INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Bhagalpur - 813210

भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर
Indian Institute of Information Technology
Bhagalpur

Summer Internship Project Report

Heartbeat Sound Classification

Submitted by:

MEHUL TYAGI (170101024)

AMAN MISHRA (170101008)

Class: B.Tech. – IV
Semester: VII

Introduction

Stethoscope detection currently provides one of the first warning signs of heart disease. During this procedure, which usually occurs in one's annual physical examination, physicians manually listen for and identify irregularities in heart sounds that indicate a need for further investigation. Early detection and intervention in heart disease greatly improves the lifespan and the effectiveness of treatment options. For this reason, the benefits of placing the first-detection capability in the hands of the individual are immense. This project aims to use machine learning methods to identify and classify heartbeat sounds from audio collected from digital stethoscopes and mobile devices available to the average consumer.

From the audio heart sounds, 18 features are extracted using both the whole signal and specific locations in a heartbeat.

A. Background

Different heart-sounds, resulting from mechanical cardiac events and heard from stethoscope examinations, can be indicators for various cardiac health concerns. Typically, a heartbeat creates two sounds, lub-dub, that can be heard through a stethoscope. These sounds are termed, respectively, S1 and S2 for the first and second heart sounds within one beat. S1 occurs when the mitral and tricuspid valves close at the end of systole, when blood from the atriums fill into the ventricles. S2 is heard when aortic and pulmonic valve leaflets close after diastole, after blood is pushed out from the ventricles. These two sounds make up what a normal heartbeat should sound like.

When abnormalities occur in cardiac physiology, they are oftentimes reflected in heartbeat sounds. A few of the most commonly occurring abnormal sounds, the ones that this project will be identifying and classifying, are heart murmurs, extrasystole, extra heart sounds, and artifact sounds. Murmurs are extra sounds that occur when there is turbulence in blood flow that causes the extra vibrations that can be heard. Physiologically, this turbulence can be caused by abnormal valve events that cause irregularities to the flow of blood, such as regurgitation through the valves. Extrasystole is a premature contraction of the heart that causes an extra sound before S1 and palpitations due to abnormal electrical circuitry within the heart. Extra heart sounds occur after diastole, S2, due to sudden deceleration of blood, caused by abnormalities in cardiac muscle physiology that affects contraction characteristics

B. Related Works

This study is based on Peter J Bentleys dataset, containing audio data, for his Heart Sound Challenge, where he has done preliminary analysis that resulted in 77% precision in identifying normal heart sounds

Setup

A. Datasets

Two different datasets are used to verify the performance of the machine learning models. Dataset A has four classes with 120 total samples. The four classes are artifact, extra heart sound, murmur and normal heartbeat. Dataset B has fewer classes and more samples, sampling heart sounds at 4000 samples per second. This dataset consists of three classes and a total of 461 samples, 149 of which are noisy. The three classes are extrasystole, murmur and normal heartbeat. These two different datasets are used to verify the generalizability of the machine learning models, as well as to compare both dataset results.

B. Preprocessing

Prior to feature extraction, the audio signals were pre-processed. The preprocessing includes removing audio files that are less than 2 seconds, down sampling, removing high frequency noise and normalizing the data. Files less than 2 seconds are removed because those files do not capture a full heartbeat cycle, making it impossible to extrapolate whether or not those samples contain heart sound irregularities. For dataset A, a total of 120 samples are used and for dataset B, a total of 407 samples are used.

Feature Extraction

For feature extraction, the signals were analyzed in two groups. For the first group of extractions, the entire signal was analyzed in both the time and frequency domain. The second group uses significant parts of the signal as a whole to extract features. The significant parts of a signal are S1 and S2. A total of 18 different features were extracted, seven of which are based on the entire signal. The features extracted for the first set of features in the time domain are zero crossings, energy and entropy of energy. In the frequency domain, spectral spread, spectral entropy, spectral flux and Mel Frequency Cepstral Coefficients (MFCCs) were used. All features except MFCCs, were extracted using *librosa*, a python library for audio signal analysis.

Methods and Features Used

- ## Sound Feature: MFCC

Mel Frequency Cepstral Coefficient (MFCC) is by far the most successful feature used in the field of Speech Processing. Speech is a non-stationary signal. As such, normal signal processing techniques cannot be directly applied to it.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

- ## Sound Feature: Onset

  - ### onset detector

    Basic onset detector. Locate note onset events by picking peaks in an onset strength envelope. The *peak_pick* parameters were chosen by large-scale hyper-parameter optimization over the dataset provided

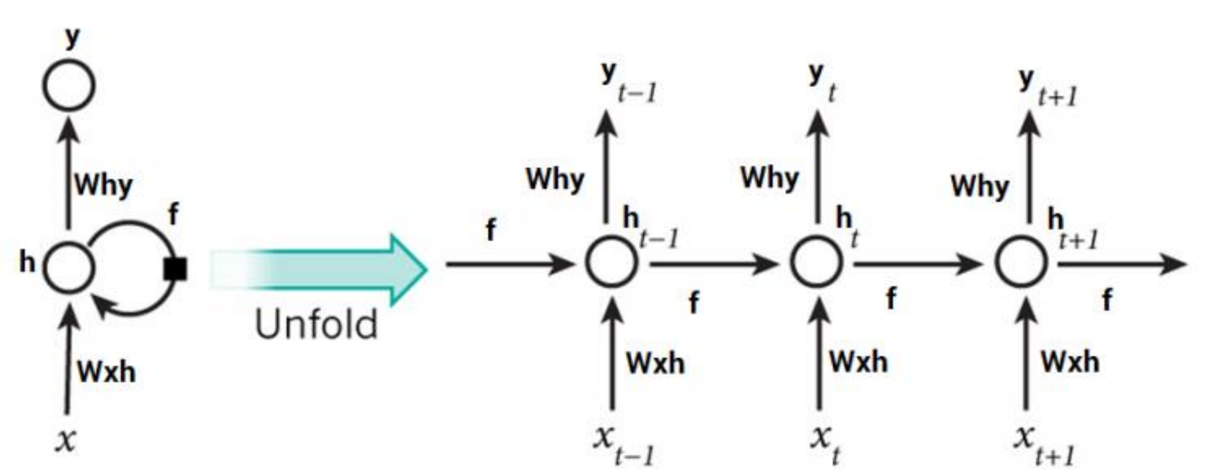
  - ### onset backtrack

    Backtrack detected onset events to the nearest preceding local minimum of an energy function. This function can be used to roll back the timing of detected onsets from a detected peak amplitude to the preceding minimum. This is most useful when using onsets to determine slice points for segmentation.

  - ### onset strength

    Compute a spectral flux onset strength envelope. Onset strength at time t is determined by: mean max(0, S[f, t] - ref_S[f, t - lag]) where ref_S is S after local max filtering along the frequency axis [1]. By default, if a time series y is provided, S will be the log-power Mel spectrogram.

- **LSTM**



LSTM network is comprised of different memory blocks called cells (the rectangles that we see in the image). There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates.

RNN and LSTM are memory-bandwidth limited problems -Temporal convolutional network (TCN) “outperform canonical recurrent networks such as LSTMs across a diverse range of tasks and datasets, while demonstrating longer effective memory”.

The advantage of an LSTM cell compared to a common recurrent unit is its cell memory unit. The cell vector has the ability to encapsulate the notion of forgetting part of its previously stored memory, as well as to add part of the new information.

In [2]:
```
import warnings                        # To ignore any warnings
warnings.filterwarnings("ignore")
%matplotlib inline
%pylab inline
import os
import pandas as pd
import librosa
import librosa.display
import glob
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [3]:
```
# gather software versions
import tensorflow as tf; print('tensorflow version: ', tf.__version__)
import keras; print('keras version: ',keras.__version__)
```

```
tensorflow version:  1.12.0
keras version:  2.2.4

Using TensorFlow backend.
```

In [4]:
```
# parent folder of sound files
INPUT_DIR="../input"
# 16 KHz
SAMPLE_RATE = 16000
# seconds
MAX_SOUND_CLIP_DURATION=12
```

Explorer data

The audio files are of varying lengths, between 1 second and 30 seconds (some have been clipped to reduce excessive noise and provide the salient fragment of the sound).

Most information in heart sounds is contained in the low frequency components, with noise in the higher frequencies. It is common to apply a low-pass filter at 195 Hz. Fast Fourier transforms are also likely to provide useful information about volume and frequency over time. More domain-specific knowledge about the difference between the categories of sounds is provided below.

let's check what is inside each directory and content and input data organization

In [5]:
```
# check what is inside each directory and content
!pwd
!ls -all ../input
```

