Accelerating biomedical signal processing using GPU: a case study of snore sound feature extraction

Jian Guo, Kun Qian, Gongxuan Zhang, Huijie Xu, Björn Schuller

Angaben zur Veröffentlichung / Publication details:

Guo, Jian, Kun Qian, Gongxuan Zhang, Huijie Xu, and Björn Schuller. 2017. "Accelerating biomedical signal processing using GPU: a case study of snore sound feature extraction." *Interdisciplinary Sciences: Computational Life Sciences* 9 (4): 550–55. https://doi.org/10.1007/s12539-017-0232-9.

Accelerating Biomedical Signal Processing Using GPU: A Case Study of Snore Sound Feature Extraction

Jian Guo[1] · Kun Qian[2] · Gongxuan Zhang[1] · Huijie Xu[3] · Björn Schuller[4]

Abstract The advent of 'Big Data' and 'Deep Learning' offers both, a great challenge and a huge opportunity for personalised health-care. In machine learning-based biomedical data analysis, feature extraction is a key step for 'feeding' the subsequent classifiers. With increasing numbers of biomedical data, extracting features from these 'big' data is an intensive and time-consuming task. In this case study, we employ a Graphics Processing Unit (GPU) via Python to extract features from a large corpus of snore sound data. Those features can subsequently be imported into many well-known deep learning training frameworks without any format processing. The snore sound data were collected from several hospitals (20 subjects, with 770–990 MB per subject – in total 17.20 GB). Experimental results show that our GPU-based processing significantly speeds up the feature extraction phase, by up to seven times, as compared to the previous CPU system.

Gongxuan Zhang
gongxuan@njust.edu.cn

Jian Guo
johnkuo83@gmail.com

Kun Qian
andykun.qian@tum.de

Huijie Xu
xhj0531@163.com

Björn Schuller
schuller@ieee.org

1 School of Computer Science and Engineering, Nanjing University of Science Technology, Nanjing, China

2 Department of Electrical and Computer Engineering, MISP group, MMK Technische University Munchen, Munich, Germany

3 Department of Otolaryngology, Beijing Hospital, Beijing, China

4 Bjorn Schuller Department of Computing, Machine Learning Group Imperial College London, London, UK

1 Introduction

Biomedical engineering is gaining more attention from researchers in both life sciences and computer science. The mass size of currently available biomedical data and its high rate of growth has great influence on the current research related to its mining [1, 2]. In fact, the exponentially increasing amount of biomedical data turns mining such data into a 'big data' problem [3]. Thus, High-Performance Computing (HPC) will be the core infrastructure enabling doctors, biologists, and engineers to manage and analyze the data collected [4]. To name but three examples, Bastrakov et al. presented an HPC-based biomedical computing system to analyze the performance and improve the efficiency within a large-scale biomedical information simulation on cluster systems [4]. Qian et al. studied how to use a private cloud computing system to process longer durations of snore-related signals [5], which considerably speeds up the snore sound processing, than compared with single CPU systems. Ioana Dogaru and Radu Dogaru implemented some natural computing complexity algorithms with Python-based high-performance platforms, and achieved better speeds than Matlab/Octave environments, using the same hardware configurations [6].

In our study, we further the works in [5], and present a Graphics Processing Unit (GPU)-based method to rapidly extract acoustic features from large amounts of snore sound

(SnS) data via Python programming. The SnS data were collected from 20 subjects suffering from Obstructive Sleep Apnea (OSA) [7], a serious chronic breathing condition. Numerous researchers and scholars focused on acoustic features analysis of SnS generated by patients over the past decade or 2 [8, 9]. It is essential for doctors to understand the obstruction site and collapse degree of the upper airway (UA) by utilizing non-invasive measurement approaches like acoustic analysis of SnS [10–13]. First, we would like to give a brief description on the feature extraction, and how to utilize GPU-based Python programming to speed up this process; then, we describe our experiment settings and discuss the results before giving our conclusion.

