Continuous Robust Sound Event Classification Task

Ian McLoughlin

Abstract

The task described here is that used in the PLoS ONE paper entitled "Continuous Robust Sound Event Classification Using Time-Frequency Features and Deep Learning". It builds upon the long established standard isolated sound evaluation task first described by Jonathan Dennis, and widely used by a number of other authors - for evaluating classifiers of isolated sounds.

By contrast the current task extends this to potentially overlapping sounds with no a priori knowledge of start and end points.

The advantage of having a standard task is obvious: it makes experiments easily repeatable by others, and eases the comparison of results when other authors make use of the same method to evaluate their own research. With at least 15 state-of-the-art sound classification papers published with Dennis' isolated sound event detection method, it is easily the most popular task defined to date.

In this current task, exactly the same raw material is used, but is extended through protocol and setup into a continuous, overlapping and robust classification task.

The task uses freely available sound recordings from the Real World Computing Partnership (RWCP) Sound Scene Database in Real Acoustic Environments. These must be obtained directly by the RWCP, and are free for non-commercial or academic users, while commercial users are charged a small free.

Robustness evaluation is performed by mixing raw sounds with background noises from the NOISEX-92 database at several signal-to-noise (SNR) levels. The NOISEX-92 data is widely available online for download.

Citation: Ian McLoughlin Continuous Robust Sound Event Classification Task. **protocols.io**
dx.doi.org/10.17504/protocols.io.iw5cfg6
Published: 13 Jul 2017

Protocol

Obtain the required data and tools

Step 1.

1. The **RWCP Sound Scene Database**

2. **NOISEX-92** recordings of *Destroyer Control Room*, *Speech Babble*, *Factory Floor 1* and *Jet Cockpit*

1.

3. All processing will be done within **MATLAB**. **Octave** is a free alternative which works well, but will require several minor changes in procedure and operating files.

4. For classification purposes, a **machine learning toolbox** is recommended (e.g. Deep Learning Toolbox which is simple for beginners to start with, although has been superceded by tools such as MatConvNet or TensorFlow).

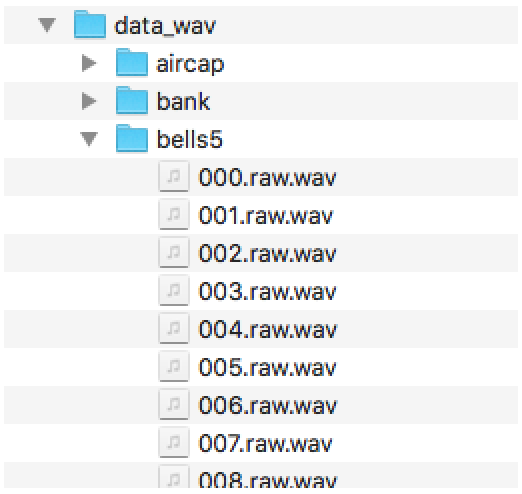
Prepare the data

Step 2.

1. Put the RWCP database into a directory called 'data_wav'. We use the following 50 classes, based on Dennis's selection criteria:

aircap , bank , bells5 , book1 , bottle1 , bowl , buzzer , candybwl , cap1 , case1 , cherry1 , clap1 , clock1 , coffcan , coffmill , coin1 , crumple , cup1 , cymbals , dice1 , doorlock , drum , dryer , file , horn , kara , magno1 , maracas , mechbell , metal05 , pan , particl1 , phone1 , pipong , pump , punch , ring , sandpp1 , saw1 , shaver , snap , spray , stapler , sticks , string , teak1 , tear , trashbox , whistle1 , wood1

So the directory structure looks like this;



```
data_wav
    aircap
    bank
    bells5
        000.raw.wav
        001.raw.wav
        002.raw.wav
        003.raw.wav
        004.raw.wav
        005.raw.wav
        006.raw.wav
        007.raw.wav
        008.raw.wav
```

The .raw files go from 000.raw.wav to 079.raw.wav in each class subdirectory.

2. Put the NOISEX-92 files into another directory called 'noisex-92'. The files we tend to use are;

babble.wav, factory1.wav, f16.wav, destroyerengine.wav

3. Mix the files. The following MATLAB *might* work. It is courtest of H.-M. Zhang;

```
clear all;
SNR=0;
noise_path{1}='noisex-92/babble.wav';
noise_path{2}='noisex-92/factory1.wav';
noise_path{3}='noisex-92/f16.wav';
noise_path{4}='noisex-92/destroyerengine.wav';
wav_dir='data_wav/';
mydir=dir(wav_dir);

fs=16000;
t=60;
n=100;
m=15;
streams=zeros(fs*t,n);
scripts=zeros(m,3,n);

datadir=cell(1,1500);
ntest=0;
for i=3:52
mysubdir=dir([wav_dir,mydir(i).name]);
for j=3:82
if mod(j-3,8)<3
ntest=ntest+1;
a=[wav_dir,mydir(i).name];
a=[a,'/'];
a=[a,mysubdir(j).name];
datadir{ntest}=a;
end;
end;
end;

rand('state',0);
sq=randperm(1500);
for test=1:1500
[data,fs]=audioread(datadir{sq(test)});
len=length(data);
start=ceil(rand()*(fs*t-len+1));

i=ceil(test/m);
streams(start:start+len-1,i)=streams(start:start+len-1,i)+data;
scripts(mod(test-1,m)+1,1,i)=ceil(sq(test)/30);
scripts(mod(test-1,m)+1,2,i)=start/fs;
scripts(mod(test-1,m)+1,3,i)=(start+len-1)/fs;
end;
```

```
for j=1:n
% [streams(:,j),noise]=add_noisem(streams(:,j),noise_path{mod(j,4)+1},SNR,fs);
end;

% sound(streams(:,1),fs);
```

These are the files used for **TESTING**, but for **TRAINING** the system it is slightly different.

4. Training first requires a separate set of directories to be created to hold noise-corrupted sound files at 20dB, 10dB, 0dB and some also use -5dB and -10dB SNR. These directories are usually called;

data_wav_mix0

data_wav_mix10

data_wav_mix-5

...

and so on. Each has the same internal structure of 50 class subdirectories each containing 80 files, and the noise is mixed in the same way as above, using MATLAB.

Training

Step 3.

In the current paper, the classifier is trained on single, isolated sounds, following the procedure outlined at http://www.lintech.org/machine_hearing

Testing

Step 4.

Testing proceeds using the structure outlined in the current paper. It will be made available when the paper is published on PLoS-One.