```
/kaggle/working
total 136
drwxr-xr-x 4 root root  4096 May  9  2018 .
drwxr-xr-x 6 root root  4096 Feb 27 05:17 ..
drwxr-xr-x 2 root root 12288 May  9  2018 set_a
-rw-r--r-- 1 root root  7031 May  9  2018 set_a.csv
-rw-r--r-- 1 root root 17115 May  9  2018 set_a_timing.csv
drwxr-xr-x 2 root root 45056 May  9  2018 set_b
-rw-r--r-- 1 root root 42145 May  9  2018 set_b.csv
```

Check input data in csv files

In [6]:
```
set_a=pd.read_csv(INPUT_DIR+"/set_a.csv")
set_a.head()
```

Out[6]:

	dataset	fname	label	sublabel
0	a	set_a/artifact__201012172012.wav	artifact	NaN
1	a	set_a/artifact__201105040918.wav	artifact	NaN
2	a	set_a/artifact__201105041959.wav	artifact	NaN
3	a	set_a/artifact__201105051017.wav	artifact	NaN
4	a	set_a/artifact__201105060108.wav	artifact	NaN

In [7]:
```
set_a_timing=pd.read_csv(INPUT_DIR+"/set_a_timing.csv")
set_a_timing.head()
```

Out[7]:

	fname	cycle	sound	location
0	set_a/normal__201102081321.wav	1	S1	10021
1	set_a/normal__201102081321.wav	1	S2	20759
2	set_a/normal__201102081321.wav	2	S1	35075
3	set_a/normal__201102081321.wav	2	S2	47244
4	set_a/normal__201102081321.wav	3	S1	62992

In [8]:
```
set_b=pd.read_csv(INPUT_DIR+"/set_b.csv")
set_b.head()
```

Out[8]:

	dataset	fname	label	sublabel
0	b	set_b/Btraining_extrastole_127_1306764300147_C...	extrastole	NaN
1	b	set_b/Btraining_extrastole_128_1306344005749_A...	extrastole	NaN
2	b	set_b/Btraining_extrastole_130_1306347376079_D...	extrastole	NaN
3	b	set_b/Btraining_extrastole_134_1306428161797_C...	extrastole	NaN
4	b	set_b/Btraining_extrastole_138_1306762146980_B...	extrastole	NaN

In [9]:
```
#merge both set-a and set-b
frames = [set_a, set_b]
train_ab=pd.concat(frames)
train_ab.describe()
```

Out[9]:

	dataset	fname	label	sublabel
count	832	832	585	149
unique	2	832	5	2
top	b	set_b/Bunlabelledtest_121_1306263877235_B.wav	normal	noisynormal
freq	656	1	351	120

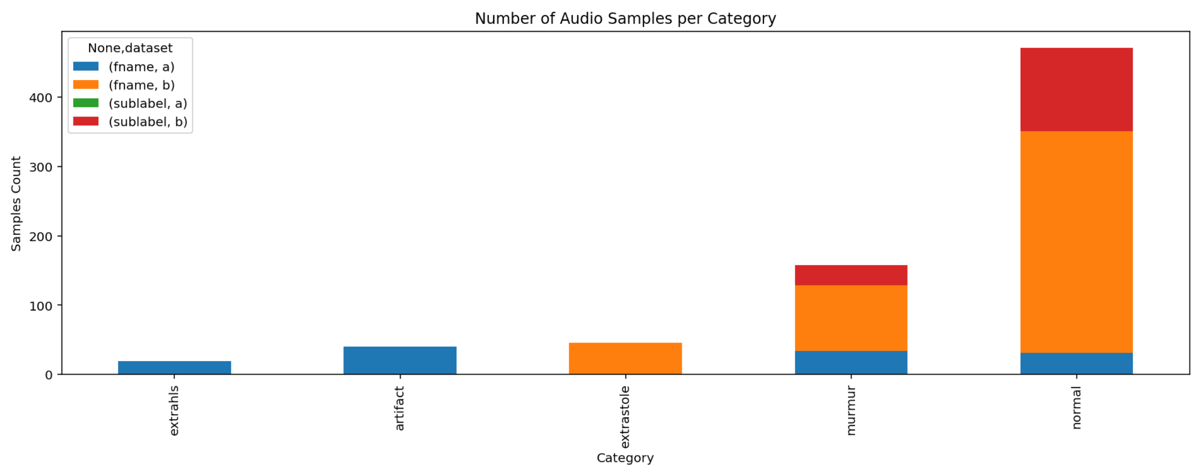
In [10]:
```
#get all unique labels
nb_classes=train_ab.label.unique()

print("Number of training examples=", train_ab.shape[0], "  Number of classes
=", len(train_ab.label.unique()))
print (nb_classes)
```

```
Number of training examples= 832   Number of classes= 6
['artifact' 'extrahls' 'murmur' 'normal' nan 'extrastole']
```

Note: nan label indicate unclassified and unlabel test files

```
In [11]: # visualize data distribution by category
category_group = train_ab.groupby(['label','dataset']).count()
plot = category_group.unstack().reindex(category_group.unstack().sum(axis=1).s
ort_values().index)\
          .plot(kind='bar', stacked=True, title="Number of Audio Samples per C
ategory", figsize=(16,5))
plot.set_xlabel("Category")
plot.set_ylabel("Samples Count");

print('Min samples per category = ', min(train_ab.label.value_counts()))
print('Max samples per category = ', max(train_ab.label.value_counts()))
```

```
Min samples per category =  19
Max samples per category =  351
```




```
In [12]: print('Minimum samples per category = ', min(train_ab.label.value_counts()))
print('Maximum samples per category = ', max(train_ab.label.value_counts()))
```

```
Minimum samples per category =  19
Maximum samples per category =  351
```

let's take a look some sample by category

1. Normal case

In the Normal category there are normal, healthy heart sounds. These may contain noise in the final second of the recording as the device is removed from the body. They may contain a variety of background noises (from traffic to radios). They may also contain occasional random noise corresponding to breathing, or brushing the microphone against clothing or skin. A normal heart sound has a clear “lub dub, lub dub” pattern, with the time from “lub” to “dub” shorter than the time from “dub” to the next “lub” (when the heart rate is less than 140 beats per minute)(source: Rita Getz)

```
In [13]: normal_file=INPUT_DIR+"/set_a/normal__201106111136.wav"
```

In [14]:
```python
# heart it
import IPython.display as ipd
ipd.Audio(normal_file)
```

Out[14]:

0:00 / 0:04

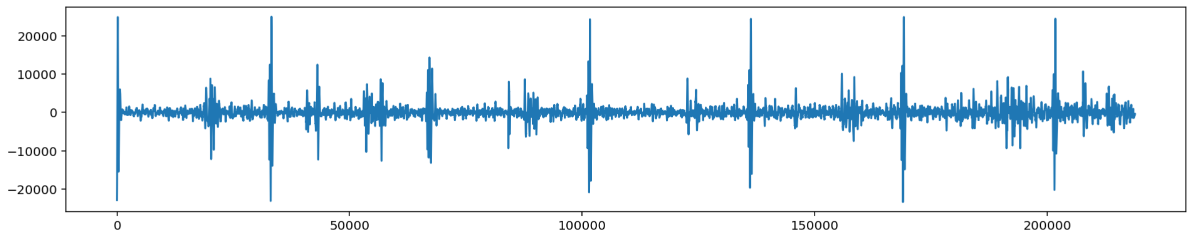
In [15]:
```python
# Load use wave
import wave
wav = wave.open(normal_file)
print("Sampling (frame) rate = ", wav.getframerate())
print("Total samples (frames) = ", wav.getnframes())
print("Duration = ", wav.getnframes()/wav.getframerate())
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  218903
Duration =  4.963786848072562
```

In [16]:
```python
# Load use scipy
from scipy.io import wavfile
rate, data = wavfile.read(normal_file)
print("Sampling (frame) rate = ", rate)
print("Total samples (frames) = ", data.shape)
print(data)
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  (218903,)
[-22835 -22726 -22595 ...   -474   -450   -439]
```

In [17]:
```python
# plot wave by audio frames
plt.figure(figsize=(16, 3))
plt.plot(data, '-', );
```

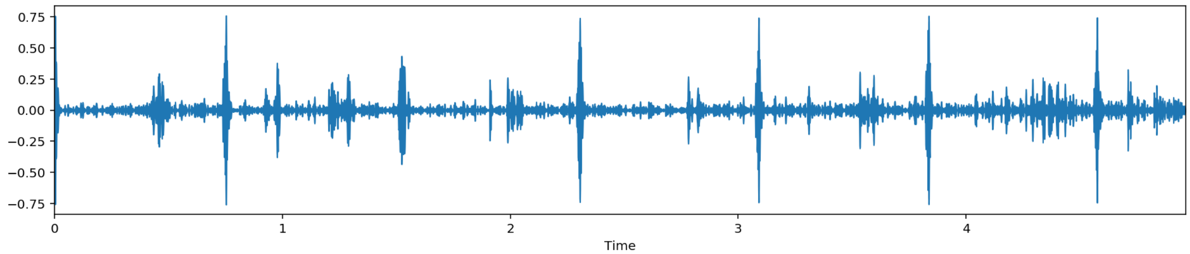


In [18]:
```python
# Load using Librosa
y, sr = librosa.load(normal_file, duration=5)   #default sampling rate is 22 H
Z
dur=librosa.get_duration(y)
print ("duration:", dur)
print(y.shape, sr)
```

```
duration: 4.963809523809524
(109452,) 22050
```

In [19]:
```
# librosa plot
plt.figure(figsize=(16, 3))
librosa.display.waveplot(y, sr=sr)
```

Out[19]: <matplotlib.collections.PolyCollection at 0x7f8fdbc3be80>



2. Murmur

Heart murmurs sound as though there is a “whooshing, roaring, rumbling, or turbulent fluid” noise in one of two temporal locations: (1) between “lub” and “dub”, or (2) between “dub” and “lub”. They can be a symptom of many heart disorders, some serious. There will still be a “lub” and a “dub”. One of the things that confuses non-medically trained people is that murmurs happen between lub and dub or between dub and lub; not on lub and not on dub.(source: Rita Getz)

In [20]:
```
# murmur case
murmur_file=INPUT_DIR+"/set_a/murmur__201108222231.wav"
y2, sr2 = librosa.load(murmur_file,duration=5)
dur=librosa.get_duration(y)
print ("duration:", dur)
print(y2.shape,sr2)
```

```
duration: 4.963809523809524
(110250,) 22050
```

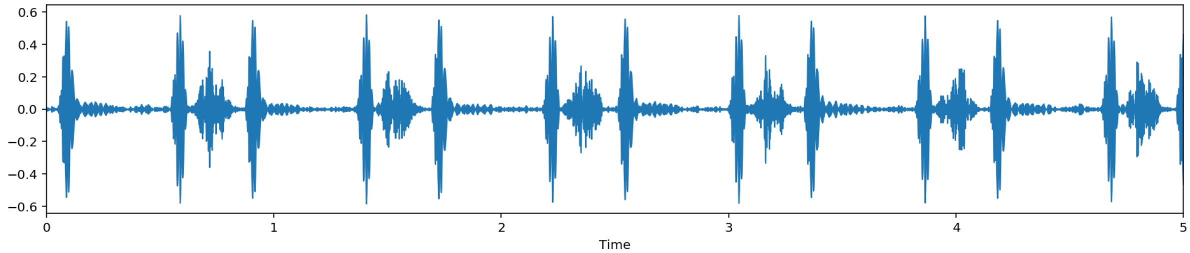
In [21]:
```
# heart it
import IPython.display as ipd
ipd.Audio(murmur_file)
```

Out[21]:

0:00 / 0:07

In [22]:
```
# show it
plt.figure(figsize=(16, 3))
librosa.display.waveplot(y2, sr=sr2)
```

Out[22]: <matplotlib.collections.PolyCollection at 0x7f8fd9adff28>



3. Extrasystole

Extrasystole sounds may appear occasionally and can be identified because there is a heart sound that is out of rhythm involving extra or skipped heartbeats, e.g. a “lub-lub dub” or a “lub dub-dub”. (This is not the same as an extra heart sound as the event is not regularly occuring.) An extrasystole may not be a sign of disease. It can happen normally in an adult and can be very common in children. However, in some situations extrasystoles can be caused by heart diseases. If these diseases are detected earlier, then treatment is likely to be more effective. (source: Rita Getz)

In [23]:
```
# Extrasystole case
extrastole_file=INPUT_DIR+"/set_b/extrastole__127_1306764300147_C2.wav"
y3, sr3 = librosa.load(extrastole_file, duration=5)
dur=librosa.get_duration(y)
print ("duration:", dur)
print(y3.shape,sr3)
```

