OpenL3 Documentation

Release 0.4.2-dev

Jason Cramer, Ho-Hsiang Wu, and Justin Salamon

Aug 09, 2021

Contents

CHAPTER 1

Installation instructions

1.1 Dependencies

1.1.1 libsndfile

OpenL3 depends on the `pysoundfile` module to load audio files, which depends on the non-Python library `libsndfile`. On Windows and macOS, these will be installed via `pip` and you can therefore skip this step. However, on Linux this must be installed manually via your platform's package manager. For Debian-based distributions (such as Ubuntu), this can be done by simply running

```
>>> apt-get install libsndfile1
```

Alternatively, if you are using `conda`, you can install `libsndfile` simply by running

```
>>> conda install -c conda-forge libsndfile
```

For more detailed information, please consult the pysoundfile installation documentation.

1.1.2 Tensorflow

Starting with `openl3>=0.4.0`, Openl3 has been upgraded to use Tensorflow 2. Because Tensorflow 2 and higher now includes GPU support, `tensorflow>=2.0.0` is included as a dependency and no longer needs to be installed separately.

If you are interested in using Tensorflow 1.x, please install using `pip install 'openl3<=0.3.1'`.

Tensorflow 1x & OpenL3 <= v0.3.1

Because Tensorflow 1.x comes in CPU-only and GPU variants, we leave it up to the user to install the version that best fits their usecase.

On most platforms, either of the following commands should properly install Tensorflow:

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

1.2 Installing OpenL3

The simplest way to install OpenL3 is by using `pip`, which will also install the additional required dependencies if needed. To install OpenL3 using `pip`, simply run

```
>>> pip install openl3
```

To install the latest version of OpenL3 from source:

1. Clone or pull the latest version, only retrieving the `main` branch to avoid downloading the branch where we store the model weight files (these will be properly downloaded during installation).

```
>>> git clone git@github.com:marl/openl3.git --branch main --single-branch
```

2. Install using pip to handle python dependencies. The installation also downloads model files, which requires a stable network connection.

```
>>> cd openl3
>>> pip install -e .
```

CHAPTER 2

OpenL3 tutorial

2.1 Introduction

Welcome to the OpenL3 tutorial! In this tutorial, we'll show how to use OpenL3 to compute embeddings for your audio files, images, and videos. The supported audio formats are those supported by the `pysoundfile` library, which is used for loading the audio (e.g. WAV, OGG, FLAC). The supported image formats are those supported by the *scikit-image*' library (e.g. PNG, JPEG). The supported video formats are those supported by `moviepy` (and therefore `ffmpeg`) (e.g. MP4).

2.2 Using the Library

2.2.1 Extracting audio embeddings

You can easily compute audio embeddings out of the box, like so:

```
import openl3
import soundfile as sf

audio, sr = sf.read('/path/to/file.wav')
emb, ts = openl3.get_audio_embedding(audio, sr)
```

`get_audio_embedding` returns two objects. The first object `emb` is a T-by-D numpy array, where T is the number of embedding frames and D is the dimensionality of the embedding (which can be either 6144 or 512, details below). The second object `ts` is a length-T numpy array containing timestamps corresponding to each embedding frame (each timestamp corresponds to the center of each analysis window by default).

Multiple audio arrays can be provided (with a specified model input batch size for processing, i.e. the number of audio windows the model processes at a time) as follows:

```
import openl3
import soundfile as sf
```

(continues on next page)

(continued from previous page)

```
audio1, sr1 = sf.read('/path/to/file1.wav')
audio2, sr2 = sf.read('/path/to/file2.wav')
audio3, sr3 = sf.read('/path/to/file3.wav')
audio_list = [audio1, audio2, audio3]
sr_list = [sr1, sr2, sr3]

# Pass in a list of audio arrays and sample rates
emb_list, ts_list = openl3.get_audio_embedding(audio_list, sr_list, batch_size=32)
# If all arrays use sample rate, can just pass in one sample rate
emb_list, ts_list = openl3.get_audio_embedding(audio_list, sr1, batch_size=32)
```

Here, we get a list of embeddings and timestamp arrays for each of the input arrays.

By default, OpenL3 extracts audio embeddings with a model that:

- Is trained on AudioSet videos containing mostly musical performances
- Uses a mel-spectrogram time-frequency representation with 128 bands
- Returns an embedding of dimensionality 6144 for each embedding frame

These defaults can be changed via the following optional parameters:

- content_type: “env”, “music” (default)
- input_repr: “linear”, “mel128” (default), “mel256”
- embedding_size: 512, 6144 (default)

For example, the following code computes an embedding using a model trained on environmental videos using a spectrogram with a linear frequency axis and an embedding dimensionality of 512:

```
emb, ts = openl3.get_audio_embedding(audio, sr, content_type="env",
                                     input_repr="linear", embedding_size=512)
```

By default OpenL3 will pad the beginning of the input audio signal by 0.5 seconds (half of the window size) so that the the center of the first window corresponds to the beginning of the signal (“zero centered”), and the returned timestamps correspond to the center of each window. You can disable this centering like this:

```
emb, ts = openl3.get_audio_embedding(audio, sr, center=False)
```