2 Acoustic Features

In this case study, we extract 20 acoustic features from a large amount of SnS data, based on Fast Fourier Transform (FFT), which can be visualized to help doctors and specialists to diagnose, research, and remedy the diseases efficiently. An energy threshold was set to detect the short-time energy of each frame segmented from SnS data. From frames exceeding an energy threshold, we extract 20 acoustic features which reflect the frequency and spectrum distribution. We extracted the center, peak, and mean points of the spectrum ($f_{\text{center}}, f_{\text{peak}}, f_{\text{mean}}$) in Eqs. 1 and 2, where S_{fi} is the absolute amplitude of SnS spectrum at the frequency of f_i Hz calculated by Fast Fourier Transform (FFT). f_c is the cut-off frequency of the SnS spectrum, which is 8 kHz in our study. Then, we propose Eq. 3 to define f_{mean}, which is a good indicator to reflect the structure of an SnS spectrum. For comparing and analysing performance between CPU and GPU, we set those three features ($f_{\text{center}}, f_{\text{peak}}, f_{\text{mean}}$) as feature group 1:

$$\text{s.t.} \sum_{f_i=0}^{f_{\text{center}}} S_{f_i} = \sum_{f_i=f_{\text{center}}}^{f_c} S_{f_i} \tag{1}$$

$$\text{s.t.}\, S_{f_{\text{peak}}} = \max\{S_{f_i}, f_i = 0, \cdots, f_c\} \tag{2}$$

$$f_{\text{mean}} = \frac{\sum_{f_i=0}^{f_c} f_i * S_{f_i}}{\sum_{f_i=0}^{f_c} S_{f_i}}. \tag{3}$$

Furthermore, in Eq. 4, eight additional spectral features, i. e., $f_{\text{mean}(1)}, f_{\text{mean}(2)}, \ldots, f_{\text{mean}(8)}$, in each 1000 Hz band are extracted for revealing the detailed information about the spectrum structure in each sub-band of SnS. We set $f_{\text{mean}(1\ldots8)}$ as feature group 2:

$$f_{\text{mean(k)}} = \frac{\sum_{f_i=1000*(k-1)}^{1000*k} f_i * S_{f_i}}{\sum_{f_i=1000*(k-1)}^{1000*k} S_{f_i}} \quad k = 1, 2, \ldots, 8. \tag{4}$$

The power ratio at a frequency of 800 Hz is known to be efficient to distinguish SnS generated by different obstruction sites in UA [14]. We define this feature as feature group 3 with Eq. 5:

$$\text{PR}_{800} = \lg\left(\frac{\sum_{f_i=0}^{800} S_{f_i}^2}{\sum_{f_i=800}^{f_c} S_{f_i}^2}\right). \tag{5}$$

Similar to PR_{800}, sub-band energy ratios (SER) are a good indicator to reflect the spectrum distribution of SnS [15]. In our study, each SER in the 1000 Hz band is calculated; thus, we obtain eight SERs as our full 8000 Hz SnS spectrum. From $\text{SER}_{(1)}$ to $\text{SER}_{(8)}$, there are 8 features to be calculated and defined in Eq. 6 as feature group 4:

$$\text{SER}_{(k)} = \frac{\sum_{f_i=1000*(k-1)}^{1000*k} S_{f_i}^2}{\sum_{f_i=0}^{f_c} S_{f_i}^2} \quad k = 1, 2, \ldots, 8. \tag{6}$$

All extracted feature data could be reviewed according to the physicians preference; meanwhile, these data could also be processed by other data mining or machine learning algorithms, including: [16].

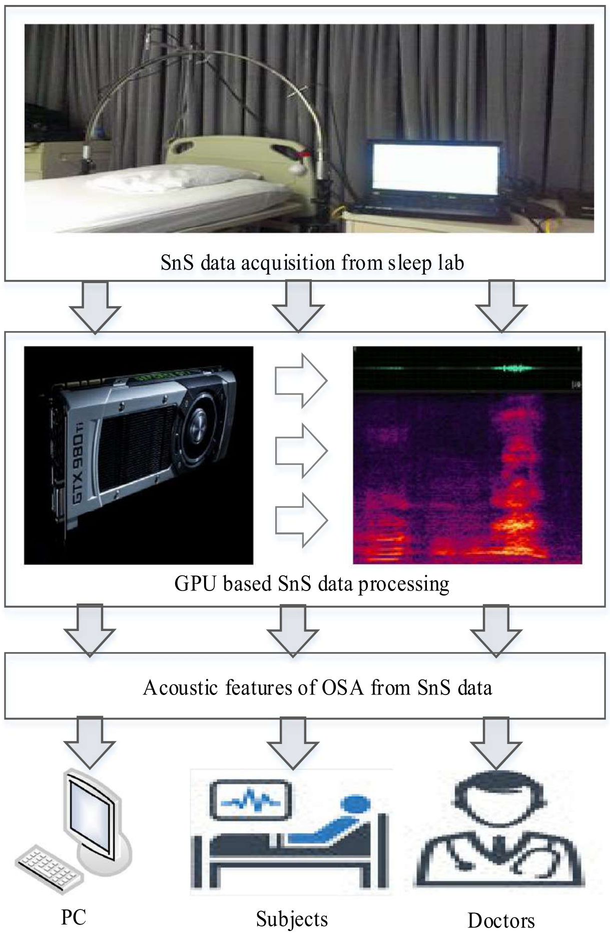

SnS data acquisition from sleep lab
GTX 980 Ti
GPU based SnS data processing
Acoustic features of OSA from SnS data
PC
Subjects
Doctors