```
duration: 4.963809523809524
(103106,) 22050
```

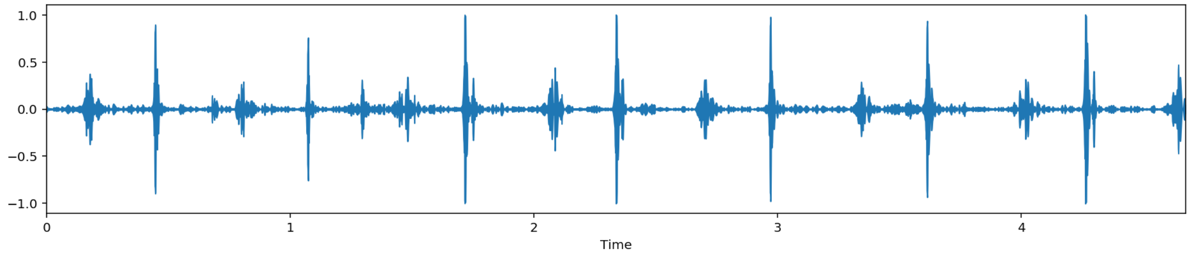
In [24]:
```
# heart it
import IPython.display as ipd
ipd.Audio(extrastole_file)
```

Out[24]:

0:00 / 0:04

In [25]:
```
# show it
plt.figure(figsize=(16, 3))
librosa.display.waveplot(y3, sr=sr3)
```

Out[25]: <matplotlib.collections.PolyCollection at 0x7f8fd9a599b0>



4. Artifact

In the Artifact category there are a wide range of different sounds, including feedback squeals and echoes, speech, music and noise. There are usually no discernable heart sounds, and thus little or no temporal periodicity at frequencies below 195 Hz. This category is the most different from the others. It is important to be able to distinguish this category from the other three categories, so that someone gathering the data can be instructed to try again.(source: Rita Getz)

In [26]:
```
# sample file
artifact_file=INPUT_DIR+"/set_a/artifact__201012172012.wav"
y4, sr4 = librosa.load(artifact_file, duration=5)
dur=librosa.get_duration(y)
print ("duration:", dur)
print(y4.shape,sr4)
```

```
duration: 4.963809523809524
(110250,) 22050
```

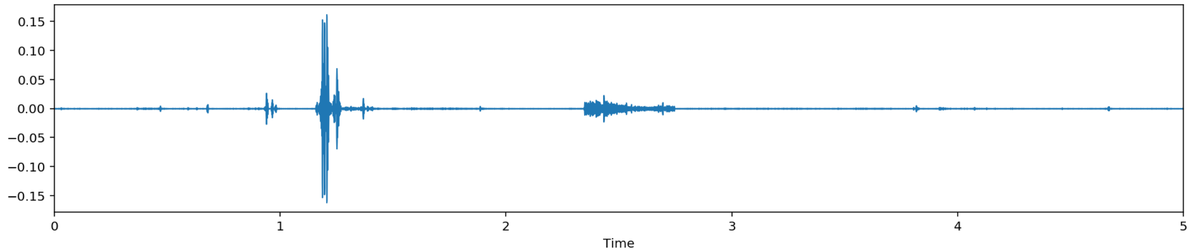
In [27]:
```
# heart it
import IPython.display as ipd
ipd.Audio(artifact_file)
```

Out[27]:

0:00 / 0:09

In [28]:
```python
# show it
plt.figure(figsize=(16, 3))
librosa.display.waveplot(y4, sr=sr4)
```

Out[28]: <matplotlib.collections.PolyCollection at 0x7f8fd9a28ba8>



5. Extra Heart Sound

In the Artifact category there are a wide range of different sounds, including feedback squeals and echoes, speech, music and noise. There are usually no discernable heart sounds, and thus little or no temporal periodicity at frequencies below 195 Hz. This category is the most different from the others. It is important to be able to distinguish this category from the other three categories, so that someone gathering the data can be instructed to try again.(source: Rita Getz)

In [29]:
```python
# sample file
extrahls_file=INPUT_DIR+"/set_a/extrahls__201101070953.wav"
y5, sr5 = librosa.load(extrahls_file, duration=5)
dur=librosa.get_duration(y)
print ("duration:", dur)
print(y5.shape,sr5)
```

```
duration: 4.963809523809524
(110250,) 22050
```

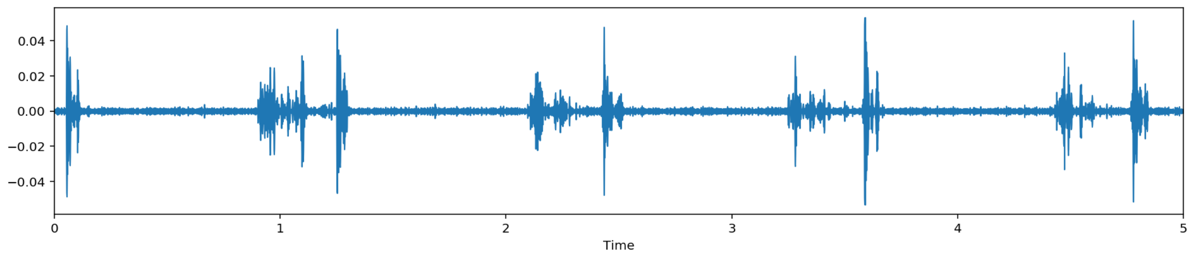
In [30]:
```python
# heart it
import IPython.display as ipd
ipd.Audio(extrahls_file)
```

Out[30]:

0:00 / 0:08

In [31]:
```
# show it
plt.figure(figsize=(16, 3))
librosa.display.waveplot(y5, sr=sr5)
```

Out[31]: <matplotlib.collections.PolyCollection at 0x7f8fd9a00828>



Audio Length

the lengths of the audio files in the dataset varies from 1 to 30 seconds long. for training purpose we use first 5 seconds of the audio. padd missing lenght for file smaller than 5 seconds.

Data Handling in Audio domain

As with all unstructured data formats, audio data has a couple of preprocessing steps which have to be followed before it is presented for analysis. Another way of representing audio data is by converting it into a different domain of data representation, namely the frequency domain.

![frequency domain] https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/08/23212155/time_freq.png (https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/08/23212155/time_freq.png)

There are a few more ways in which audio data can be represented. example. using MFCs (Mel-Frequency cepstrums)

General Audio Features

- Time Domain features (eg. RMSE of waveform)
- Frequency domain features (eg. Amplitude of individual freuencies)
- Perceptual features (eg. MFCC)
- Windowing features (eg. Hamming distances of windows)

After extracting these features, it is then sent to the machine learning model for further analysis.

Sound Feature: MFCC

Mel Frequency Cepstral Coefficient (MFCC) is by far the most successful feature used in the field of Speech Processing. Speech is a non-stationary signal. As such, normal signal processing techniques cannot be directly applied to it.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

MFCCs are commonly derived as follows: -Take the Fourier transform of (a windowed excerpt of) a signal. -Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows. -Take the logs of the powers at each of the mel frequencies. -Take the discrete cosine transform of the list of mel log powers, as if it were a signal. The MFCCs are the amplitudes of the resulting spectrum.

In general, a 39-dimensional feature vector is used which is composed of first 13 MFCCs and their corresponding 13 delta and 13 delta-delta.

In [32]:

```
# Here's a sample generate mfccs from a wave file
normal_file=INPUT_DIR+"/set_a/normal__201106111136.wav"
#y, sr = librosa.load(sample_file, offset=7, duration=7)
y, sr = librosa.load(normal_file)
mfccs = librosa.feature.mfcc(y=y, sr=sr)
print (mfccs)
```

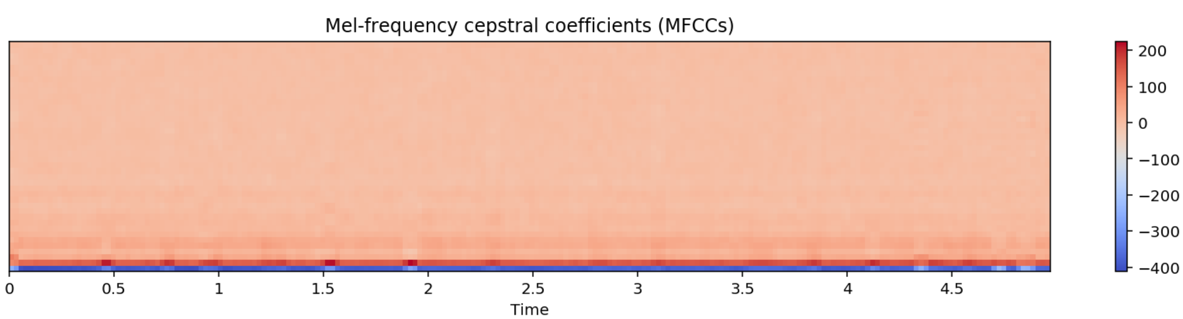
```
[[-2.45461310e+02 -2.91119158e+02 -4.02448048e+02 ... -3.67871637e+02
  -3.70708414e+02 -3.72469945e+02]
 [ 8.86406929e+01  9.88661324e+01  1.33256498e+02 ...  1.51281027e+02
   1.57261842e+02  1.52452273e+02]
 [ 1.03978908e+02  8.56985019e+01  2.37443259e+01 ...  2.91626730e+01
   2.93766991e+01  3.74463005e+01]
 ...
 [-1.33243318e+01 -8.19430184e+00 -1.18989196e+00 ...  9.66035179e-02
   6.24897256e-01  1.19534810e+00]
 [ 3.16322374e-01 -4.99245923e-01 -1.58862224e-01 ...  2.22770953e+00
  -2.15090204e-01  4.84898894e+00]
 [ 3.34313266e+00 -9.89011623e-01 -2.77108967e+00 ...  2.65077442e+00
  -1.38751247e+00  1.85793453e+00]]
```

In [33]:
```
# Use a pre-computed log-power Mel spectrogram
S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128,fmax=8000)
log_S=librosa.feature.mfcc(S=librosa.power_to_db(S))
print (log_S)
```

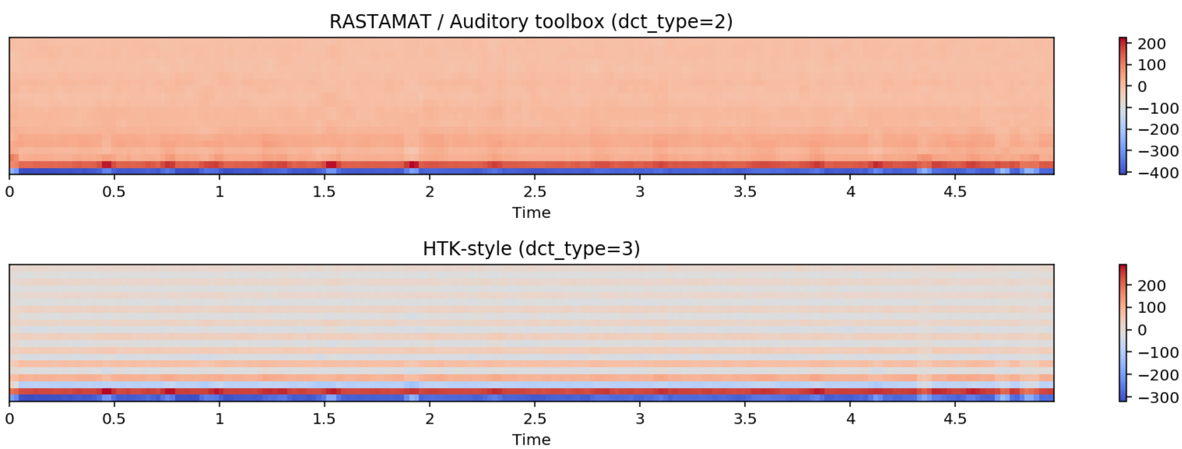
```
[[-2.46578451e+02 -2.89466533e+02 -3.86061424e+02 ... -3.50173598e+02
  -3.51358336e+02 -3.55300579e+02]
 [ 1.09104030e+02  1.16516227e+02  1.30325863e+02 ...  1.49302754e+02
   1.54021441e+02  1.51842599e+02]
 [ 9.83255629e+01  7.55811535e+01  1.59119043e+01 ...  2.06767459e+01
   2.19900360e+01  3.01192216e+01]
 ...
 [ 4.26571311e+00  2.46345083e-01 -2.07913916e+00 ...  2.77262558e+00
  -1.40852842e+00  3.58013971e+00]
 [ 3.38976142e-01 -9.63366773e-01 -3.46149708e+00 ...  1.52363932e+00
   1.05708759e-01 -2.32015820e+00]
 [-5.24993637e+00 -2.43681813e+00 -1.56827403e+00 ...  2.74896890e-01
   5.19361242e-01 -6.77453398e+00]]
```

In [34]:
```
# Get more components
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
#print (mfccs)
```

In [35]:
```
# Visualize the MFCC series
# Mel-frequency cepstral coefficients (MFCCs)
plt.figure(figsize=(12, 3))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('Mel-frequency cepstral coefficients (MFCCs)')
plt.tight_layout()
```



In [36]:
```
# Compare different DCT bases
m_slaney = librosa.feature.mfcc(y=y, sr=sr, dct_type=2)

