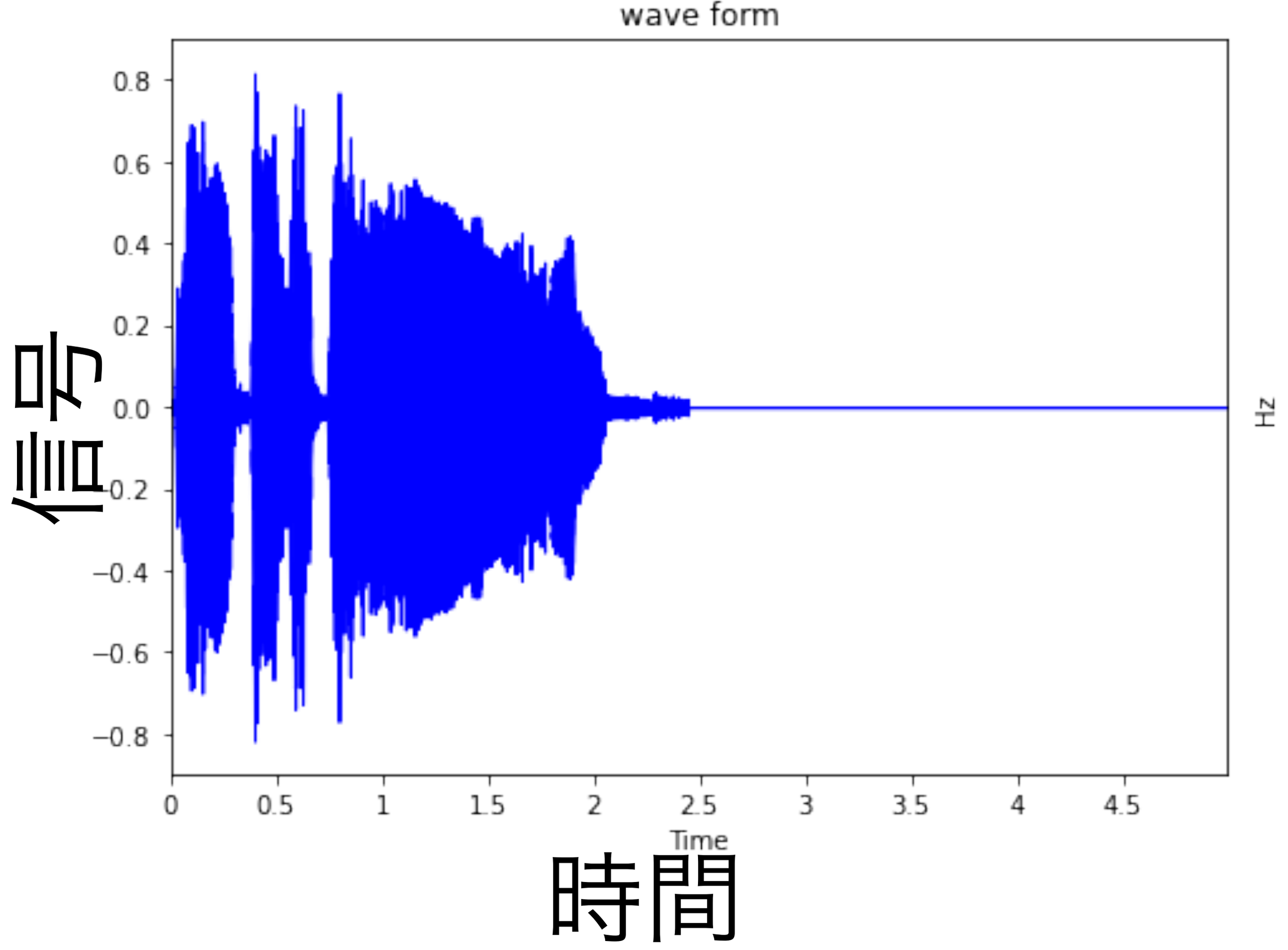
音声解析入門

Acoustic/Speech Analysis

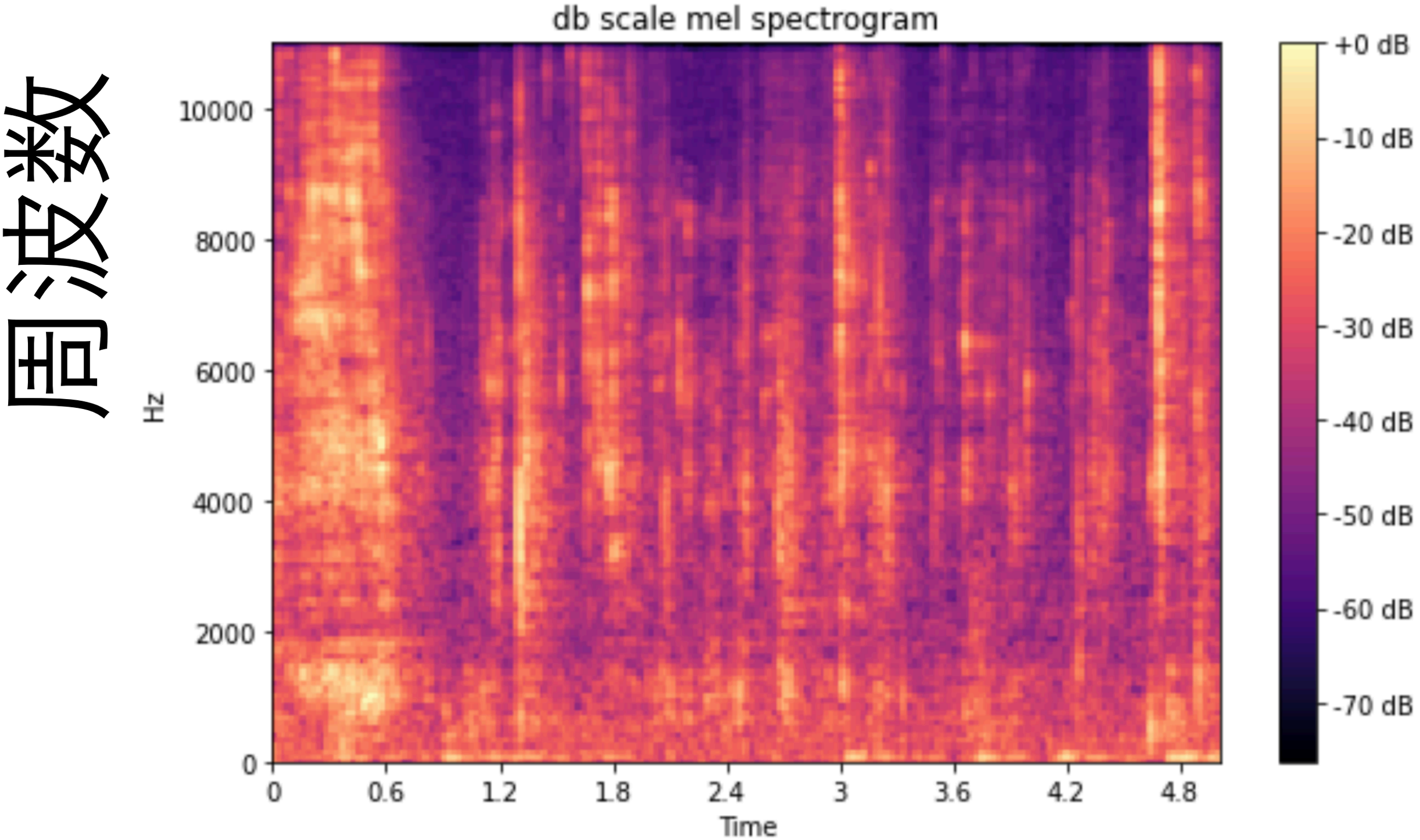
メルスペクトログラム

スペクトルの可視化

メル尺度／メルスペクトル