Fig. 1 GPU-based system for processing SnS data collected in hospitals

3 System Framework

Figure 1 shows our GPU-based system for processing the large SnS data. The data were collected by a high-quality microphone setup. This system was hung at roughly 0.5 m above the subjects mouth (moving range between 0.4 and 0.8 m according to the movements of subjects). Sampling rate is 16 kHz, and 16-bit resolution was selected to make recording an entire night (8 h) of snoring possible, from 20 subjects. Considering the hardware, the GPU can achieve better performance for linear algebra-based acoustic analysis tasks.

As to the software-side matters, owing to recent toolkits such as Numpy [17], Scipy [18], and alike, Python's efficiency in scientific computing has vastly improved in recent years [19]. We select Python as our programming language due to its high programming efficiency. For example, during algorithm debugging, researchers prefer to program with interpretative languages like MathWorks Matlab and Python, because of their high programming efficiency and their saving of lots of time on coding. As well as code release, with assistance from specialists of high-performance computing, algorithms can be optimized into compiled languages such as C or C++, even parallelization by CPUs or GPUs for high running performance is possible. However, coding and optimization of high-performance parallelization programs are not only time-consuming but too advanced for most bioinformation researchers. Our Python-based method is both programming efficient and high running performance. Besides, most of the mainstream deep learning frameworks such as Caffe [20], MXNet [21], Tensorflow [22], Theano [23], etc. and alike interfaces are implemented via Python or are even coded directly into Python natively. With feature extraction in Python, programmers have an easier time linking up the feature extractor with Python-based machine learning packages such as scikit-learn [24] or the deep learning training frameworks mentioned above. For GPU programming, the Anaconda Python distribution with Numba/NumbaPro [25] supporting NVIDIA's CUDA-based GPU programming is chosen by us for speeding up our feature extraction. Numba offers an elegant way to accelerate computation with some minor changes in the original Python code. Likewise, equivalent performance of C++ and the like can be reached. This effect is due to just-in-time (JIT) code compilation. In addition, NumbaPro provides a CUDA-based API for calling CUDA libraries including CUDA BLAS Library (cuBLAS), Fast Fourier Transform library (cuFFT), CUDA Sparse (cuSPARSE), and CUDA Random Number Generation library (cuRAND) [25]. In our work, first, we import NumbaPro to call cuFFT for accelerating FFT computation in our feature extraction algorithms. Then, we import Numba, and modify the original Python codes by adding '@jit' decoration before defining functions to accelerate most of the remaining feature extraction programs; furthermore, we can achieve a better acceleration if we declare variables types for '@jit' labeled functions. The example codes in Listing 1 show those accelerate methods which we mentioned above. Except for adding some decorations like '@jit' into original Python code, we can see that there are almost no large-scale modifications on our original Python code.

Listing 1 Accelerating Python with Numba and NumbaPro

```
import numpy as np
from numba.decorators import jit
from numbapro import cuda
from numbapro.cudalib import cufft
@jit('f8[:],i4', nopython=True )
def feature_g(r,fs):
    nf = len(r)
    ap = np.sum(r[:nf/2 - 1])
    result = []
def main():
    threadperblock = 32, 16
    blockpergrid = best_grid_size(
    tuple(reversed(frames)), threadperblock)

    cufft.FFTPlan(frames, itype=np.float32
    ,otype=np.float32)

    fftplan1 = cufft.FFTPlan(frames,
    itype=np.float32 ,otype=np.float32)
```

Table 1 Configuration of the experimental setup

	Configuration
CPU	Intel Xeon Processor E5-2650 v3 @ 2.30 GHz
GPU	NVIDIA GTX 980Ti with 6 GB GDDR5
Memory	256 GB DDR3 1600
Storage	HDD 1.5 TB 7200 rpm
OS	CentOS 7.2 with CUDA 7.5
Python	Anaconda Python 2.7 with numpy 1.10.4 scipy 0.17, Numba/Numbapro 0.23.1