The hop size used to extract the embedding is 0.1 seconds by default (i.e. an embedding frame rate of 10 Hz). In the following example we change the hop size from 0.1 (10 frames per second) to 0.5 (2 frames per second):

```
emb, ts = openl3.get_audio_embedding(audio, sr, hop_size=0.5)
```

Finally, you can silence the Keras printout during inference (verbosity) by changing it from 1 (default) to 0:

```
emb, ts = openl3.get_audio_embedding(audio, sr, verbose=0)
```

By default, the model file is loaded from disk every time `get_audio_embedding` is called. To avoid unnecessary I/O when processing multiple files with the same model, you can load it manually and pass it to the function via the `model` parameter:

```
model = openl3.models.load_audio_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)
emb1, ts1 = openl3.get_audio_embedding(audio1, sr1, model=model)
emb2, ts2 = openl3.get_audio_embedding(audio2, sr2, model=model)
```

Note that when a model is provided via the `model` parameter any values passed to the `input_repr`, `content_type` and `embedding_size` parameters of `get_audio_embedding` will be ignored.

To compute embeddings for an audio file and directly save them to disk you can use `process_audio_file`:

```
import openl3
import numpy as np

audio_filepath = '/path/to/file.wav'

# Save the embedding to '/path/to/file.npz'
openl3.process_audio_file(audio_filepath)

# Save the embedding to `/path/to/file_suffix.npz`
openl3.process_audio_file(audio_filepath, suffix='suffix')

# Save the embedding to '/different/dir/file_suffix.npz'
openl3.process_audio_file(audio_filepath, suffix='suffix', output_dir='/different/dir
↪')
```

The embddings can be loaded from disk using numpy:

```
import numpy as np

data = np.load('/path/to/file.npz')
emb, ts = data['embedding'], data['timestamps']
```

Multiple files can be processed as well (with a specified model input batch size used for processing, i.e. the number of audio windows the model processes at a time) as follows:

```
import openl3
import numpy as np

audio_filepath1 = '/path/to/file1.wav'
audio_filepath2 = '/path/to/file2.wav'
audio_filepath3 = '/path/to/file3.wav'
audio_filepath_list = [audio_filepath1, audio_filepath2, audio_filepath3]

# Saves embeddings to '/path/to/file1.npz', '/path/to/file2.npz', and '/path/to/file3.
↪npz'
openl3.process_audio_file(audio_filepath_list, batch_size=32)
```

As with `get_audio_embedding`, you can load the model manually and pass it to `process_audio_file` to avoid loading the model multiple times:

```
import openl3
import numpy as np

model = openl3.models.load_audio_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)

audio_filepath = '/path/to/file.wav'

# Save the file to '/path/to/file.npz'
openl3.process_audio_file(audio_filepath, model=model)

# Save the file to `/path/to/file_suffix.npz`
```

(continues on next page)

(continued from previous page)

```
openl3.process_audio_file(audio_filepath, model=model, suffix='suffix')

# Save the file to '/different/dir/file_suffix.npz'
openl3.process_audio_file(audio_filepath, model=model, suffix='suffix', output_dir='/
↪different/dir')
```

Again, note that if a model is provided via the `model` parameter, then any values passed to the `input_repr`, `content_type` and `embedding_size` parameters of `process_audio_file` will be ignored.

2.2.2 Extracting image embeddings

You can easily compute image embeddings out of the box, like so:

```
import openl3
from skimage.io import imread

image = imread('/path/to/file.png)
emb = openl3.get_image_embedding(image, content_type="env",
                                 input_repr="linear", embedding_size=512)

# Preload model
model = openl3.models.load_image_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)
emb = openl3.get_image_embedding(image, model=model)
```

When given a single image, `get_image_embedding` returns a size D numpy array, where D is the dimensionality of the embedding (which can be either 8192 or 512).

A sequence of images (e.g. from a video) can also be provided:

```
from moviepy.video.io.VideoFileClip import VideoFileClip

video_filepath = '/path/to/file.mp4'

# Load video and get image frames
clip = VideoFileClip(video_filepath)
images = np.array([frame for frame in clip.iter_frames()])

emb = get_image_embedding(images, batch_size=32)
```

When given a sequence of images (as a numpy array), `get_image_embedding` returns an N-by-D numpy array, where N is the number of embedding frames (corresponding to each video frame) and D is the dimensionality of the embedding (which can be either 8192 or 512).

```
from moviepy.video.io.VideoFileClip import VideoFileClip

video_filepath = '/path/to/file.mp4'

# Load video and get image frames
clip = VideoFileClip(video_filepath)
images = np.array([frame for frame in clip.iter_frames()])

# If the frame rate is provided, returns an array of timestamps
emb, ts = get_image_embedding(images, frame_rate=clip.fps, batch_size=32)
```

When given a sequence of images (as a numpy array) and a frame rate, `get_image_embedding` returns two objects. The first object `emb` is an N-by-D numpy array, where N is the number of embedding frames and D is the dimensionality of the embedding (which can be either 8192 or 512, details below). The second object `ts` is a length-N numpy array containing timestamps corresponding to each embedding frame (corresponding to each video frame).