wave form
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Time
Hz
時間

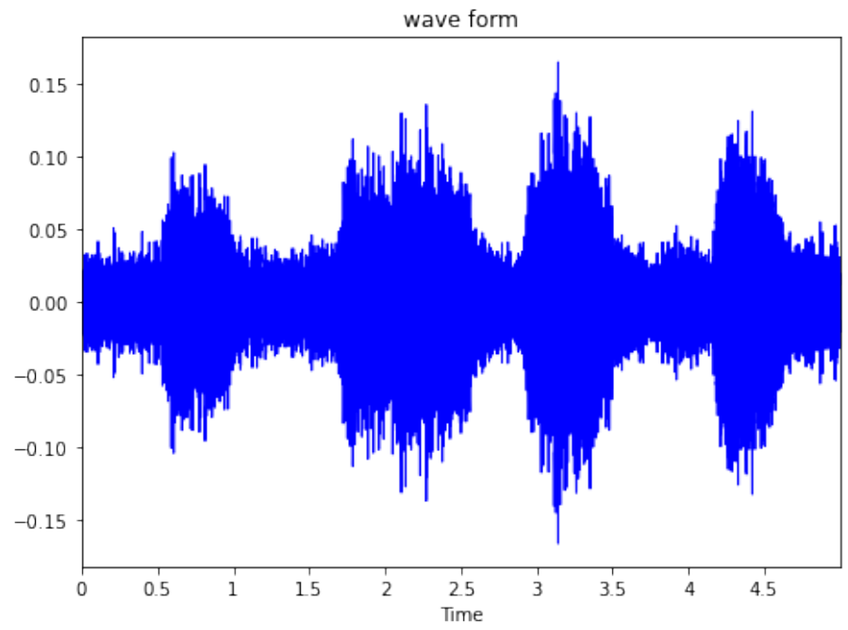
メルスペクトログラム

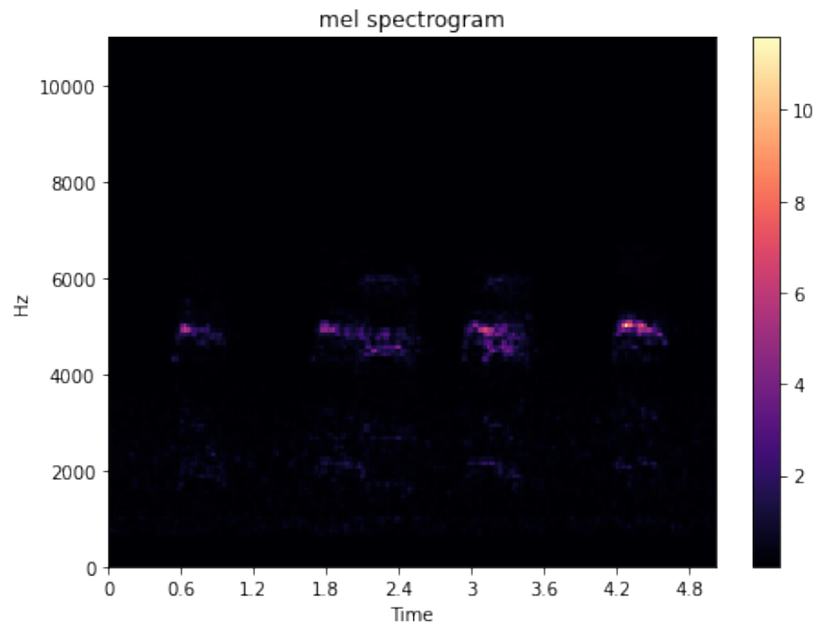

db scale mel spectrogram
周波数
Hz
10000
8000
6000
4000
2000
0
0
0.6
1.2
1.8
2.4
3
3.6
4.2
4.8
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB

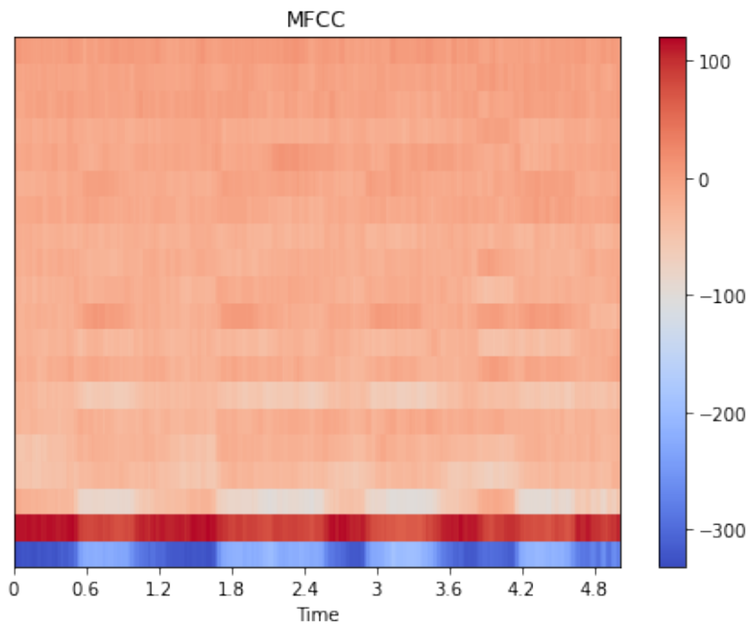
音響／音声解析のアイディア

- 時系列の信号をそのまま取り扱う：LSTM

- **音を画像に変換して画像分類の問題に置き換える**



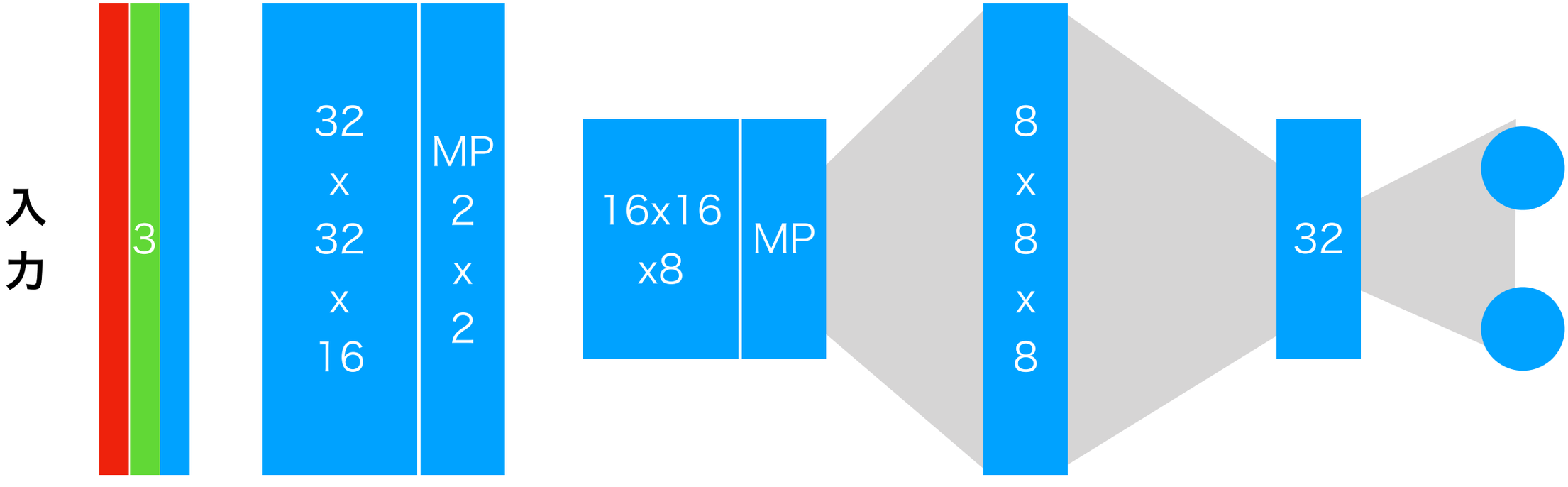




音をどう画像に変換するか？

画像に変換できれば

演習１と演習２と同じ



音声を画像に変換する方法

スペクトル

信号のフーリエ変換

メルスペクトル

スペクトルに人の聴覚特性のフィルタバンクを乗算した係数

MFCC

Mel-Frequency Cepstrum Coefficients: **MFCC**

メルスペクトルのコサイン変換の係数

PCEN

Per-Channel Energy Normalization

https://ieeexplore.ieee.org/document/8514023

Trainable Frontend For Robust and Far-Field Keyword Spotting

Yuxuan Wang, Pascal Getreuer, Thad Hughes, Richard F. Lyon, Rif A. Saurous

Google, Mountain View, USA

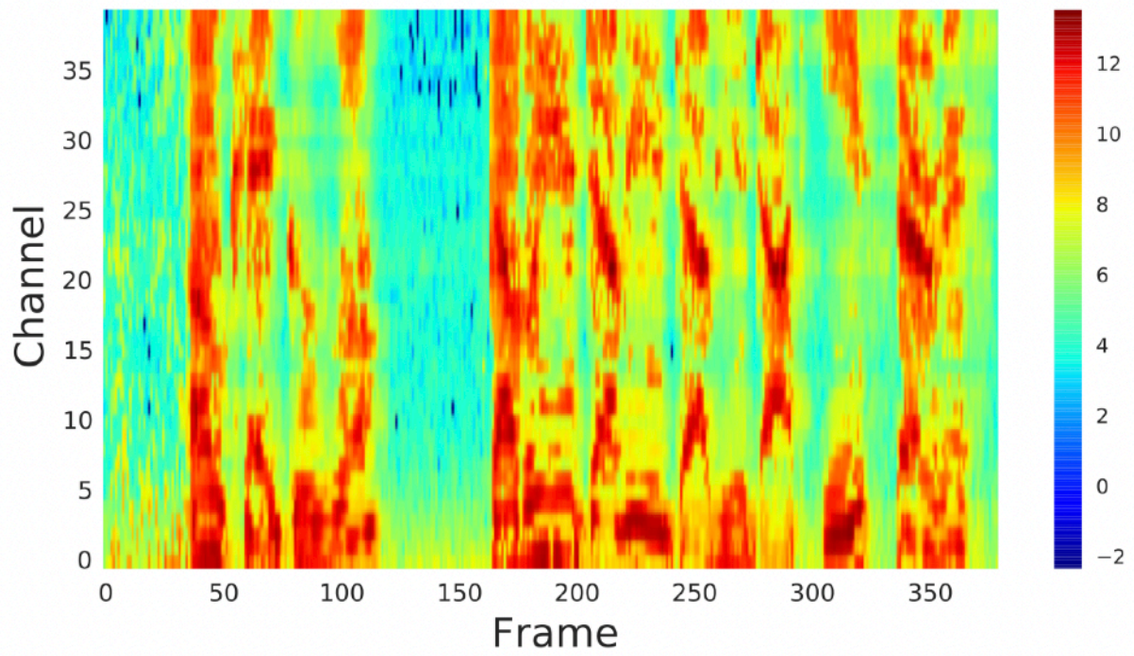
{yxwang,getreuer,thadh,dicklyon,rif}@google.com

Abstract

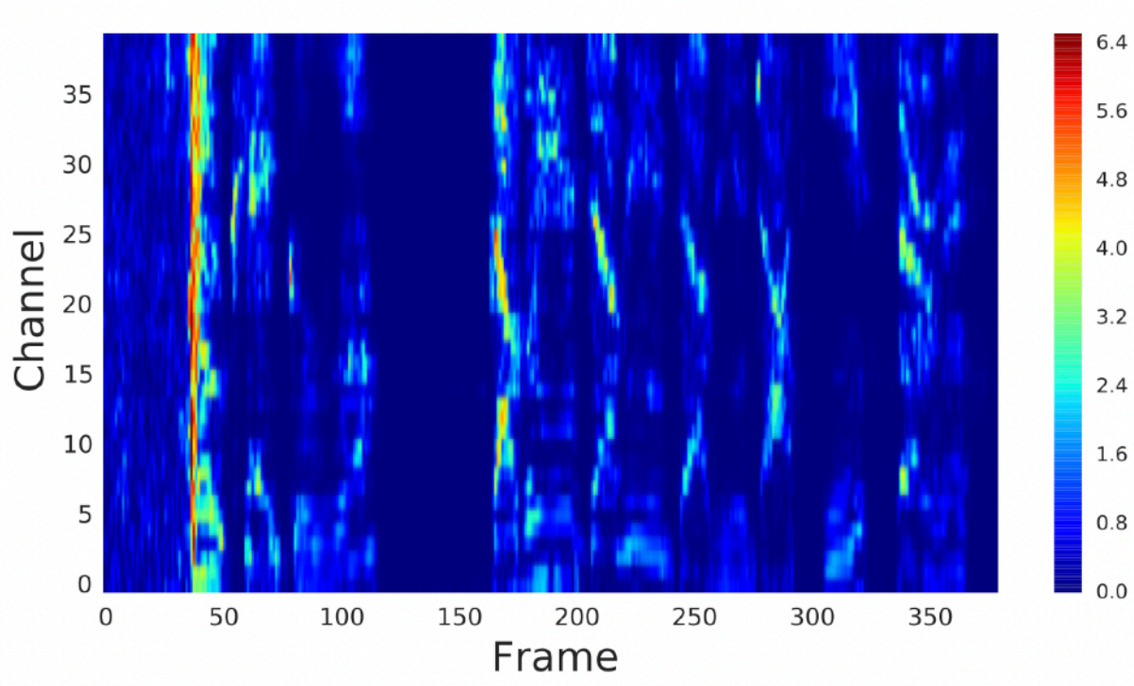
Robust and far-field speech recognition is critical to enable true hands-free communication. In far-field conditions, signals are attenuated due to distance. To improve robustness to loudness variation, we introduce a novel frontend called per-channel energy normalization (PCEN). The key ingredient of PCEN is the use of an automatic gain control based dynamic compression to replace the widely used static (such as log or root) compression. We evaluate PCEN on the keyword spotting task. On our large rerecorded noisy and far-field eval sets, we show that PCEN significantly improves recognition performance. Furthermore, we model PCEN as neural network layers and optimize high-dimensional PCEN parameters jointly with the keyword spotting acoustic model. The trained PCEN frontend demonstrates significant further improvements without increasing model complexity or inference-time cost.

