VGGish

The initial AudioSet release included 128-dimensional embeddings of each AudioSet segment produced from a VGG-like audio classification model that was trained on a large YouTube dataset (a preliminary version of what later became YouTube-8M).

We provide a TensorFlow definition of this model, which we call ***VGGish***, as well as supporting code to extract input features for the model from audio waveforms and to post-process the model embedding output into the same format as the released embedding features.

Installation

VGGish depends on the following Python packages:

- `numpy`
- `resampy`
- `tensorflow`
- `tf_slim`
- `six`
- `soundfile`

These are all easily installable via, e.g., `pip install numpy` (as in the sample installation session below). Any reasonably recent version of these packages shold work.

VGGish also requires downloading two data files:

- VGGish model checkpoint, in TensorFlow checkpoint format.
- Embedding PCA parameters, in NumPy compressed archive format.

After downloading these files into the same directory as this README, the installation can be tested by running `python vggish_smoke_test.py` which runs a known signal through the model and checks the output.

Here's a sample installation and test session:

```
# You can optionally install and test VGGish within a Python virtualenv, which
# is useful for isolating changes from the rest of your system. For example, you
# may have an existing version of some packages that you do not want to upgrade,
# or you want to try Python 3 instead of Python 2. If you decide to use a
# virtualenv, you can create one by running
#   $ virtualenv vggish   # For Python 2
# or
#   $ python3 -m venv vggish # For Python 3
# and then enter the virtual environment by running
#   $ source vggish/bin/activate  # Assuming you use bash
# Leave the virtual environment at the end of the session by running
#   $ deactivate
# Within the virtual environment, do not use 'sudo'.

# Upgrade pip first. Also make sure wheel is installed.
$ sudo python -m pip install --upgrade pip wheel

# Install all dependences.
$ sudo pip install numpy resampy tensorflow tf_slim six soundfile

# Clone TensorFlow models repo into a 'models' directory.
$ git clone https://github.com/tensorflow/models.git
$ cd models/research/audioset/vggish
# Download data files into same directory as code.
$ curl -O https://storage.googleapis.com/audioset/vggish_model.ckpt
$ curl -O https://storage.googleapis.com/audioset/vggish_pca_params.npz

# Installation ready, let's test it.
$ python vggish_smoke_test.py
# If we see "Looks Good To Me", then we're all set.
```

Usage

VGGish can be used in two ways:

- *As a feature extractor*: VGGish converts audio input features into a semantically meaningful, high-level 128-D embedding which can be fed as input to a downstream classification model. The downstream model can be shallower than usual because the VGGish embedding is more semantically compact than raw audio features.

  So, for example, you could train a classifier for 10 of the AudioSet classes by using the released embeddings as features. Then, you could use that trained classifier with any arbitrary audio input by running the audio through the audio feature extractor and VGGish model provided here, passing the resulting embedding features as input to your trained model. `vggish_inference_demo.py` shows how to produce VGGish embeddings from arbitrary audio.

- *As part of a larger model*: Here, we treat VGGish as a "warm start" for the lower layers of a model that takes audio features as input and adds more layers on top of the VGGish embedding. This can be used to fine-tune VGGish (or parts thereof) if you have large datasets that might be very different from the typical YouTube video clip. `vggish_train_demo.py` shows how to add layers on top of VGGish and train the whole model.

About the Model

The VGGish code layout is as follows:

- `vggish_slim.py`: Model definition in TensorFlow Slim notation.
- `vggish_params.py`: Hyperparameters.
- `vggish_input.py`: Converter from audio waveform into input examples.
- `mel_features.py`: Audio feature extraction helpers.
- `vggish_postprocess.py`: Embedding postprocessing.
- `vggish_inference_demo.py`: Demo of VGGish in inference mode.
- `vggish_train_demo.py`: Demo of VGGish in training mode.
- `vggish_smoke_test.py`: Simple test of a VGGish installation

Architecture

See `vggish_slim.py` and `vggish_params.py`.

VGGish is a variant of the VGG model, in particular Configuration A with 11 weight layers. Specifically, here are the changes we made:

- The input size was changed to 96x64 for log mel spectrogram audio inputs.
- We drop the last group of convolutional and maxpool layers, so we now have only four groups of convolution/maxpool layers instead of five.
- Instead of a 1000-wide fully connected layer at the end, we use a 128-wide fully connected layer. This acts as a compact embedding layer.

The model definition provided here defines layers up to and including the 128-wide embedding layer. Note that the embedding layer does not include a final non-linear activation, so the embedding value is pre-activation. When training a model stacked on top of VGGish, you should send the embedding through a non-linearity of your choice before adding more layers.

Input: Audio Features

See `vggish_input.py` and `mel_features.py`.

VGGish was trained with audio features computed as follows:

- All audio is resampled to 16 kHz mono.
- A spectrogram is computed using magnitudes of the Short-Time Fourier Transform with a window size of 25 ms, a window hop of 10 ms, and a periodic Hann window.
- A mel spectrogram is computed by mapping the spectrogram to 64 mel bins covering the range 125-7500 Hz.
- A stabilized log mel spectrogram is computed by applying log(mel-spectrum + 0.01) where the offset is used to avoid taking a logarithm of zero.
- These features are then framed into non-overlapping examples of 0.96 seconds, where each example covers 64 mel bands and 96 frames of 10 ms each.

We provide our own NumPy implementation that produces features that are very similar to those produced by our internal production code.

Output: Embeddings

See `vggish_postprocess.py`.

The released AudioSet embeddings were postprocessed before release by applying a PCA transformation (which performs both PCA and whitening) as well as quantization to 8 bits per embedding element. This was done to be compatible with the YouTube-8M project which has released visual and audio embeddings for millions of YouTube videos in the same PCA/whitened/quantized format.

We provide a Python implementation of the postprocessing which can be applied to batches of embeddings produced by VGGish. `vggish_inference_demo.py` shows how the postprocessor can be run after inference.

If you don't need to use the released embeddings or YouTube-8M, then you could skip postprocessing and use raw embeddings.

A Colab showing how to download the model and calculate the embeddings on your own sound data is available here: VGGish Embedding Colab.