Multiple sequences of images can be provided as well:

```
from moviepy.video.io.VideoFileClip import VideoFileClip

video_filepath1 = '/path/to/file1.mp4'
video_filepath2 = '/path/to/file2.mp4'
video_filepath3 = '/path/to/file2.mp4'

# Load video and get image frames
clip1 = VideoFileClip(video_filepath1)
images1 = np.array([frame for frame in clip1.iter_frames()])
clip2 = VideoFileClip(video_filepath2)
images2 = np.array([frame for frame in clip2.iter_frames()])
clip3 = VideoFileClip(video_filepath3)
images3 = np.array([frame for frame in clip3.iter_frames()])

image_list = [images1, images2, images3]
frame_rate_list = [clip1.fps, clip2.fps, clip3.fps]

# If the frame rates is provided...
emb_list, ts_list = get_image_embedding(image_list, frame_rate=frame_rate_list, batch_
↪size=32)
# or if a single frame rate applying to all sequences is provided, returns a list of␣
↪timestamps
emb_list, ts_list = get_image_embedding(image_list, frame_rate=clip1.fps, batch_
↪size=32)
# ...otherwise, just the embeddings are returnedj
emb_list = get_image_embedding(image_list, batch_size=32)
```

Here, we get a list of embeddings for each sequence. If a frame rate (or list of frame rates) is given, timestamp arrays for each of the input arrays are returned as well.

By default, OpenL3 extracts image embeddings with a model that:

- Is trained on AudioSet videos containing mostly musical performances
- Is trained using a mel-spectrogram time-frequency representation with 128 bands (for the audio embedding model)
- Returns an embedding of dimensionality 8192 for each embedding frame

These defaults can be changed via the following optional parameters:

- content_type: “env”, “music” (default)
- input_repr: “linear”, “mel128” (default), “mel256”
- embedding_size: 512, 8192 (default)

For example, the following code computes an embedding using a model trained on environmental videos using an audio spectrogram with a linear frequency axis and an embedding dimensionality of 512:

```
emb = openl3.get_image_embedding(image, content_type="env",
                                 input_repr="linear", embedding_size=512)
```

Finally, you can silence the Keras printout during inference (verbosity) by changing it from 1 (default) to 0:

```
emb = openl3.get_image_embedding(image, verbose=0)
```

By default, the model file is loaded from disk every time `get_image_embedding` is called. To avoid unnecessary I/O when processing multiple files with the same model, you can load it manually and pass it to the function via the `model` parameter:

```
model = openl3.models.load_image_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)
emb1 = openl3.get_image_embedding(image1, model=model)
emb2 = openl3.get_image_embedding(image2, model=model)
```

Note that when a model is provided via the `model` parameter any values passed to the `input_repr`, `content_type` and `embedding_size` parameters of `get_image_embedding` will be ignored.

Image files can also be processed with just the filepath:

```
import openl3
import numpy as np

image_filepath = '/path/to/file.png'

# Save the file to '/path/to/file.npz'
openl3.process_image_file(image_filepath)

# Preload model
model = openl3.models.load_image_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)
openl3.process_image_file(image_filepath, model=model)

# Save the file to `/path/to/file_suffix.npz`
openl3.process_image_file(image_filepath, model=model, suffix='suffix')

# Save the file to '/different/dir/file_suffix.npz'
openl3.process_image_file(image_filepath, model=model, suffix='suffix', output_dir='/
↪different/dir')
```

The embdding can be loaded from disk using numpy:

```
import numpy as np

data = np.load('/path/to/file.npz')
emb = data['embedding']
```

Multiple files can be processed as well (with a specified model input batch size used for processing, i.e. the number of audio windows the model processes at a time) as follows:

```
import openl3
import numpy as np

image_filepath1 = '/path/to/file1.png'
image_filepath2 = '/path/to/file2.png'
image_filepath3 = '/path/to/file3.png'
image_filepath_list = [image_filepath1, image_filepath2, image_filepath3]

# Saves embeddings to '/path/to/file1.npz', '/path/to/file2.npz', and '/path/to/file3.
↪npz'
```

(continues on next page)

(continued from previous page)

```
openl3.process_image_file(image_filepath_list, batch_size=32)
```

As with `get_image_embedding`, you can load the model manually and pass it to `process_image_file` to avoid loading the model multiple times:

```
import openl3
import numpy as np

model = openl3.models.load_image_embedding_model(input_repr="mel256", content_type=
↪"music",
                                                 embedding_size=512)

image_filepath = '/path/to/file.png'

# Save the file to '/path/to/file.npz'
openl3.process_image_file(image_filepath, model=model)

# Save the file to `/path/to/file_suffix.npz`
openl3.process_image_file(image_filepath, model=model, suffix='suffix')

# Save the file to '/different/dir/file_suffix.npz'
openl3.process_image_file(image_filepath, model=model, suffix='suffix', output_dir='/
↪different/dir')
```

Again, note that if a model is provided via the `model` parameter, then any values passed to the `input_repr`, `content_type` and `embedding_size` parameters of `process_image_file` will be ignored.