#m_dct1 = librosa.feature.mfcc(y=y, sr=sr, dct_type=1)
plt.figure(figsize=(12, 6))
#plt.subplot(3, 1, 1)
#librosa.display.specshow(m_dct1, x_axis='time')
#plt.title('Discrete cosine transform (dct_type=1)')
#plt.colorbar()
m_htk = librosa.feature.mfcc(y=y, sr=sr, dct_type=3)
plt.subplot(3, 1, 2)
librosa.display.specshow(m_slaney, x_axis='time')
plt.title('RASTAMAT / Auditory toolbox (dct_type=2)')
plt.colorbar()
plt.subplot(3, 1, 3)
librosa.display.specshow(m_htk, x_axis='time')
plt.title('HTK-style (dct_type=3)')
plt.colorbar()
plt.tight_layout()
```



Sound Feature: Onset

onset detector

Basic onset detector. Locate note onset events by picking peaks in an onset strength envelope. The peak_pick parameters were chosen by large-scale hyper-parameter optimization over the dataset provided

In [37]:
```
# Get onset times from a signal
onset_frames = librosa.onset.onset_detect(y=y, sr=sr)
librosa.frames_to_time(onset_frames, sr=sr)
```

Out[37]:
```
array([0.48761905, 0.7662585 , 0.95201814, 1.25387755, 1.55573696,
       1.92725624, 2.32199546, 2.80961451, 3.11147392, 3.57587302,
       3.80807256, 3.85451247, 4.13315193, 4.36535147, 4.59755102,
       4.73687075, 4.85297052])
```

In [38]:

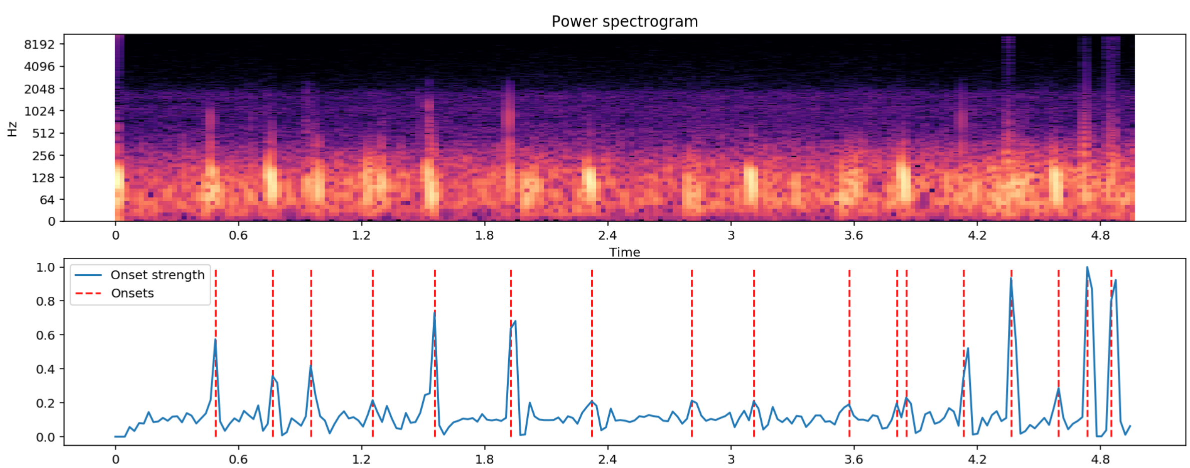
```
# use a pre-computed onset envelope
o_env = librosa.onset.onset_strength(y, sr=sr)
times = librosa.frames_to_time(np.arange(len(o_env)), sr=sr)
onset_frames = librosa.onset.onset_detect(onset_envelope=o_env, sr=sr)
```

In [39]:

```
# visualize it
D = np.abs(librosa.stft(y))
plt.figure(figsize=(16, 6))
ax1 = plt.subplot(2, 1, 1)
librosa.display.specshow(librosa.amplitude_to_db(D, ref=np.max),x_axis='time',
y_axis='log')
plt.title('Power spectrogram')
plt.subplot(2, 1, 2, sharex=ax1)

plt.plot(times, o_env, label='Onset strength')
plt.vlines(times[onset_frames], 0, o_env.max(), color='r', alpha=0.9,linestyle
='--', label='Onsets')
plt.axis('tight')
plt.legend(frameon=True, framealpha=0.75)
```

Out[39]:

```
<matplotlib.legend.Legend at 0x7f8fd98d8f98>
```



onset_backtrack

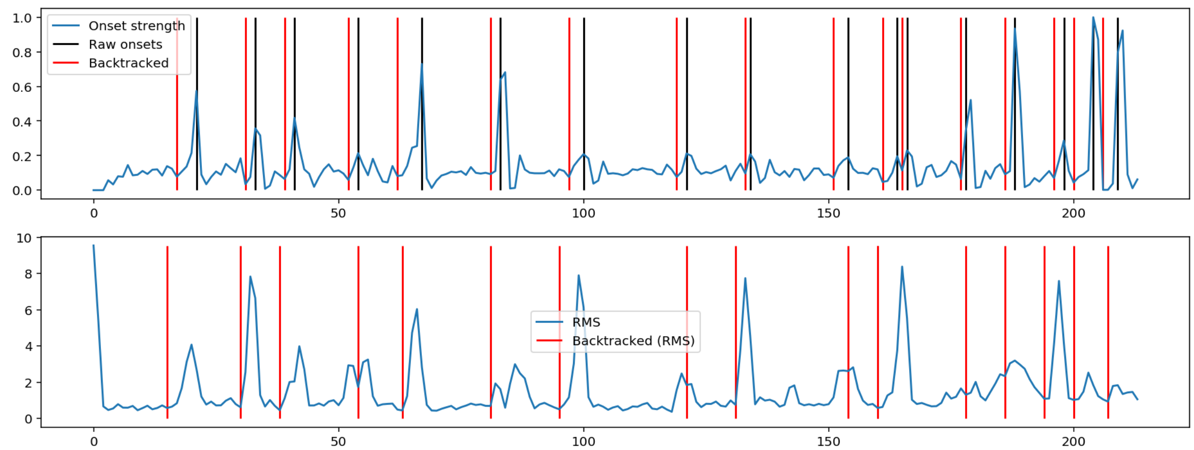
Backtrack detected onset events to the nearest preceding local minimum of an energy function. This function can be used to roll back the timing of detected onsets from a detected peak amplitude to the preceding minimum. This is most useful when using onsets to determine slice points for segmentation

In [40]:

```
oenv = librosa.onset.onset_strength(y=y, sr=sr)
# Detect events without backtracking
onset_raw = librosa.onset.onset_detect(onset_envelope=oenv, backtrack=False)
# Backtrack the events using the onset envelope
onset_bt = librosa.onset.onset_backtrack(onset_raw, oenv)
# Backtrack the events using the RMS values
rms = librosa.feature.rms(S=np.abs(librosa.stft(y=y)))
onset_bt_rms = librosa.onset.onset_backtrack(onset_raw, rms[0])
```

In [41]:
```
# Plot the results
plt.figure(figsize=(16, 6))
plt.subplot(2,1,1)
plt.plot(oenv, label='Onset strength')
plt.vlines(onset_raw, 0, oenv.max(), label='Raw onsets')
plt.vlines(onset_bt, 0, oenv.max(), label='Backtracked', color='r')
plt.legend(frameon=True, framealpha=0.75)
plt.subplot(2,1,2)
plt.plot(rms[0], label='RMS')
plt.vlines(onset_bt_rms, 0, rms.max(), label='Backtracked (RMS)', color='r')
plt.legend(frameon=True, framealpha=0.75)
```

Out[41]:
```
<matplotlib.legend.Legend at 0x7f8fd8be24a8>
```

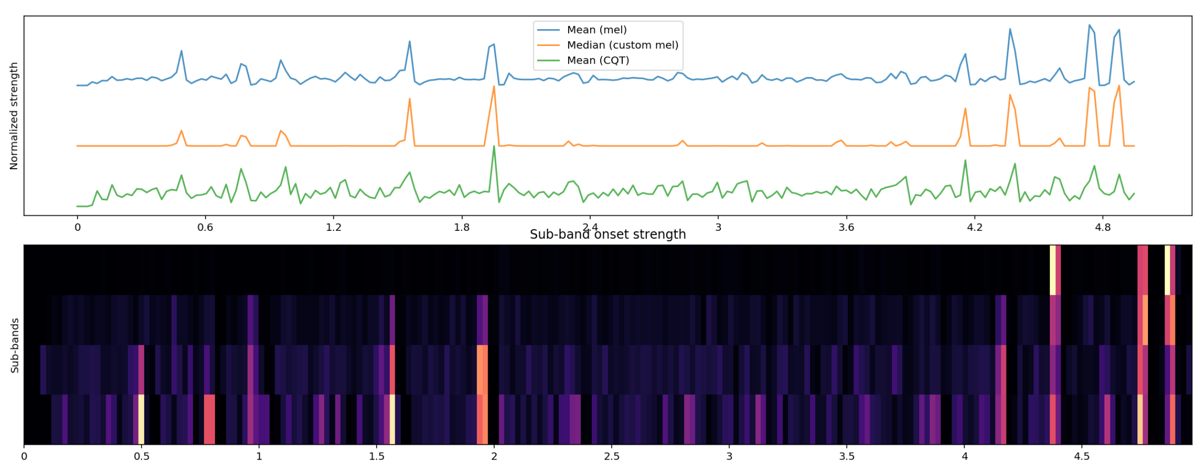


onset strength

Compute a spectral flux onset strength envelope. Onset strength at time t is determined by: mean_f max(0, S[f, t] - ref_S[f, t - lag]) where ref_S is S after local max filtering along the frequency axis [1]. By default, if a time series y is provided, S will be the log-power Mel spectrogram.

In [42]:
```python
D = np.abs(librosa.stft(y))
times = librosa.frames_to_time(np.arange(D.shape[1]))

plt.figure(figsize=(16, 6))
#ax1 = plt.subplot(2, 1, 1)
#librosa.display.specshow(librosa.amplitude_to_db(D, ref=np.max),y_axis='log', x_axis='time')
#plt.title('Power spectrogram')

# Construct a standard onset function
onset_env = librosa.onset.onset_strength(y=y, sr=sr)
plt.subplot(2, 1, 1, sharex=ax1)
plt.plot(times, 2 + onset_env / onset_env.max(), alpha=0.8,label='Mean (mel)')

# median
onset_env = librosa.onset.onset_strength(y=y, sr=sr,aggregate=np.median,fmax=8000, n_mels=256)
plt.plot(times, 1+ (onset_env/onset_env.max()), alpha=0.8,label='Median (custom mel)')

# Constant-Q spectrogram instead of Mel
onset_env = librosa.onset.onset_strength(y=y, sr=sr,feature=librosa.cqt)
plt.plot(times, onset_env / onset_env.max(), alpha=0.8,label='Mean (CQT)')
plt.legend(frameon=True, framealpha=0.75)
plt.ylabel('Normalized strength')
plt.yticks([])
plt.axis('tight')
plt.tight_layout()

onset_subbands = librosa.onset.onset_strength_multi(y=y, sr=sr, channels=[0, 32, 64, 96, 128])
#plt.figure(figsize=(16, 6))
plt.subplot(2, 1, 2)
librosa.display.specshow(onset_subbands, x_axis='time')
plt.ylabel('Sub-bands')
plt.title('Sub-band onset strength')
```

Out[42]:
```
Text(0.5,1,'Sub-band onset strength')
```



Loading Data

In [43]:
```
print("Number of training examples=", train_ab.shape[0], "  Number of classes
=", len(train_ab.label.unique()))
```

```
Number of training examples= 832   Number of classes= 6
```

Loading od the audio data file will be based on content from directory since each filename is associate with the category type. hence, we can use csv file for cross reference check. Based on directory content approach will be more flexible.

In [44]:
```
def audio_norm(data):
    max_data = np.max(data)
    min_data = np.min(data)
    data = (data-min_data)/(max_data-min_data+0.0001)
    return data-0.5

# get audio data without padding highest qualify audio
def load_file_data_without_change(folder,file_names, duration=3, sr=16000):
    input_length=sr*duration
    # function to load files and extract features
    # file_names = glob.glob(os.path.join(folder, '*.wav'))
    data = []
    for file_name in file_names:
        try:
            sound_file=folder+file_name
            print ("load file ",sound_file)
            # use kaiser_fast technique for faster extraction
            X, sr = librosa.load( sound_file, res_type='kaiser_fast')
            dur = librosa.get_duration(y=X, sr=sr)
            # extract normalized mfcc feature from data
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sr, n_mfcc=40).T,axis
=0)
        except Exception as e:
            print("Error encountered while parsing file: ", file)
        feature = np.array(mfccs).reshape([-1,1])
        data.append(feature)
    return data


# get audio data with a fix padding may also chop off some file
def load_file_data (folder,file_names, duration=12, sr=16000):
    input_length=sr*duration
    # function to load files and extract features
    # file_names = glob.glob(os.path.join(folder, '*.wav'))
    data = []
    for file_name in file_names:
        try:
            sound_file=folder+file_name
            print ("load file ",sound_file)
            # use kaiser_fast technique for faster extraction
            X, sr = librosa.load( sound_file, sr=sr, duration=duration,res_typ
e='kaiser_fast')
            dur = librosa.get_duration(y=X, sr=sr)
            # pad audio file same duration
            if (round(dur) < duration):
                print ("fixing audio lenght :", file_name)
                y = librosa.util.fix_length(X, input_length)
            #normalized raw audio
            # y = audio_norm(y)
            # extract normalized mfcc feature from data
            mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sr, n_mfcc=40).T,axis
=0)
        except Exception as e:
            print("Error encountered while parsing file: ", file)
        feature = np.array(mfccs).reshape([-1,1])
```

```
        data.append(feature)
    return data
```

In [45]:
```
# simple encoding of categories, limited to 3 types
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

# Map label text to integer
CLASSES = ['artifact','murmur','normal']
# {'artifact': 0, 'murmur': 1, 'normal': 3}
NB_CLASSES=len(CLASSES)

# Map integer value to text labels
label_to_int = {k:v for v,k in enumerate(CLASSES)}
print (label_to_int)
print (" ")
# map integer to label text
int_to_label = {v:k for k,v in label_to_int.items()}
print(int_to_label)
```

```
{'artifact': 0, 'murmur': 1, 'normal': 2}

{0: 'artifact', 1: 'murmur', 2: 'normal'}
```

In [46]:

```
# load dataset-a, keep them separate for testing purpose
import os, fnmatch

A_folder=INPUT_DIR+'/set_a/'
# set-a
A_artifact_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_a'), 'artifact*.w
av')
A_artifact_sounds = load_file_data(folder=A_folder,file_names=A_artifact_files
, duration=MAX_SOUND_CLIP_DURATION)
A_artifact_labels = [0 for items in A_artifact_files]

A_normal_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_a'), 'normal*.wav')
A_normal_sounds = load_file_data(folder=A_folder,file_names=A_normal_files, du
ration=MAX_SOUND_CLIP_DURATION)
A_normal_labels = [2 for items in A_normal_sounds]

A_extrahls_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_a'), 'extrahls*.w
av')
A_extrahls_sounds = load_file_data(folder=A_folder,file_names=A_extrahls_files
, duration=MAX_SOUND_CLIP_DURATION)
A_extrahls_labels = [1 for items in A_extrahls_sounds]

A_murmur_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_a'), 'murmur*.wav')
A_murmur_sounds = load_file_data(folder=A_folder,file_names=A_murmur_files, du
ration=MAX_SOUND_CLIP_DURATION)
A_murmur_labels = [1 for items in A_murmur_files]

# test files
A_unlabelledtest_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_a'), 'Aunla
belledtest*.wav')
A_unlabelledtest_sounds = load_file_data(folder=A_folder,file_names=A_unlabell
edtest_files, duration=MAX_SOUND_CLIP_DURATION)
A_unlabelledtest_labels = [-1 for items in A_unlabelledtest_sounds]

print ("loaded dataset-a")
```

```
load file  ../input/set_a/artifact__201106050353.wav
fixing audio lenght : artifact__201106050353.wav
load file  ../input/set_a/artifact__201106030612.wav
fixing audio lenght : artifact__201106030612.wav
load file  ../input/set_a/artifact__201106040947.wav
fixing audio lenght : artifact__201106040947.wav
load file  ../input/set_a/artifact__201106101314.wav
fixing audio lenght : artifact__201106101314.wav
load file  ../input/set_a/artifact__201106040933.wav
fixing audio lenght : artifact__201106040933.wav
load file  ../input/set_a/artifact__201106221254.wav
fixing audio lenght : artifact__201106221254.wav
load file  ../input/set_a/artifact__201106211041.wav
fixing audio lenght : artifact__201106211041.wav
load file  ../input/set_a/artifact__201106031558.wav
fixing audio lenght : artifact__201106031558.wav
load file  ../input/set_a/artifact__201106041452.wav
fixing audio lenght : artifact__201106041452.wav
load file  ../input/set_a/artifact__201105190800.wav
fixing audio lenght : artifact__201105190800.wav
load file  ../input/set_a/artifact__201106110909.wav
fixing audio lenght : artifact__201106110909.wav
load file  ../input/set_a/artifact__201106161016.wav
fixing audio lenght : artifact__201106161016.wav
load file  ../input/set_a/artifact__201105051017.wav
fixing audio lenght : artifact__201105051017.wav
load file  ../input/set_a/artifact__201106161219.wav
fixing audio lenght : artifact__201106161219.wav
load file  ../input/set_a/artifact__201105060108.wav
fixing audio lenght : artifact__201105060108.wav
load file  ../input/set_a/artifact__201106111119.wav
fixing audio lenght : artifact__201106111119.wav
load file  ../input/set_a/artifact__201106131834.wav
fixing audio lenght : artifact__201106131834.wav
load file  ../input/set_a/artifact__201106212112.wav
fixing audio lenght : artifact__201106212112.wav
load file  ../input/set_a/artifact__201105061143.wav
fixing audio lenght : artifact__201105061143.wav
load file  ../input/set_a/artifact__201106061233.wav
fixing audio lenght : artifact__201106061233.wav
load file  ../input/set_a/artifact__201106010559.wav
fixing audio lenght : artifact__201106010559.wav
load file  ../input/set_a/artifact__201106010602.wav
fixing audio lenght : artifact__201106010602.wav
load file  ../input/set_a/artifact__201106121242.wav
fixing audio lenght : artifact__201106121242.wav
load file  ../input/set_a/artifact__201106211430.wav
fixing audio lenght : artifact__201106211430.wav
load file  ../input/set_a/artifact__201106141701.wav
fixing audio lenght : artifact__201106141701.wav
load file  ../input/set_a/artifact__201012172012.wav
fixing audio lenght : artifact__201012172012.wav
load file  ../input/set_a/artifact__201106161019.wav
fixing audio lenght : artifact__201106161019.wav
load file  ../input/set_a/artifact__201106070537.wav
fixing audio lenght : artifact__201106070537.wav
load file  ../input/set_a/artifact__201106171003.wav
```