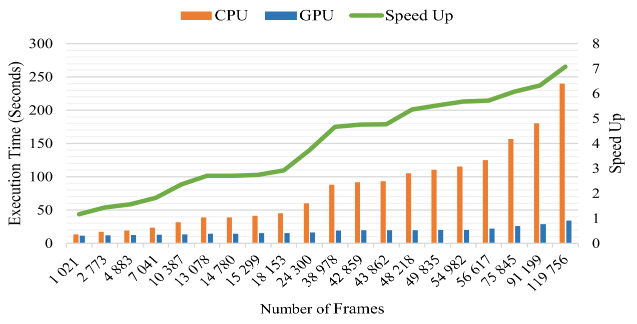


Fig. 2 Execution time of the CPU and the GPU, accordingly speed up for processing SnS data by individual subjects (these are given by the numbers of frames)

Table 2 SnS data as employed during testing

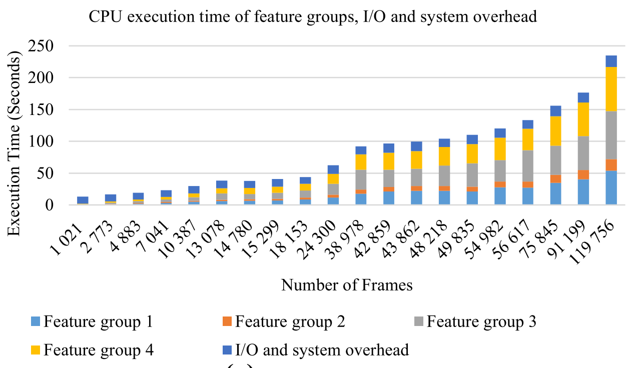
Subject index	1	2	3	4	5
Data size (MB)	770	990	880	880	770
Frames processed	54,982	49,835	10,387	2773	4883
Subject index	6	7	8	9	10
Data size (MB)	770	880	770	770	770
Frames processed	48,218	119,756	38,978	56,617	18,153
Subject index	11	12	13	14	15
Data size (MB)	770	880	880	880	880
Frames processed	7041	75,845	13,078	15,299	24,300
Subject index	16	17	18	19	20
Data size (MB)	880	770	880	880	880
Frames processed	43,862	14,780	1021	42,859	91,199

4 Experiments and Results

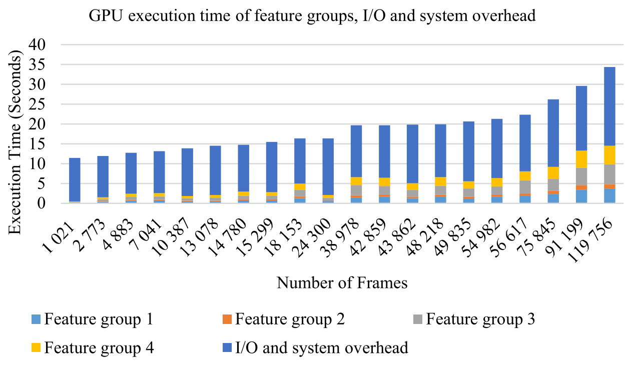
The SnS data in this work are provided by the Department of Otolaryngology, Beijing Hospital, P.R. China. Overall, the above described 20 acoustic features (cf. Sect. 2) from frames which passed a defined energy threshold are extracted from audio recordings of 20 subjects' entire night of sleep (8 h). The system configuration of our experimental environment is given in detail in Table 1.

As can be seen in Fig. 2, our proposed GPU-based system considerably accelerates the SnS data processing as compared to the CPU platform (sorted by processed numbers of frames per subject from small to large). Table 2 gives details on the SnS data as used in the experiments. From Fig. 2, we find that, for the subject that has the smallest number of frames processed (subject 18, 1021 frames), a comparably smaller speed-up is reached: 13.17 s are required by the CPU, and 11.36 s by the GPU (only speed up by 1.15x). In contrast, feature extraction for the data of the subject with the highest number of processed frames (subject 7, 119 756 frames) demands for 240.13 s by the CPU, but only 33.95 s when computed by the GPU (resembling a speed up by 7.0x). Note that among the 20 subjects' SnS data, even within a similar data size of audio recordings, the speed-up rate can differ. This is simply caused due to potentially different frames numbers needing to be processed. Even if the audio data are of almost the same size owing to the pre-selection of frames to be processed by the snore-activity detection. Independent of this trivial fact, one can summarize that, the more frames from the need processing in the audio data, the higher and more significant our methods acceleration effect will be.