2.2.3 Processing video files

Video files can be processed to extract both audio and image embeddings. Please note that the audio and image embeddings are not synchronized, so the respective timestamps for each modality will not generally be aligned. Image embeddings are computed for every frame of the video, while the specified audio hop size is used for chunking the audio signal in the video. Additionally, please note that available embedding sizes differ for audio and image embeddings. Video files can be processed as follows:

```
import openl3
import numpy as np

video_filepath = '/path/to/file.mp4'

# Save audio embedding to '/path/to/file_audio.npz'
# and image embedding to '/path/to/file_image.npz'
openl3.process_video_file(video_filepath)

# Preload models
audio_model = openl3.models.load_audio_embedding_model(input_repr="mel256", content_
↪type="music",
                                                       embedding_size=512)
image_model = openl3.models.load_image_embedding_model(input_repr="mel256", content_
↪type="music",
                                                       embedding_size=512)
openl3.process_video_file(video_filepath, audio_model=audio_model, image_model=image_
↪model)

# Save audio embedding to '/path/to/file_audio_suffix.npz'
```

(continues on next page)

(continued from previous page)

```
# and image embedding to '/path/to/file_image_suffix.npz'
openl3.process_video_file(video_filepath, audio_model=audio_model, image_model=image_
↪model,
                          suffix='suffix')

# Save audio embedding to '/different/dir/file_audio_suffix.npz'
# and image embedding to '/different/dir/file_image_suffix.npz'
openl3.process_video_file(video_filepath, audio_model=audio_model, image_model=image_
↪model,
                          suffix='suffix', output_dir='/different/dir')
```

Multiple files can be processed as well (with a specified model input batch size used for processing, i.e. the number of video frames the model processes at a time) as follows:

```
import openl3
import numpy as np

video_filepath1 = '/path/to/file1.mp4'
video_filepath2 = '/path/to/file2.mp4'
video_filepath3 = '/path/to/file3.mp4'
video_filepath_list = [video_filepath1, video_filepath2, video_filepath3]

# Saves audio embeddings to '/path/to/file1_audio.npz', '/path/to/file2_audio.npz',
# and '/path/to/file3_audio.npz' and saves image embeddings to
# '/path/to/file1_image.npz', '/path/to/file2_image.npz', and '/path/to/file3_image.
↪npz'
openl3.process_video_file(video_filepath_list, batch_size=32)
```

2.2.4 Choosing an Audio Frontend (CPU / GPU)

OpenL3 provides two different audio frontends to choose from.

Kapre (GPU)

Kapre implements audio processing as neural network layers, and computes the audio front-end (spectrogram or mel-spectrogram) as part of running model inference. This means that when a GPU is available, audio processing happens on the GPU. The OpenL3 Keras model takes batches of audio PCM as input `(shape=(None, 1, samples))`.

NOTE: due to updates in Kapre, the embeddings produced by OpenL3>=0.4.0 are not identical to those produced by previous versions (e.g. 0.3.1), even though both use Kapre for audio processing. If you are running inference with a model that was trained on OpenL3 embeddings, we strongly recommend using the same version of OpenL3 that was used to train that model.

```
import openl3
import soundfile as sf

audio, sr = sf.read('/path/to/file1.wav')

# get embedding using kapre frontend
emb_list, ts_list = openl3.get_audio_embedding(audio, sr)
emb_list, ts_list = openl3.get_audio_embedding(audio, sr, frontend='kapre')  # same␣
↪result

# pre-loading a model
```

(continues on next page)

(continued from previous page)

```
# load model with the kapre frontend
input_repr, content_type, embedding_size = 'mel128', 'music', 6144
model_kapre = openl3.models.load_audio_embedding_model(input_repr, content_type,␣
↪embedding_size)

# get embedding using pre-loaded model with kapre frontend
emb_list, ts_list = openl3.get_audio_embedding(audio, sr, model=model_kapre)

# computing embeddings manually for a single file.
# Note: If no input representation (`input_repr`) is passed to `preprocess_audio`
# then it will prepare the input data for the kapre frontend.

input_data = openl3.preprocess_audio(audio, sr)
emb, ts = model_kapre.predict(input_data)
```

Librosa (CPU)

Beginning with version 0.4.0, OpenL3 also provides support for computing audio features using Librosa, which offers you flexibility and allows you to precompute or parallelize your computations across multiple CPUs.

Contrary to the Kapre frontend, the Librosa frontend is not included as part of the neural network model. Instead the Keras model takes either linear or mel spectrograms (according to the input representation chosen). OpenL3's high-level interfaces still work the same though, it just changes the mechanics of how features are calculated under the hood.

This also decouples the feature extraction, which consists of specialized audio code that may not export well (e.g., to ONNX), from the standard convolutional and fully-connected layers that comprise the rest of OpenL3, making it easier to port the model to other neural network frameworks.