```
load file  ../input/set_a/Aunlabelledtest__201106061215.wav
fixing audio lenght : Aunlabelledtest__201106061215.wav
load file  ../input/set_a/Aunlabelledtest__201108222241.wav
fixing audio lenght : Aunlabelledtest__201108222241.wav
load file  ../input/set_a/Aunlabelledtest__201103011036.wav
fixing audio lenght : Aunlabelledtest__201103011036.wav
load file  ../input/set_a/Aunlabelledtest__201106171155.wav
fixing audio lenght : Aunlabelledtest__201106171155.wav
load file  ../input/set_a/Aunlabelledtest__201108011117.wav
fixing audio lenght : Aunlabelledtest__201108011117.wav
loaded dataset-a
```

In [47]:
```
%%time
# load dataset-b, keep them separate for testing purpose
B_folder=INPUT_DIR+'/set_b/'
# set-b
B_normal_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_b'), 'normal*.wav')
# include noisy files
B_normal_sounds = load_file_data(folder=B_folder,file_names=B_normal_files, du
ration=MAX_SOUND_CLIP_DURATION)
B_normal_labels = [2 for items in B_normal_sounds]

B_murmur_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_b'), 'murmur*.wav')
# include noisy files
B_murmur_sounds = load_file_data(folder=B_folder,file_names=B_murmur_files, du
ration=MAX_SOUND_CLIP_DURATION)
B_murmur_labels = [1 for items in B_murmur_files]

B_extrastole_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_b'), 'extrastol
e*.wav')
B_extrastole_sounds = load_file_data(folder=B_folder,file_names=B_extrastole_f
iles, duration=MAX_SOUND_CLIP_DURATION)
B_extrastole_labels = [1 for items in B_extrastole_files]

#test files
B_unlabelledtest_files = fnmatch.filter(os.listdir(INPUT_DIR+'/set_b'), 'Bunla
belledtest*.wav')
B_unlabelledtest_sounds = load_file_data(folder=B_folder,file_names=B_unlabell
edtest_files, duration=MAX_SOUND_CLIP_DURATION)
B_unlabelledtest_labels = [-1 for items in B_unlabelledtest_sounds]
print ("loaded dataset-b")
```

```
load file  ../input/set_b/normal__286_1311170606028_B1.wav
fixing audio lenght : normal__286_1311170606028_B1.wav
load file  ../input/set_b/normal__173_1307973611151_C.wav
fixing audio lenght : normal__173_1307973611151_C.wav
load file  ../input/set_b/normal__190_1308076920011_D.wav
fixing audio lenght : normal__190_1308076920011_D.wav
load file  ../input/set_b/normal__181_1308052613891_D.wav
fixing audio lenght : normal__181_1308052613891_D.wav
load file  ../input/set_b/normal_noisynormal_271_1309369876160_D.wav
load file  ../input/set_b/normal_noisynormal_117_1306262456650_C.wav
fixing audio lenght : normal_noisynormal_117_1306262456650_C.wav
load file  ../input/set_b/normal_noisynormal_144_1306522408528_C.wav
fixing audio lenght : normal_noisynormal_144_1306522408528_C.wav
load file  ../input/set_b/normal__176_1307988171173_B1.wav
fixing audio lenght : normal__176_1307988171173_B1.wav
load file  ../input/set_b/normal__184_1308073010307_D.wav
load file  ../input/set_b/normal__113_1306244002866_D.wav
fixing audio lenght : normal__113_1306244002866_D.wav
load file  ../input/set_b/normal_noisynormal_170_1307970562729_C1.wav
fixing audio lenght : normal_noisynormal_170_1307970562729_C1.wav
load file  ../input/set_b/normal__296_1311682952647_A2.wav
fixing audio lenght : normal__296_1311682952647_A2.wav
load file  ../input/set_b/normal_noisynormal_105_1305033453095_C.wav
fixing audio lenght : normal_noisynormal_105_1305033453095_C.wav
load file  ../input/set_b/normal__177_1307989650056_B.wav
fixing audio lenght : normal__177_1307989650056_B.wav
load file  ../input/set_b/normal__145_1307987561278_C.wav
load file  ../input/set_b/normal__168_1307970069434_A2.wav
fixing audio lenght : normal__168_1307970069434_A2.wav
load file  ../input/set_b/normal__181_1308052613891_B.wav
fixing audio lenght : normal__181_1308052613891_B.wav
load file  ../input/set_b/normal__170_1307970562729_C.wav
fixing audio lenght : normal__170_1307970562729_C.wav
load file  ../input/set_b/normal__217_1308246111629_C1.wav
fixing audio lenght : normal__217_1308246111629_C1.wav
load file  ../input/set_b/normal_noisynormal_278_1311163365896_B.wav
load file  ../input/set_b/normal__154_1306935608852_B.wav
fixing audio lenght : normal__154_1306935608852_B.wav
load file  ../input/set_b/normal__176_1307988171173_A.wav
fixing audio lenght : normal__176_1307988171173_A.wav
load file  ../input/set_b/normal__159_1307018640315_B1.wav
fixing audio lenght : normal__159_1307018640315_B1.wav
load file  ../input/set_b/normal_noisynormal_125_1306332456645_A2.wav
fixing audio lenght : normal_noisynormal_125_1306332456645_A2.wav
load file  ../input/set_b/normal__250_1309202496494_A.wav
fixing audio lenght : normal__250_1309202496494_A.wav
load file  ../input/set_b/normal__159_1307018640315_C1.wav
fixing audio lenght : normal__159_1307018640315_C1.wav
load file  ../input/set_b/normal__152_1306779561195_C1.wav
fixing audio lenght : normal__152_1306779561195_C1.wav
load file  ../input/set_b/normal_noisynormal_137_1306764999211_A2.wav
fixing audio lenght : normal_noisynormal_137_1306764999211_A2.wav
load file  ../input/set_b/normal__179_1307990076841_B.wav
fixing audio lenght : normal__179_1307990076841_B.wav
load file  ../input/set_b/normal_noisynormal_137_1306764999211_A1.wav
fixing audio lenght : normal_noisynormal_137_1306764999211_A1.wav
load file  ../input/set_b/normal__129_1306344506305_D1.wav
```

```
load file  ../input/set_b/Bunlabelledtest__161_1307101199321_D.wav
fixing audio lenght : Bunlabelledtest__161_1307101199321_D.wav
load file  ../input/set_b/Bunlabelledtest__154_1306935608852_D.wav
fixing audio lenght : Bunlabelledtest__154_1306935608852_D.wav
load file  ../input/set_b/Bunlabelledtest__186_1308073648738_D1.wav
fixing audio lenght : Bunlabelledtest__186_1308073648738_D1.wav
load file  ../input/set_b/Bunlabelledtest__148_1306768801551_C.wav
fixing audio lenght : Bunlabelledtest__148_1306768801551_C.wav
load file  ../input/set_b/Bunlabelledtest__109_1305653646620_B.wav
load file  ../input/set_b/Bunlabelledtest__287_1311170903290_D.wav
fixing audio lenght : Bunlabelledtest__287_1311170903290_D.wav
load file  ../input/set_b/Bunlabelledtest__272_1309370164386_B.wav
fixing audio lenght : Bunlabelledtest__272_1309370164386_B.wav
load file  ../input/set_b/Bunlabelledtest__254_1309350589009_B.wav
fixing audio lenght : Bunlabelledtest__254_1309350589009_B.wav
loaded dataset-b
CPU times: user 43.2 s, sys: 26.4 s, total: 1min 9s
Wall time: 36.1 s
```

In [48]:

```
#combine set-a and set-b
x_data = np.concatenate((A_artifact_sounds, A_normal_sounds,A_extrahls_sounds,
A_murmur_sounds,
                         B_normal_sounds,B_murmur_sounds,B_extrastole_sounds))

y_data = np.concatenate((A_artifact_labels, A_normal_labels,A_extrahls_labels,
A_murmur_labels,
                         B_normal_labels,B_murmur_labels,B_extrastole_labels))

test_x = np.concatenate((A_unlabelledtest_sounds,B_unlabelledtest_sounds))
test_y = np.concatenate((A_unlabelledtest_labels,B_unlabelledtest_labels))

print ("combined training data record: ",len(y_data), len(test_y))
```

```
combined training data record:  585 247
```

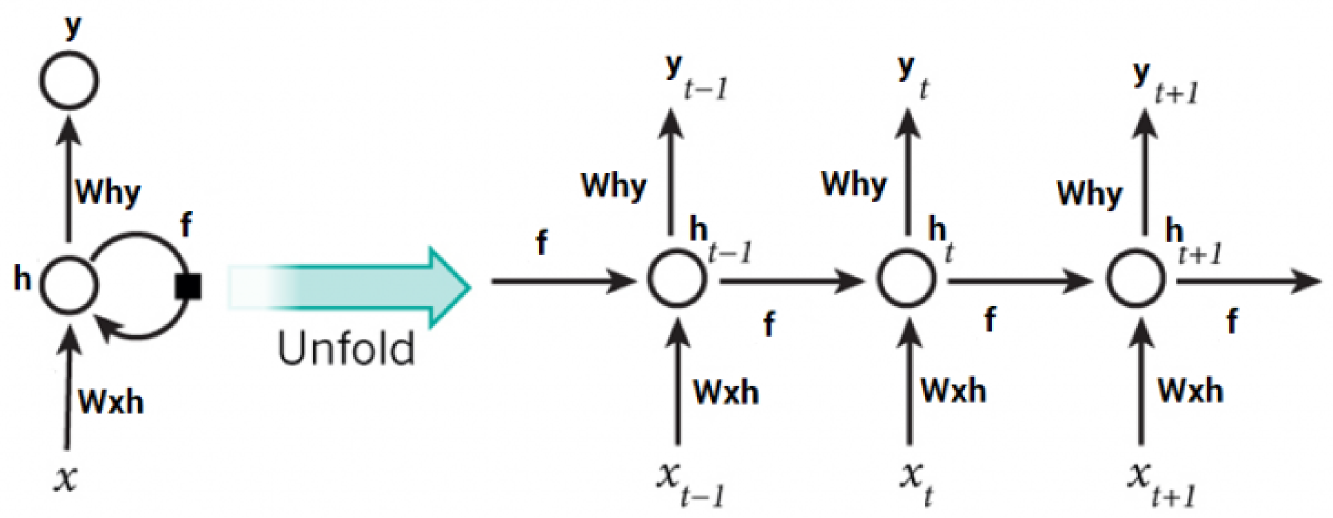
In [49]:

```
# shuffle - whether or not to shuffle the data before splitting. If shuffle=Fa
lse then stratify must be None.
# random_state is the seed used by the random number generator; If RandomState
instance, random_state is the random number generator; If None, the random num
ber generator is the RandomState instance used by np.random.

seed = 1000
# split data into Train, Validation and Test
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, train_size
=0.9, random_state=seed, shuffle=True)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size
=0.9, random_state=seed, shuffle=True)

# One-Hot encoding for classes
y_train = np.array(keras.utils.to_categorical(y_train, len(CLASSES)))
y_test = np.array(keras.utils.to_categorical(y_test, len(CLASSES)))
y_val = np.array(keras.utils.to_categorical(y_val, len(CLASSES)))
test_y=np.array(keras.utils.to_categorical(test_y, len(CLASSES)))
```

In [50]:
```
print ("label shape: ", y_data.shape)
print ("data size of the array: : %s" % y_data.size)
print ("length of one array element in bytes: ", y_data.itemsize)
print ("total bytes consumed by the elements of the array: ", y_data.nbytes)
print (y_data[1])
print ("")
print ("audio data shape: ", x_data.shape)
print ("data size of the array: : %s" % x_data.size)
print ("length of one array element in bytes: ", x_data.itemsize)
print ("total bytes consumed by the elements of the array: ", x_data.nbytes)
#print (x_data[1])
print ("")
print ("training data shape: ", x_train.shape)
print ("training label shape: ", y_train.shape)
print ("")
print ("validation data shape: ", x_val.shape)
print ("validation label shape: ", y_val.shape)
print ("")
print ("test data shape: ", x_test.shape)
print ("test label shape: ", y_test.shape)
```

```
label shape:  (585,)
data size of the array: : 585
length of one array element in bytes:  8
total bytes consumed by the elements of the array:  4680
0

audio data shape:  (585, 40, 1)
data size of the array: : 23400
length of one array element in bytes:  8
total bytes consumed by the elements of the array:  187200

training data shape:  (473, 40, 1)
training label shape:  (473, 3)

validation data shape:  (53, 40, 1)
validation label shape:  (53, 3)

test data shape:  (59, 40, 1)
test label shape:  (59, 3)
```

Deep learning RNN (Recurrent Neural Networks)-LSTM (Long Short-Term Memory)



LSTM network is comprised of different memory blocks called cells (the rectangles that we see in the image). There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates.

-RNN and LSTM are memory-bandwidth limited problems -Temporal convolutional network (TCN) “outperform canonical recurrent networks such as LSTMs across a diverse range of tasks and datasets, while demonstrating longer effective memory”.

In [51]:
```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, LSTM
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,Te
nsorBoard,ProgbarLogger
from keras.utils import np_utils
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
score
from sklearn.preprocessing import LabelEncoder
import itertools
```

Build Model

In [52]:
```
print('Build LSTM RNN model ...')
model = Sequential()
model.add(LSTM(units=64, dropout=0.05, recurrent_dropout=0.20, return_sequence
s=True,input_shape = (40,1)))
model.add(LSTM(units=32, dropout=0.05, recurrent_dropout=0.20, return_sequence
s=False))
model.add(Dense(len(CLASSES), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='Adamax', metrics=['a
cc','mse', 'mae', 'mape', 'cosine'])
model.summary()
```

```
Build LSTM RNN model ...
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 40, 64)            16896
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                12416
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 29,411
Trainable params: 29,411
Non-trainable params: 0
_________________________________________________________________
```

Train Model

In [53]:

```
%%time
# saved model checkpoint file
best_model_file="./best_model_trained.hdf5"
#train_model_file=file_path+"/checkpoints/weights.best_{epoch:02d}-{loss:.2f}.
hdf5"
MAX_PATIENT=12
MAX_EPOCHS=100
MAX_BATCH=32

# callbacks
# removed EarlyStopping(patience=MAX_PATIENT)
callback=[ReduceLROnPlateau(patience=MAX_PATIENT, verbose=1),
          ModelCheckpoint(filepath=best_model_file, monitor='loss', verbose=1,
save_best_only=True)]

print ("training started..... please wait.")
# training
history=model.fit(x_train, y_train,
                  batch_size=MAX_BATCH,
                  epochs=MAX_EPOCHS,
                  verbose=0,
                  validation_data=(x_val, y_val),
                  callbacks=callback)

print ("training finised!")
```

```
training started..... please wait.

Epoch 00001: loss improved from inf to 1.00303, saving model to ./best_model_
trained.hdf5

Epoch 00002: loss improved from 1.00303 to 0.81297, saving model to ./best_mo
del_trained.hdf5

Epoch 00003: loss improved from 0.81297 to 0.72901, saving model to ./best_mo
del_trained.hdf5

Epoch 00004: loss improved from 0.72901 to 0.71217, saving model to ./best_mo
del_trained.hdf5

Epoch 00005: loss improved from 0.71217 to 0.68847, saving model to ./best_mo
del_trained.hdf5

Epoch 00006: loss improved from 0.68847 to 0.67817, saving model to ./best_mo
del_trained.hdf5

Epoch 00007: loss did not improve from 0.67817

Epoch 00008: loss did not improve from 0.67817

Epoch 00009: loss improved from 0.67817 to 0.65555, saving model to ./best_mo
del_trained.hdf5

Epoch 00010: loss improved from 0.65555 to 0.65520, saving model to ./best_mo
del_trained.hdf5

Epoch 00011: loss did not improve from 0.65520

Epoch 00012: loss did not improve from 0.65520

Epoch 00013: loss did not improve from 0.65520

Epoch 00014: loss improved from 0.65520 to 0.63898, saving model to ./best_mo
del_trained.hdf5

Epoch 00015: loss did not improve from 0.63898

Epoch 00016: loss did not improve from 0.63898

Epoch 00017: loss did not improve from 0.63898

Epoch 00018: loss improved from 0.63898 to 0.62734, saving model to ./best_mo
del_trained.hdf5

Epoch 00019: loss did not improve from 0.62734

Epoch 00020: loss improved from 0.62734 to 0.62509, saving model to ./best_mo
del_trained.hdf5

Epoch 00021: loss did not improve from 0.62509

Epoch 00022: loss improved from 0.62509 to 0.62353, saving model to ./best_mo
del_trained.hdf5
```

```
Epoch 00023: loss did not improve from 0.62353

Epoch 00024: loss did not improve from 0.62353

Epoch 00025: loss did not improve from 0.62353

Epoch 00026: loss improved from 0.62353 to 0.61419, saving model to ./best_mo
del_trained.hdf5

Epoch 00027: loss did not improve from 0.61419

Epoch 00028: loss did not improve from 0.61419

Epoch 00029: loss did not improve from 0.61419

Epoch 00030: loss improved from 0.61419 to 0.60609, saving model to ./best_mo
del_trained.hdf5

Epoch 00031: ReduceLROnPlateau reducing learning rate to 0.000200000009499490
26.

Epoch 00031: loss did not improve from 0.60609

Epoch 00032: loss did not improve from 0.60609

Epoch 00033: loss did not improve from 0.60609

Epoch 00034: loss did not improve from 0.60609

Epoch 00035: loss did not improve from 0.60609

Epoch 00036: loss did not improve from 0.60609

Epoch 00037: loss did not improve from 0.60609

Epoch 00038: loss improved from 0.60609 to 0.59986, saving model to ./best_mo
del_trained.hdf5

Epoch 00039: loss did not improve from 0.59986

Epoch 00040: loss did not improve from 0.59986

Epoch 00041: loss did not improve from 0.59986

Epoch 00042: loss improved from 0.59986 to 0.59737, saving model to ./best_mo
del_trained.hdf5

Epoch 00043: ReduceLROnPlateau reducing learning rate to 2.0000000949949027e-
05.

Epoch 00043: loss did not improve from 0.59737

Epoch 00044: loss did not improve from 0.59737

Epoch 00045: loss did not improve from 0.59737
```

```
Epoch 00046: loss did not improve from 0.59737

Epoch 00047: loss did not improve from 0.59737

Epoch 00048: loss did not improve from 0.59737

Epoch 00049: loss did not improve from 0.59737

Epoch 00050: loss did not improve from 0.59737

Epoch 00051: loss did not improve from 0.59737

Epoch 00052: loss did not improve from 0.59737

Epoch 00053: loss did not improve from 0.59737

Epoch 00054: loss did not improve from 0.59737

Epoch 00055: ReduceLROnPlateau reducing learning rate to 2.0000001313746906e-
06.

Epoch 00055: loss did not improve from 0.59737

Epoch 00056: loss did not improve from 0.59737

Epoch 00057: loss did not improve from 0.59737

Epoch 00058: loss did not improve from 0.59737

Epoch 00059: loss did not improve from 0.59737

Epoch 00060: loss did not improve from 0.59737

Epoch 00061: loss did not improve from 0.59737

Epoch 00062: loss did not improve from 0.59737

Epoch 00063: loss did not improve from 0.59737

Epoch 00064: loss did not improve from 0.59737

Epoch 00065: loss did not improve from 0.59737

Epoch 00066: loss did not improve from 0.59737

Epoch 00067: ReduceLROnPlateau reducing learning rate to 2.000000222324161e-0
7.

Epoch 00067: loss did not improve from 0.59737

Epoch 00068: loss did not improve from 0.59737

Epoch 00069: loss did not improve from 0.59737

Epoch 00070: loss did not improve from 0.59737

Epoch 00071: loss did not improve from 0.59737
```

```
Epoch 00072: loss did not improve from 0.59737

Epoch 00073: loss did not improve from 0.59737

Epoch 00074: loss did not improve from 0.59737

Epoch 00075: loss did not improve from 0.59737

Epoch 00076: loss did not improve from 0.59737

Epoch 00077: loss did not improve from 0.59737

Epoch 00078: loss did not improve from 0.59737

Epoch 00079: ReduceLROnPlateau reducing learning rate to 2.000000165480742e-0
8.

