the CRNN architecture.

If you want to keep it freezed, we already provide the pre-extracted embeddings for you.

This is useful if you want to train with a big batch size because you won't have to store the rather heavy

PANNs or AST models on your GPU.

Here are the links to the pre-extracted embeddings for AST and PANNs:

https://zenodo.org/record/6541454#.YnzHq2YzbDI (unalabeled ast)

https://zenodo.org/record/6539466#.YnvtWmYzbAM (ast synth train, ast synth val,ast weak val)

https://zenodo.org/record/6518380#.YnvWZGYzbAM (panns, ast weak train, ast devtest)

You can download and unpack them in your preferred directory.

Do not forget then to set in the configuration

above `extracted\_embeddings\_dir: YOUR\_PATH`.

The script expects a folder structure like this:

```

YOUR\_PATH |--- ast

|---- devtest.hdf5

|---- synth\_train.hdf5

|---- unlabeled\_train.hdf5

|---- weak\_train.hdf5

|---- weak\_val.hdf5

|---- synth\_val.hdf5

|--- panns

|---- devtest.hdf5

|---- synth\_train.hdf5

|---- unlabeled\_train.hdf5

|---- weak\_train.hdf5

|---- weak\_val.hdf5

|---- synth\_val.hdf5

```

You can also select if you want to do late fusion with global, whole-clip features from PANNs or

frame-level features in `./confs/pretrained.yaml`:

```yaml

nb\_filters: [ 16, 32, 64, 128, 128, 128, 128 ]

pooling: [ [ 2, 2 ], [ 2, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ] ]

dropout\_recurrent: 0

use\_embeddings: True

embedding\_size: 768 # use 2048 for PANNs global and frame, 527 for AST global and 768 for AST frame

embedding\_type: frame # or global

```

\*\*The training can be started simply with\*\*

`python train\_pretrained.py`

By default this uses AST with frame-level embeddings. The pre-trained model is freezed and expects the pre-extracted AST

embeddings in a local folder `./embeddings` as you can see from the details provided before about the YAML config.

Thus you would need to download the AST embeddings from the Zenodo links above, unless you set `freezed: False`.

However, the latter requires significant GPU memory.

Also in this case, we provide a [pre-trained checkpoint][zenodo\_pretrained\_audioset\_models]. The baseline can be tested on the development set of the dataset using the following command:

`python train\_pretrained.py --test\_from\_checkpoint /path/to/downloaded.ckpt`

#### Results for best system, late fusion with AST frame:

Dataset | \*\*PSDS-scenario1\*\* | \*\*PSDS-scenario2\*\* | \*Intersection-based F1\* | \*Collar-based F1\*

--------|------------|--------------------|-------------------------|-----------------

Dev-test| \*\*32.24%\*\* | \*\*72.22%\*\* | \*\*90.34\*\* | \*\*37.16\*\*

\*\*Energy Consumption\*\* (GPU: NVIDIA A100 40Gb)

\*\*Note we used pre-extracted embeddings, so the power consuption for the pre-trained model is not accounted for\*\*

Dataset | Training | Dev-Test |

--------|----------|--------------------

\*\*kWh\*\* | \*\*4.41\*\* | \*\*0.036\*\*

Collar-based = event-based. More information about the metrics in the DCASE Challenge [webpage][dcase22\_webpage].

The results are computed from the \*\*teacher\*\* predictions.

All the comments related to the possibility of resuming the training and the fast development run in the [SED baseline][sed\_baseline] are valid also in this case.

\*\*Architecture\*\*

The architecture of the SED Audioset baseline is the same as the [SED baseline][sed\_baseline].

[audioset]: https://research.google.com/audioset/

[dcase22\_webpage]: https://dcase.community/challenge2022/task-sound-event-detection-in-domestic-environments

[dcase\_21\_repo]: https://github.com/DCASE-REPO/DESED\_task/tree/master/recipes/dcase2021\_task4\_baseline

[dcase\_21\_dataset]: https://dcase.community/challenge2021/task-sound-event-detection-and-separation-in-domestic-environments#audio-dataset

[desed]: https://github.com/turpaultn/DESED

[fuss\_git]: https://github.com/google-research/sound-separation/tree/master/datasets/fuss

[fsd50k]: https://zenodo.org/record/4060432

[zenodo\_pretrained\_models]: https://zenodo.org/record/4639817

[zenodo\_pretrained\_audioset\_models]: https://zenodo.org/record/6447197

[zenodo\_pretrained\_ast\_embedding\_model]:

[google\_sourcesep\_repo]: https://github.com/google-research/sound-separation/tree/master/datasets/yfcc100m

[sdk\_installation\_instructions]: https://cloud.google.com/sdk/docs/install

[zenodo\_evaluation\_dataset]: https://zenodo.org/record/4892545#.YMHH\_DYzadY

[scaper]: https://github.com/justinsalamon/scaper

[sed\_baseline]: https://github.com/DCASE-REPO/DESED\_task/tree/master/recipes/dcase2022\_task4\_baseline#sed-baseline

#### References

[1] L. Delphin-Poulat & C. Plapous, technical report, dcase 2019.

[2] Turpault, Nicolas, et al. "Sound event detection in domestic environments with weakly labeled data and soundscape synthesis."

[3] Zhang, Hongyi, et al. "mixup: Beyond empirical risk minimization." arXiv preprint arXiv:1710.09412 (2017).

[4] Thomee, Bart, et al. "YFCC100M: The new data in multimedia research." Communications of the ACM 59.2 (2016)

[5] Wisdom, Scott, et al. "Unsupervised sound separation using mixtures of mixtures." arXiv preprint arXiv:2006.12701 (2020).

[6] Turpault, Nicolas, et al. "Improving sound event detection in domestic environments using sound separation." arXiv preprint arXiv:2007.03932 (2020).

[7] Ronchini, Francesca, et al. "The impact of non-target events in synthetic soundscapes for sound event detection." arXiv preprint arXiv:2109.14061 (DCASE2021)

[8] Ronchini, Francesca, et al. "A benchmark of state-of-the-art sound event detection systems evaluated on synthetic soundscapes." arXiv preprint arXiv:2202.01487