NOTE: `input_repr` is a required parameter to `openl3.get_audio_embedding` when using the librosa frontend, because the feature extraction steps are no longer compiled into the model.

```
import openl3
import soundfile as sf

audio, sr = sf.read('/path/to/file1.wav')

# get embedding using librosa frontend
emb_list, ts_list = openl3.get_audio_embedding(audio, sr, frontend='librosa')

# pre-loading a model

# load model with no frontend for use with an external librosa frontend
input_repr, content_type, embedding_size = 'mel128', 'music', 6144
model = openl3.models.load_audio_embedding_model(
    input_repr, content_type, embedding_size, frontend='librosa')

# get embedding using pre-loaded model and librosa frontend
emb_list, ts_list = openl3.get_audio_embedding(audio, sr, model=model, frontend=
↪'librosa')

# computing embeddings manually for a single file
# Note how we pass the input representation (`input_repr`) to `preprocess_audio` so␣
↪that
# librosa knows how to compute the inputs.
```

(continues on next page)

(continued from previous page)

```
input_data = openl3.preprocess_audio(audio, sr, input_repr=input_repr)
emb, ts = model.predict(input_data)
```

The same applies for video file processing, using `audio_frontend='librosa'`

```
# load model with no frontend for use with an external librosa frontend
input_repr, content_type, embedding_size = 'mel128', 'music', 6144
model = openl3.models.load_audio_embedding_model(
    input_repr, content_type, embedding_size, frontend='librosa')

openl3.process_video_file(video_fname, audio_model=model, audio_frontend='librosa')
```

2.3 Using the Command Line Interface (CLI)

2.3.1 Extracting audio embeddings

To compute embeddings for a single audio file via the command line run:

```
$ openl3 audio /path/to/file.wav
```

This will create an output file at `/path/to/file.npz`.

You can change the output directory as follows:

```
$ openl3 audio /path/to/file.wav --output /different/dir
```

This will create an output file at `/different/dir/file.npz`.

You can also provide multiple input files:

```
$ openl3 audio /path/to/file1.wav /path/to/file2.wav /path/to/file3.wav
```

which will create the output files `/different/dir/file1.npz`, `/different/dir/file2.npz`, and `different/dir/file3.npz`.

You can also provide one (or more) directories to process:

```
$ openl3 audio /path/to/audio/dir
```

This will process all supported audio files in the directory, though it will not recursively traverse the directory (i.e. audio files in subfolders will not be processed).

You can append a suffix to the output file as follows:

```
$ openl3 audio /path/to/file.wav --suffix somesuffix
```

which will create the output file `/path/to/file_somesuffix.npz`.

Arguments can also be provided to change the model used to extract the embedding including the content type used for training (music or env), input representation (linear, mel128, mel256), and output dimensionality (512 or 6144), for example:

```
$ openl3 audio /path/to/file.wav --content-type env --input-repr mel128 --embedding-
↪size 512
```

The default value for –content-type is music, for –input-repr is mel128 and for –embedding-size is 6144.

By default, OpenL3 will pad the beginning of the input audio signal by 0.5 seconds (half of the window size) so that the the center of the first window corresponds to the beginning of the signal, and the timestamps correspond to the center of each window. You can disable this centering as follows:

```
$ openl3 audio /path/to/file.wav --no-audio-centering
```

The hop size used to extract the embedding is 0.1 seconds by default (i.e. an embedding frame rate of 10 Hz). In the following example we change the hop size from 0.1 (10 frames per second) to 0.5 (2 frames per second):

```
$ openl3 audio /path/to/file.wav --audio-hop-size 0.5
```

You can change the batch size used as the input to the audio embedding model (i.e. the number of audio windows the model processes at a time) to one appropriate for your computational resources:

```
$ openl3 audio /path/to/file.wav --audio-batch-size 16
```

By default, the CLI will use the Kapre frontend to compute audio embeddings. You may also choose to use Librosa to compute the embeddings:

```
$ openl3 audio /path/to/file.wav --audio-frontend librosa
```

Finally, you can suppress non-error printouts by running:

```
$ openl3 audio /path/to/file.wav --quiet
```

2.3.2 Extracting image embeddings

To compute embeddings for a single image file via the command line run:

```
$ openl3 image /path/to/file.png
```

This will create an output file at `/path/to/file.npz`.

You can change the output directory as follows:

```
$ openl3 image /path/to/file.png --output /different/dir
```

This will create an output file at `/different/dir/file.npz`.

You can also provide multiple input files:

```
$ openl3 image /path/to/file1.png /path/to/file2.png /path/to/file3.png
```

which will create the output files `/different/dir/file1.npz`, `/different/dir/file2.npz`, and `different/dir/file3.npz`.

You can also provide one (or more) directories to process:

```
$ openl3 image /path/to/image/dir
```

This will process all supported image files in the directory, though it will not recursively traverse the directory (i.e. image files in subfolders will not be processed).

You can append a suffix to the output file as follows:

```
$ openl3 image /path/to/file.png --suffix somesuffix
```

which will create the output file `/path/to/file_somesuffix.npz`.

Arguments can also be provided to change the model used to extract the embedding including the content type used for training (music or env), input representation (linear, mel128, mel256), and output dimensionality (512 or 8192), for example:

```
$ openl3 image /path/to/file.png --content-type env --input-repr mel128 --embedding-
↪size 512
```

The default value for –content-type is music, for –input-repr is mel128 and for –embedding-size is 8192.

You can change the batch size used as the input to the image embedding model (i.e. the number of video frames the model processes at a time) to one appropriate for your computational resources:

```
$ openl3 image /path/to/file.png --image-batch-size 16
```

Finally, you can suppress non-error printouts by running:

```
$ openl3 image /path/to/file.png --quiet
```

2.3.3 Processing video files

To compute embeddings for a single video file via the command line run:

```
$ openl3 video /path/to/file.mp4
```

This will create output files at `/path/to/file_audio.npz` and `/path/to/file_image.npz` for the audio and image embeddings, respectively. Please note that the audio and image embeddings are not synchronized, so the respective timestamps for each modality will not generally be aligned. Image embeddings are computed for every frame of the video, while the specified audio hop size is used for chunking the audio signal in the video. Additionally, please note that available embedding sizes differ for audio and image embeddings.