Furthermore, we did a deeper performance analysis of the feature groups that we defined previously in Sect. 2 for understanding why we achieved different speed-up from different subjects, with the possibility to achieve better speed-up with a GPU. As we defined in Sect. 2, we divide 20 acoustic features into 4 feature groups. We record the execution times of these feature groups and file I/O times of subject data (including slight system overhead) separately using profile functions. As shown in part (a) of Fig. 3. First, according to the different complexity feature extraction algorithms, we describe, in Sect 2, feature groups cost differing percentages of total execution time with each subject. Let us take the most time-consuming one (subject who has 119756 frames) as an example, feature group 1–4 costs 53.66, 18.55, 75.14, and 69.48 s, respectively (22.83, 7.89, 31.97 and 29.57% of total execution time); in addition, I/O and system overheads cost 18.13 s (7.71% of total execution time at that subject). Besides this, subject with only 1021 frames also costs 10.93 s for I/O operation, 83.88% of total execution time (13.03 s). From this, we can see clearly that, no matter the data size of audio recordings, and despite the amount of frames need for computation, the I/O operation always costs around 11–14 s (770–990 MB per subject). Therefore, we can confirm that, for CPU-based feature extractions, different feature extraction algorithms have different time costs due to there being different complexities, and as previously shown, the more processing need for frames in the audio data, the more time needed. However, the I/O operation is not affected by total amount of frames to process, every subject costs around 11–14 s for I/O overhead in processing.



(a) CPU Execution Time



(b) GPU Execution Time

Fig. 3 Execution time of feature groups, I/O, and system overhead

Table 3 Mean value and standard deviation of execution time between CPU and GPU with ten replicates

	Mean	Std.,dev.
CPU	1.6361	±8.1055
GPU	0.3662	±2.2523

In part (b) of Fig. 3, of the GPU-based accelerate processing. We can see that feature group 1–4 have been accelerated and the execution times of feature groups are reduced significantly. However, just like CPU-based processing, each subject still costs around 11–14 s on subject data I/O operations with GPU-based processing. Let us consider the most time-consuming example, subject who has 119756 frames, in GPU-based processing, feature group 1–4 costs 3.59, 1.24, 5.03, and 4.65 s, respectively (10.45, 3.60, 14.64 and 13.53% of total execution time); meanwhile, I/O and system overhead costs 19.82 s (57.70% of total execution time at that subject). I/O operation costs more than half of the total execution time in this case. And in the case subjects with 1021 frames, the I/O operation reached the proportion of 96.07% of total execution time (11.01 seconds at I/O and 11.46 in total). Although we have accelerated feature extractions in SnS data processing with our methods, we cannot accelerate I/O operation in this way, which significantly reduces the speed-up ratio when comparing CPU and GPU. Replacing the storage from hard disk to SSD may theoretically reduce the I/O operation time costs, at the same time, to achieve better speed-up values.

The mean value and stand deviation of execution time compared CPU to GPU with ten replicated experimental results, as shown in Table 3. We can see that the GPU-based method has not only better computational ability but also better performance stability compared with the CPU-based processing.

5 Conclusion

In this work, we proposed a GPU-based system for processing snore sound data, which considerably outperforms the traditional CPU-based computing platforms. We utilized Python programming to provide a direct interface for fast GPU implementation of our signal processing algorithms. We found it highly efficient to accelerate the processing given the suited hardware. The experimental results showed that the more frames needed to be processed, the higher the system's speed-up. We also analysed and revealed deeply the affect between computation (both CPU and GPU based) and I/O operation for speed-up. Finally, we propose that I/O operation is an important factor which cannot be ignored when trying to accelerate big biomedical signal processing. Despite the programming languages or strategy used, the results should be consistent between algorithms. Moreover, after the computation time is optimized to a certain extent, I/O operation and unparalleled parts become the main obstacle to limit the running performance of the whole program.

In our future work, we would like to upgrade our storage from hard disk to SSD and to introduce cache policies into experiments as we suggest that this will achieve better speed-up. We will further advance related biomedical signal processing via MPI-based multiple CPUs, and multiple GPU clusters, to handle the increasing 'big' data collected within health-care and well-being field. As well as this, some more sophisticated signal processing algorithms like the most recent work of [26], will also be implemented along with GPUs for acceleration purposes.