Index Terms: Keyword spotting, robust and far-field speech recognition, automatic gain control, deep neural networks

used frontend is the so-called log-mel frontend, consisting of mel-filterbank energy extraction followed by log compression, where the log compression is used to reduce the dynamic range of filterbank energy. However, there are several issues with the log function. First, a log has a singularity at 0. Common methods to deal with the singularity are to use either a clipped log (i.e. $\log(\max(\text{offset}, x))$) or a stabilized log (i.e. $\log(x+\text{offset})$). However, the choice of the offset in both methods is ad hoc and may have different performance impacts on different signals. Second, the log function uses a lot of its dynamic range on low level, such as silence, which is likely the least informative part of the signal. Third, the log function is loudness dependent. With different loudness, the log function can produce different feature values even when the underlying signal content (e.g. keywords) is the same, which introduces another factor of variation into training and inference. Although techniques such as mean–variance normalization [6] and cepstral mean normalization [7] can be used to alleviate this issue to some extent, it is nontrivial to deal with time-varying loud-



(a) log-mel frontend



(b) PCEN frontend

Figure 1: *log-mel and PCEN features on a speech utterance.*

メルスペクトログラム
ECS-50の利用

Audio

ESC-50

Introduced by Karol J. Piczak in ESC: Dataset for Environmental Sound Classification

The **ESC-50** dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification. It comprises 2000 5s-clips of 50 different classes across natural, human and domestic sounds, again, drawn from Freesound.org.

Source: The NIGENS General Sound Events Database

L05_01_MelSpectrogram

The latest from Google Research

特徴の比較

Launching the Speech Commands Dataset

Thursday, August 24, 2017

Posted by Pete Warden, Software Engineer, Google Brain Team

Speech Command Dataset

At Google, we're often asked how to get started using deep learning for speech and other audio recognition problems, like detecting keywords or commands. And while there are some great open source speech recognition systems like Kaldi that can use neural networks as a component, their sophistication makes them tough to use as a guide to a simpler tasks. Perhaps more importantly, there aren't many free and openly available datasets ready to be used for a beginner's tutorial (many require preprocessing before a neural network model can be built on them) or that are well suited for simple keyword detection.

To solve these problems, the TensorFlow and AIY teams have created the Speech Commands Dataset, and used it to add training* and inference sample code to TensorFlow. The dataset has 65,000 one-second long utterances of 30 short words, by thousands of different people, contributed by members of the public through the AIY website. It's released under a Creative Commons BY 4.0 license, and will continue to grow in future releases as more contributions are received. The dataset is designed to let you build basic but useful voice interfaces for applications, with common words like "Yes", "No", digits, and directions included. The infrastructure we used to create the data has been open sourced too, and we hope to see it used by the wider community to create their own versions, especially to cover underserved languages and applications.

To try it out for yourself, download the prebuilt set of the TensorFlow Android demo applications and open up "TF Speech". You'll be asked for permission to access your microphone, and then see a list of ten words, each of which should light up as you say them.

L05_02 to L05_04

https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

音声から「文字起こし」

wave2vec

L05_02 to L05_05

```
SPEECH_URL = "https://pytorch-tutorial-assets.s3.amazonaws.com/VOiCES_dev
SPEECH_FILE = "_assets/speech.wav"
```

```
if not os.path.exists(SPEECH_FILE):
    os.makedirs("_assets", exist_ok=True)
    with open(SPEECH_FILE, "wb") as file:
        file.write(requests.get(SPEECH_URL).content)
```

```
bundle = torchaudio.pipelines.WAV2VEC2_ASR_BASE_960H

print("Sample Rate:", bundle.sample_rate)

print("Labels:", bundle.get_labels())
```

```
Sample Rate: 16000
Labels: ('-', '|', '
'J', 'Q', 'Z')
```

```
model = bundle.

print(model.__c
```

```
<class 'torchaudio.m
```

```
IPython.display
```

▶ 0:03 / 0:03

```
print(transcript)
IPython.display.Audio(SPEECH_FILE)
```

```
I|HAD|THAT|CURIOSITY|BESIDE|ME|AT|THIS|MOMENT|
```

▶ 0:00 / 0:03