Functionality regarding specifying models, multiple input files, verbosity, output directories, and suffixes behave the same as with extracting audio and image embeddings. Functionality specific to audio or image embeddings can also be specified as previously specified.

You can change the batch size used as the input to the audio embedding and image embedding models (i.e. the number of audio windows or video frames, respectively, that the model processes at a time) to one appropriate for your computational resources:

```
$ openl3 video /path/to/file.mp4 --audio-batch-size 16 --image-batch-size 16
```

OpenL3 is an open-source Python library for computing deep audio and image embeddings.

The audio and image embedding models provided here are published as part of [1], and are based on the Look, Listen and Learn approach [2]. For details about the embedding models and how they were trained, please see:

Look, Listen and Learn More: Design Choices for Deep Audio Embeddings Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pages 3852-3856, Brighton, UK, May 2019.

CHAPTER 3

API Reference

3.1 API Reference

3.1.1 Core functionality

openl3.core.**get_audio_embedding**(*audio*, *sr*, *model=None*, *input_repr=None*, *content_type='music'*, *embedding_size=6144*, *center=True*, *hop_size=0.1*, *batch_size=32*, *frontend='kapre'*, *verbose=True*)

Computes and returns L3 embedding for given audio data.

Embeddings are computed for 1-second windows of audio.

Parameters

audio [np.ndarray [shape=(N,) or (N,C)] or list[np.ndarray]] 1D numpy array of audio data or list of audio arrays for multiple inputs.

sr [int or list[int]] Sampling rate, or list of sampling rates. If not 48kHz audio will be resampled.

model [tf.keras.Model or None] Loaded model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for model. Ignored if *model* is a valid Keras model.

content_type ["music" or "env"] Type of content used to train the embedding model. Ignored if *model* is a valid Keras model.

embedding_size [6144 or 512] Embedding dimensionality. Ignored if *model* is a valid Keras model.

center [boolean] If True, pads beginning of signal so timestamps correspond to center of window.

hop_size [float] Hop size in seconds.

batch_size [int] Batch size used for input to embedding model

frontend ["kapre" or "librosa"] The audio frontend to use. By default, it will use "kapre".

verbose [bool] If True, prints verbose messages.

Returns

embedding [np.ndarray [shape=(T, D)] or list[np.ndarray]] Array of embeddings for each window or list of such arrays for multiple audio clips.

timestamps [np.ndarray [shape=(T,)] or list[np.ndarray]] Array of timestamps corresponding to each embedding in the output or list of such arrays for multiple audio cplips.

openl3.core.**get_image_embedding**(*image*, *frame_rate=None*, *model=None*, *input_repr='mel256'*, *content_type='music'*, *embedding_size=8192*, *batch_size=32*, *verbose=True*)

Computes and returns L3 embedding for given video frame (image) data.

Embeddings are computed for every image in the input.

Parameters

image [np.ndarray [shape=(H, W, C) or (N, H, W, C)] or list[np.ndarray]] 3D or 4D numpy array of image data. If the images are not 224x224, the images are resized so that the smallest size is 256 and then the center 224x224 patch is extracted from the images. Any type is accepted, and will be converted to np.float32 in the range [-1,1]. Signed data-types are assumed to take on negative values. A list of image arrays can also be provided.

frame_rate [int or list[int] or None] Video frame rate (if applicable), which if provided results in a timestamp array being returned. A list of frame rates can also be provided. If None, no timestamp array is returned.

model [tf.keras.Model or None] Loaded model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for to train audio part of embedding model. Ignored if *model* is a valid Keras model.

content_type ["music" or "env"] Type of content used to train the embedding model. Ignored if *model* is a valid Keras model.

embedding_size [8192 or 512] Embedding dimensionality. Ignored if *model* is a valid Keras model.

batch_size [int] Batch size used for input to embedding model

verbose [bool] If True, prints verbose messages.

Returns

embedding [np.ndarray [shape=(N, D)]]

Array of embeddings for each frame.

timestamps [np.ndarray [shape=(N,)]] Array of timestamps for each frame. If *frame_rate* is None, this is not returned.

openl3.core.**get_output_path**(*filepath*, *suffix*, *output_dir=None*)

Returns path to output file corresponding to the given input file.

Parameters

filepath [str] Path to audio file to be processed

suffix [str] String to append to filename (including extension)

output_dir [str or None] Path to directory where file will be saved. If None, will use directory of given filepath.

Returns

output_path [str] Path to output file

openl3.core.**preprocess_audio**(*audio*, *sr*, *hop_size=0.1*, *input_repr=None*, *center=True*, ***kw*)

Preprocess the audio into a format compatible with the model.

Parameters

audio [np.ndarray [shape=(N,) or (N,C)] or list[np.ndarray]] 1D numpy array of audio data or list of audio arrays for multiple inputs.

sr [int or list[int]] Sampling rate, or list of sampling rates. If not 48kHz audio will be resampled.

hop_size [float] Hop size in seconds.

input_repr [str or None] Spectrogram representation used for model. If input_repr, is None, then no spectrogram is computed and it is assumed that the model contains the details about the input representation.

center [boolean] If True, pads beginning of signal so timestamps correspond to center of window.