6 Acknowledgements

This work is partially supported by National Natural Science Foundation of China (NSFC) under Grant no.,61472189, 61272420, the China Scholarship Council

(CSC), and the European Union's Seventh Framework under Grant Agreements no.,338164 (ERC Starting Grant iHEARu).

References

1. H. Ali (2015) Big data analytics in biomedical informatics, BIOSTEC. Tech Rep
2. M. S. Kamal and S. F. Nimmy (2016) Strucbreak: a computational framework for structural break detection in dna sequences. Interdiscip Sci 1–16
3. Grossman R, White K (2012) A vision for a biomedical cloud. J Intern Med 271(2):122–130
4. Bastrakov S, Meyerov I, Gergel V, Gonoskov A, Gorshkov A, Efimenko E, Ivanchenko M, Kirillin M, Malova A, Osipov G et al (2013) High performance computing in biomedical applications. Procedia Comput Sci 18:10–19
5. Qian K, Guo J, Xu H, Zhu Z, Zhang G (2014) Snore related signals processing in a private cloud computing system. Interdiscip Sci 6(3):216–221
6. I. Dogaru and R. Dogaru (2015) Using python and julia for efficient implementation of natural computing and complexity related algorithms. In: 2015 20th International Conference on Control Systems and Computer Science. IEEE, pp. 599–604
7. Strollo PJ Jr, Rogers RM (1996) Obstructive sleep apnea. N Engl J Med 334(2):99–104
8. Pevernagie D, Aarts RM, De Meyer M (2010) The acoustics of snoring. Sleep Med Rev 14(2):131–144
9. Mesquita J, Solà-Soler J, Fiz JA, Morera J, Jané R (2012) All night analysis of time interval between snores in subjects with sleep apnea hypopnea syndrome. Med Biol Eng Computing 50(4):373–381
10. Abeyratne U, De Silva S, Hukins C, Duce B (2013) Obstructive sleep apnea screening by integrating snore feature classes. Physiol Meas 34(2):99
11. M. Schmitt, C. Janott, V. Pandit, K. Qian, C. Heiser, W. Hemmert, and B. Schuller (2016) A bag-of-audio-words approach for snore sounds excitation localisation. In: Proceedings of 12th ITG Conference on Speech Communication. pp. 230–234
12. Qian K, Janott C, Pandit V, Zhang Z, Heiser C, Hohenhorst W, Herzog M, Hemmert W, Schuller B (2017) Classification of the excitation location of snore sounds in the upper airway by acoustic multi-feature analysis. IEEE Trans Biomed Eng 64(8):11
13. A. Adeshina and R. Hashim (2016) Computational approach for securing radiology-diagnostic data in connected health network using high-performance gpu-accelerated aes. Interdiscip Sci 1–13
14. Huijie X, Weining H, Yulisheng CL (2011) Spectral analysis of snoring sound and site of obstruction in obstructive sleep apnea/hypopnea syndrome. Am J Audiol Speech Pathol 1:009
15. Azarbarzin A, Moussavi ZM (2011) Automatic and unsupervised snore sound extraction from respiratory sound signals. IEEE Trans Biomed Eng 58(5):1156–1162
16. Al-Ameen Z, Sulong G (2015) Deblurring computed tomography medical images using a novel amended landweber algorithm. Interdiscip Sci 7(3):319–325
17. Van Der Walt S, Colbert SC, Varoquaux G (2011) The numpy array: a structure for efficient numerical computation. Computing Sci Eng 13(2):22–30
18. E. Jones, T. Oliphant, P. Peterson *et al.* (2001) Open source scientific tools for python
19. G. Van Rossum *et al.* (2007) Python programming language. In: *USENIX Annual Technical Conference*. vol. 41
20. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014) Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia. ACM, pp. 675–678
21. T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang (2015) Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274
22. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.* (2015) Tensorflow: large-scale machine learning on heterogeneous systems, Software available from tensorflow. org 1
23. Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A et al (2011) Theano: deep learning on gpus with python, in NIPS 2011. BigLearning Workshop, Granada
24. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830
25. C. Analytics (2015) Anaconda software distribution, Computer software, nov. [Online]. Available: https://continuum.io
26. K. Qian, C. Janott, Z. Zhang, C. Heiser, and B. Schuller, "Wavelet Features for Classification of VOTE Snore Sounds," in Proceedings ICASSP.Shanghai, P. R. China: IEEE, 2016, pp. 221–225