Epoch 00079: loss did not improve from 0.59737

Epoch 00080: loss did not improve from 0.59737

Epoch 00081: loss did not improve from 0.59737

Epoch 00082: loss did not improve from 0.59737

Epoch 00083: loss did not improve from 0.59737

Epoch 00084: loss did not improve from 0.59737

Epoch 00085: loss did not improve from 0.59737

Epoch 00086: loss did not improve from 0.59737

Epoch 00087: loss did not improve from 0.59737

Epoch 00088: loss did not improve from 0.59737

Epoch 00089: loss did not improve from 0.59737

Epoch 00090: loss did not improve from 0.59737

Epoch 00091: ReduceLROnPlateau reducing learning rate to 2.000000165480742e-0
9.

Epoch 00091: loss did not improve from 0.59737

Epoch 00092: loss did not improve from 0.59737

Epoch 00093: loss did not improve from 0.59737

Epoch 00094: loss did not improve from 0.59737

Epoch 00095: loss did not improve from 0.59737

Epoch 00096: loss did not improve from 0.59737
```

```
Epoch 00097: loss did not improve from 0.59737

Epoch 00098: loss did not improve from 0.59737

Epoch 00099: loss improved from 0.59737 to 0.59707, saving model to ./best_mo
del_trained.hdf5

Epoch 00100: loss did not improve from 0.59707
training finised!
CPU times: user 6min 13s, sys: 37.9 s, total: 6min 51s
Wall time: 4min 36s
```

Model Evaluation

In [54]:

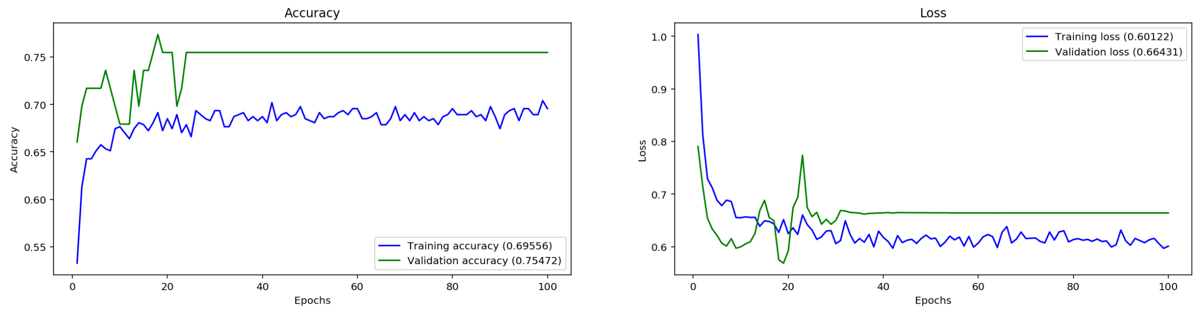
```
# Keras reported accuracy:
score = model.evaluate(x_train, y_train, verbose=0)
print ("model train data score       : ",round(score[1]*100) , "%")

score = model.evaluate(x_test, y_test, verbose=0)
print ("model test data score        : ",round(score[1]*100) , "%")

score = model.evaluate(x_val, y_val, verbose=0)
print ("model validation data score  : ", round(score[1]*100), "%")

score = model.evaluate(test_x, test_y, verbose=0)
print ("model unlabeled data score   : ", round(score[1]*100), "%")
```

```
model train data score       :  70.0 %
model test data score        :  69.0 %
model validation data score  :  75.0 %
model unlabeled data score   :  80.0 %
```

In [55]:

```
%%time
#Plot Keras History
#Plot loss and accuracy for the training and validation set.
def plot_history(history):
    loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]
    if len(loss_list) == 0:
        print('Loss is missing in history')
        return
    plt.figure(figsize=(22,10))
    ## As loss always exists
    epochs = range(1,len(history.history[loss_list[0]]) + 1)
    ## Accuracy
    plt.figure(221, figsize=(20,10))
    ## Accuracy
    # plt.figure(2,figsize=(14,5))
    plt.subplot(221, title='Accuracy')
    for l in acc_list:
        plt.plot(epochs, history.history[l], 'b', label='Training accuracy (' + str(format(history.history[l][-1],'.5f'))+')')
    for l in val_acc_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation accuracy (' + str(format(history.history[l][-1],'.5f'))+')')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    ## Loss
    plt.subplot(222, title='Loss')
    for l in loss_list:
        plt.plot(epochs, history.history[l], 'b', label='Training loss (' + str(str(format(history.history[l][-1],'.5f'))+')'))
    for l in val_loss_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation loss (' + str(str(format(history.history[l][-1],'.5f'))+')'))
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# plot history
plot_history(history)
```

```
<Figure size 1584x720 with 0 Axes>
```




```
CPU times: user 1.2 s, sys: 264 ms, total: 1.46 s
Wall time: 1.12 s
```

In [56]:
```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        title='Normalized confusion matrix'
    else:
        title='Confusion matrix'

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

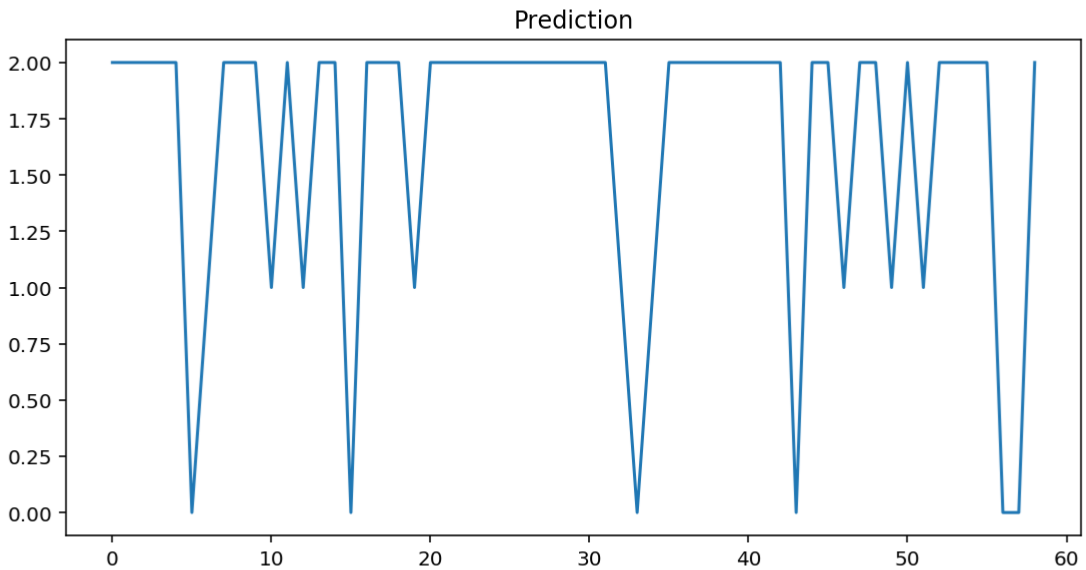
Prediction Test

make a prediction x: The input data, as a Numpy array (or list of Numpy arrays if the model has multiple inputs). batch_size: Integer. If unspecified, it will default to 32. steps = Total number of steps (batches of samples) before declaring the prediction round finished. callbacks: List of keras.callbacks.Callback instances. returns Numpy array(s) of predictions.

In [57]:
```
# prediction class
y_pred = model.predict_classes(x_test, batch_size=32)
print ("prediction test return :",y_pred[1], "-", int_to_label[y_pred[1]])
```

```
prediction test return : 2 - normal
```

In [58]:
```
plt.figure(1,figsize=(20,10))
# plot Classification Metrics: Accuracy
plt.subplot(221, title='Prediction')
plt.plot(y_pred)
plt.show()
```



Loading a saved training model

In [59]:
```
print (best_model_file)
```

```
./best_model_trained.hdf5
```

In [60]:
```
### Loading a Check-Pointed Neural Network Model
# How to load and use weights from a checkpoint
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import numpy
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# create model
print('Build LSTM RNN model ...')
model = Sequential()
model.add(LSTM(units=64, dropout=0.05, recurrent_dropout=0.35, return_sequence
s=True,input_shape = (40,1)))
model.add(LSTM(units=32, dropout=0.05, recurrent_dropout=0.35, return_sequence
s=False))
model.add(Dense(len(CLASSES), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
c','mse', 'mae', 'mape', 'cosine'])
model.summary()
# load weights
model.load_weights(best_model_file)
# Compile model (required to make predictions)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
print("Created model and loaded weights from file")
```

```
Build LSTM RNN model ...
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (None, 40, 64)            16896     
_________________________________________________________________
lstm_4 (LSTM)                (None, 32)                12416     
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 99        
=================================================================
Total params: 29,411
Trainable params: 29,411
Non-trainable params: 0
_________________________________________________________________
Created model and loaded weights from file
```

Test loaded model

In [61]:
```
# make a prediction
y_pred = model.predict_classes(x_test, batch_size=32)
#check scores
scores = model.evaluate(x_test, y_test, verbose=0)
print ("Model evaluation accuracy: ", round(scores[1]*100),"%")
```

```
Model evaluation accuracy:  79.0 %
```

Conclusion:

In this report, the process of classifying audio heart sounds is presented and machine learning techniques for this task are compared. Two audio heart sound datasets are used to train and verify the six chosen methods. The process to classify heartbeats includes preprocessing the datasets, extracting audio features, training methods and finally analyzing results.

Preprocessing was done to normalize the data. Feature extraction then uses the preprocessed data to extract features using the whole signal and significant parts of the signal, such as S1 and S2. Using the feature extracted, we build our LSTM model. After the compilation of which, we can determine the category of any unlabeled heart sounds. Our Model was successful to get an accuracy of **79 %** to predict the correct category of the unlabeled Heart Sound.

As the future objective, we shall try to improve the accuracy score of our LSTM model in the future.

References:

- Classifying Heart Sounds Challenge
  http://www.peterjbentley.com/heartchallenge/

- "Diagnosis & Tests", WebMD. [Online]. Available: https://www.webmd.com/heart-disease/guide/heart-disease-diagnosis-tests

- "Heart Disease", Mayo Clinic. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/heart-disease/diagnosis-treatment/drc-20353124

- "Heart Murmurs - Heart Sounds", Practical Clinical Skills. [Online]. Available: https://www.practicalclinicalskills.com/heart-murmurs

- "Medical Definition of Extrasystole", MedicineNet.com. [Online]. Available:
  https://www.medicinenet.com/script/main/art.asp?articlekey=32159