Returns

input_data (np.ndarray): The preprocessed audio. Depending on the value of input_repr, it will be np.ndarray[batch, time, frequency, 1] if a valid input representation is provided, or np.ndarray[batch, time, 1] if no input_repr is provided.

openl3.core.**process_audio_file**(*filepath*, *output_dir=None*, *suffix=None*, *model=None*, *input_repr=None*, *content_type='music'*, *embedding_size=6144*, *center=True*, *hop_size=0.1*, *batch_size=32*, *overwrite=False*, *frontend='kapre'*, *verbose=True*)

Computes and saves L3 embedding for a given audio file

Parameters

filepath [str or list[str]] Path or list of paths to WAV file(s) to be processed.

output_dir [str or None] Path to directory for saving output files. If None, output files will be saved to the directory containing the input file.

suffix [str or None] String to be appended to the output filename, i.e. <base filename>_<suffix>.npz. If None, then no suffix will be added, i.e. <base filename>.npz.

model [tf.keras.Model or None] Loaded model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used as model input. Ignored if *model* is a valid Keras model with a Kapre frontend. This is required with a Librosa frontend.

content_type ["music" or "env"] Type of content used to train the embedding model. Ignored if *model* is a valid Keras model.

embedding_size [6144 or 512] Embedding dimensionality. Ignored if *model* is a valid Keras model.

center [boolean] If True, pads beginning of signal so timestamps correspond to center of window.

hop_size [float] Hop size in seconds.

batch_size [int] Batch size used for input to embedding model

overwrite [bool] If True, overwrites existing output files

frontend ["kapre" or "librosa"] The audio frontend to use. By default, it will use "kapre".

verbose [bool] If True, prints verbose messages.

openl3.core.**process_image_file**(*filepath*, *output_dir=None*, *suffix=None*, *model=None*, *input_repr='mel256'*, *content_type='music'*, *embedding_size=8192*, *batch_size=32*, *overwrite=False*, *verbose=True*)

Computes and saves L3 embedding for a given image file

Parameters

filepath [str or list[str]] Path or list of paths to image file(s) to be processed.

output_dir [str or None] Path to directory for saving output files. If None, output files will be saved to the directory containing the input file.

suffix [str or None] String to be appended to the output filename, i.e. <base filename>_<suffix>.npz. If None, then no suffix will be added, i.e. <base filename>.npz.

model [tf.keras.Model or None] Loaded model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for model. Ignored if *model* is a valid Keras model.

content_type ["music" or "env"] Type of content used to train the embedding model. Ignored if *model* is a valid Keras model.

embedding_size [8192 or 512] Embedding dimensionality. Ignored if *model* is a valid Keras model.

batch_size [int] Batch size used for input to embedding model

overwrite [bool] If True, overwrites existing output files

verbose [bool] If True, prints verbose messages.

openl3.core.**process_video_file**(*filepath*, *output_dir=None*, *suffix=None*, *audio_model=None*, *image_model=None*, *input_repr=None*, *content_type='music'*, *audio_embedding_size=6144*, *audio_center=True*, *audio_hop_size=0.1*, *image_embedding_size=8192*, *audio_batch_size=32*, *image_batch_size=32*, *audio_frontend='kapre'*, *overwrite=False*, *verbose=True*)

Computes and saves L3 audio and video frame embeddings for a given video file

Note that image embeddings are computed for every frame of the video. Also note that embeddings for the audio and images are not temporally aligned. Please refer to the timestamps in the output files for the corresponding timestamps for each set of embeddings.

Parameters

filepath [str or list[str]] Path or list of paths to video file(s) to be processed.

output_dir [str or None] Path to directory for saving output files. If None, output files will be saved to the directory containing the input file.

suffix [str or None] String to be appended to the output filename, i.e. <base filename>_<modality>_<suffix>.npz. If None, then no suffix will be added, i.e. <base filename>_<modality>.npz.

audio_model [tf.keras.Model or None] Loaded audio model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

image_model [tf.keras.Model or None] Loaded audio model object. If a model is provided, then *input_repr*, *content_type*, and *embedding_size* will be ignored. If None is provided, the model will be loaded using the provided values of *input_repr*, *content_type* and *embedding_size*.

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for audio model. Ignored if *model* is a valid Keras model with a Kapre frontend. This is required with a Librosa frontend.

content_type ["music" or "env"] Type of content used to train the embedding model. Ignored if *model* is a valid Keras model.

audio_embedding_size [6144 or 512] Audio embedding dimensionality. Ignored if *model* is a valid Keras model.

audio_center [boolean] If True, pads beginning of audio signal so timestamps correspond to center of window.

audio_hop_size [float] Hop size in seconds.

image_embedding_size [8192 or 512] Video frame embedding dimensionality. Ignored if *model* is a valid Keras model.

audio_batch_size [int] Batch size used for input to audio embedding model

image_batch_size [int] Batch size used for input to image embedding model

audio_frontend ["kapre" or "librosa"] The audio frontend to use. By default, it will use "kapre".

overwrite [bool] If True, overwrites existing output files

verbose [bool] If True, prints verbose messages.

3.1.2 Models functionality

`openl3.models.`**`get_audio_embedding_model_path`**(*input_repr*, *content_type*)

Returns the local path to the model weights file for the model with the given characteristics

Parameters

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for model.

content_type ["music" or "env"] Type of content used to train embedding.

Returns

output_path [str] Path to given model object

`openl3.models.`**`get_image_embedding_model_path`**(*input_repr*, *content_type*)

Returns the local path to the model weights file for the model with the given characteristics

Parameters

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for model.

content_type ["music" or "env"] Type of content used to train embedding.

Returns

output_path [str] Path to given model object

openl3.models.**kapre_v0_1_4_magnitude_to_decibel**(*x*, *ref_value=1.0*, *amin=1e-10*, *dynamic_range=80.0*)

log10 tensorflow function.

openl3.models.**load_audio_embedding_model**(*input_repr*, *content_type*, *embedding_size*, *frontend='kapre'*)

Returns a model with the given characteristics. Loads the model if the model has not been loaded yet.

Parameters

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for audio model.

content_type ["music" or "env"] Type of content used to train embedding.

embedding_size [6144 or 512] Embedding dimensionality.

frontend ["kapre" or "librosa"] The audio frontend to use. If frontend == 'kapre', then the kapre frontend will be included. Otherwise no frontend will be added inside the keras model.

Returns

model [tf.keras.Model] Model object.

openl3.models.**load_image_embedding_model**(*input_repr*, *content_type*, *embedding_size*)

Returns a model with the given characteristics. Loads the model if the model has not been loaded yet.

Parameters

input_repr ["linear", "mel128", or "mel256"] Spectrogram representation used for audio model.

content_type ["music" or "env"] Type of content used to train embedding.

embedding_size [8192 or 512] Embedding dimensionality.

Returns

model [tf.keras.Model] Model object.

CHAPTER 4

Contribute

- Issue tracker
- Source code

CHAPTER 5

Acknowledging OpenL3

Please cite the following papers when using OpenL3 in your work:

[1] Look, Listen and Learn More: Design Choices for Deep Audio Embeddings Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), pages 3852-3856, Brighton, UK, May 2019.

[2] Look, Listen and Learn Relja Arandjelović and Andrew Zisserman IEEE International Conference on Computer Vision (ICCV), Venice, Italy, Oct. 2017.

CHAPTER 6

Changes

6.1 Changelog

6.1.1 v0.4.2

- Fix incorrect embedding_size in `load_image_embedding_model` docstring
- Add `tensorflow.keras` mock modules to `docs/conf.py` to fix docs build
- Remove pin on `sphinx` version

6.1.2 v0.4.1

- Add librosa as an explicit dependency
- Remove upper limit pinning for scikit-image dependency
- Fix version number typo in README
- Update TensorFlow information in README

6.1.3 v0.4.0

- Upgraded to *tensorflow>=2.0.0*. Tensorflow is now included as a dependency because of dual CPU-GPU support.
- Upgraded to *kapre>=0.3.5*. Reverted magnitude scaling method to match *kapre<=0.1.4* as that's what the model was trained on.
- Removed Python 2/3.5 support as they are not supported by Tensorflow 2 (and added 3.7 & 3.8)
- **Add librosa frontend, and allow frontend to be configurable between *kapre* and *librosa***
  - Added `frontend='kapre'` parameter to `get_audio_embedding`, `process_audio_file`, and `load_audio_embedding_model`

  - Added `audio_frontend='kapre'` parameter to `process_video_file` and the CLI
  - Added *frontend='librosa'* flag to *load_audio_embedding_model* for use with a librosa or other external frontend
  - Added a *openl3.preprocess_audio* function that computes the input features needed for each frontend
- Model .h5 no longer have Kapre layers in them and are all importable from `tf.keras`
- Made `skimage` and `moviepy.video.io.VideoFileClip import VideoFileClip` use lazy imports
- Added new regression data for both Kapre 0.3.5 and Librosa
- Parameterized some of the tests to reduce duplication
- Added developer helpers for regression data, weight packaging, and .h5 file manipulation

6.1.4 v0.3.1

- Require *keras>=2.0.9,<2.3.0* in dependencies to avoid force installation of TF 2.x during pip installation.
- Update README and installation docs to explicitly state that we do not yet support TF 2.x and to offer a working dependency combination.
- Require *kapre==0.1.4* in dependencies to avoid installing *tensorflow>=1.14* which break regression tests.

6.1.5 v0.3.0

- Rename audio related embedding functions to indicate that they are specific to audio.
- Add image embedding functionality to API and CLI.
- Add video processing functionality to API and CLI.
- Add batch processing functionality to API and CLI to more efficiently process multiple inputs.
- Update documentation with new functionality.
- Address build issues with updated dependencies.

6.1.6 v0.2.0

- Update embedding models with ones that have been trained with the kapre bug fixed.
- Allow loaded models to be passed in and used in *process_file* and *get_embedding*.
- Rename *get_embedding_model* to *load_embedding_model*.

6.1.7 v0.1.1

- Update kapre to fix issue with dynamic range normalization for decibel computation when computing spectrograms.

6.1.8 v0.1.0

- First release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index