卒業課題

マンション管理における
外壁タイル打音検診の異常音検知

DIVEINTOCODE 2020年１月期

加藤　裕也

自己紹介

加藤　裕也（かとう　ゆうや）
神奈川県出身　1990年生まれ

＜前職＞
建設業向けに画像などを利用した業務改善アプリ開発会社の
営業職を約6年間経験

＜機械学習のきっかけ＞
建設業など特に人材不足の業種に向けて機械学習を一つのツールとして、
人材不足に役立つサポートができるのではないかと感じていたので。

＜興味のある分野＞
画像分野、セグメンテーションなどの物体検知での空間認識など
人の五感をサポートできるようなソリューション

マンション管理における外壁タイルの維持管理について

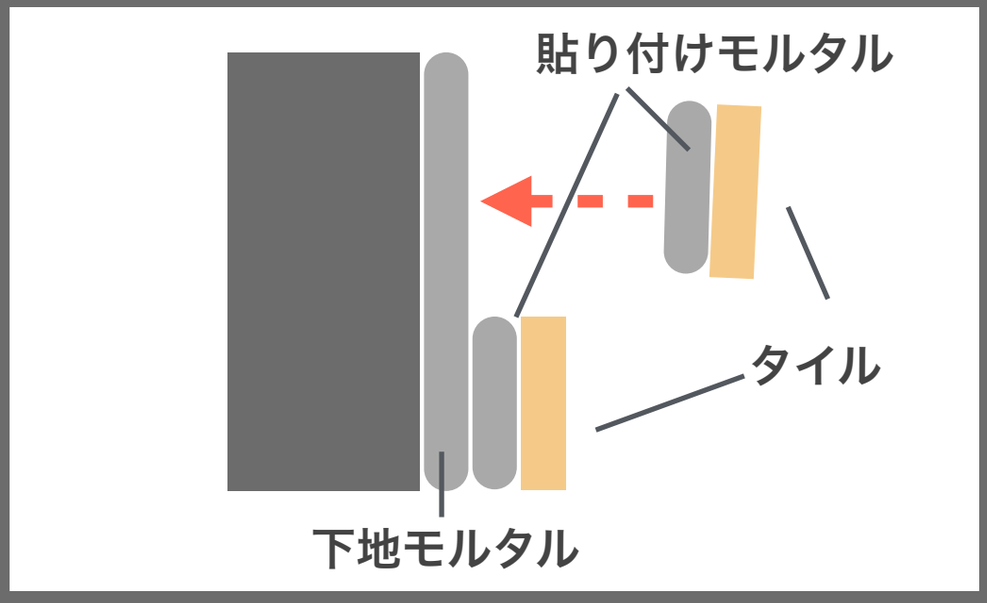
・マンションの維持管理は、所有者である管理組合の義務であり、定期報告制度がある。

建築基準法では、１.特定建築物、２.防火設備、３.昇降機、遊戯施設、４.特定建築物に設ける建築設備について、その所有者・管理者が、安全を確保するため、専門技術者に定期的に調査・検査をさせて、その結果を特定行政庁に報告することが定められています。（＊一般社団法人日本建築防災協会ホームページ抜粋）

- 政令および特定行政庁で対象とする建物が指定されている（用途、規模等）。
- タイル仕上げ外壁等」は定期調査の対象になっている：
  a. 半年～3年に1回毎に手の届く範囲を打診検査して、その結果を報告する
  b. 竣工から10年（大規模修繕から10年）を目安に、
  外壁全面を打診等により 調査を実施し、報告する

外壁タイルの異常について

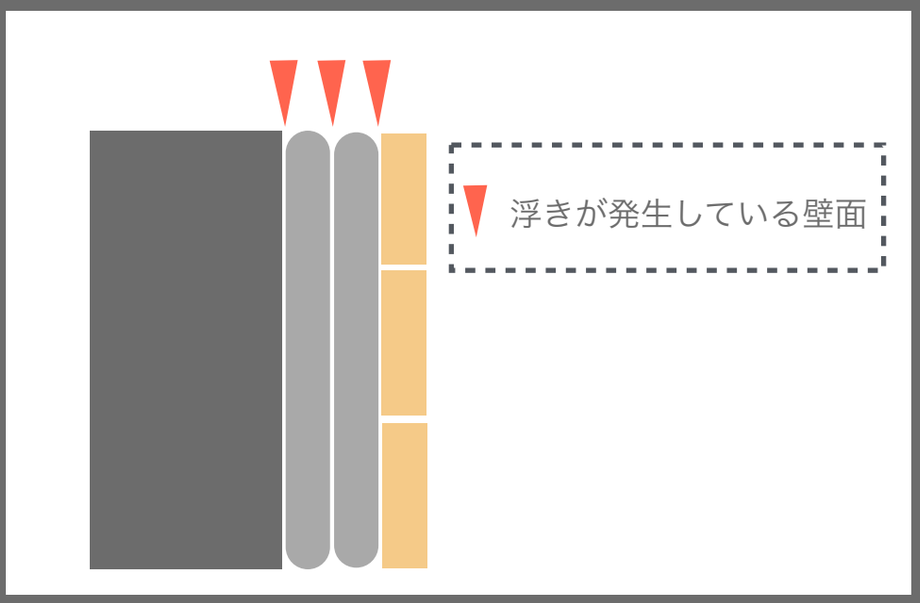
外壁タイルの施工イメージ（＊今回は湿式工法のみ）



躯体（コンクリ）に下地モルタルを塗り、
タイル側に貼り付けモルタルを塗り圧着させる方法
＊そのほかにも、様々な施行方法がある

外壁タイルの劣化について

外壁は経年劣化により浮き（剥離）、ひび割れ、欠損などが生じて生じる。



①タイル陶片の浮き

・タイルと張付けモルタルの接着不良
・張付けモルタルの凝集力の不足

②タイル張りの浮き

・モルタル塗りつけ界面の接着不良
・モルタルの凝集力不足

③躯体コンクリートの境界面での浮き

・コンクリート自体の強度劣化や鉄筋 などのさび

適用されている外壁調査方法（目視検査は基本として除外）

今回はこの方法での異常音検知を試す

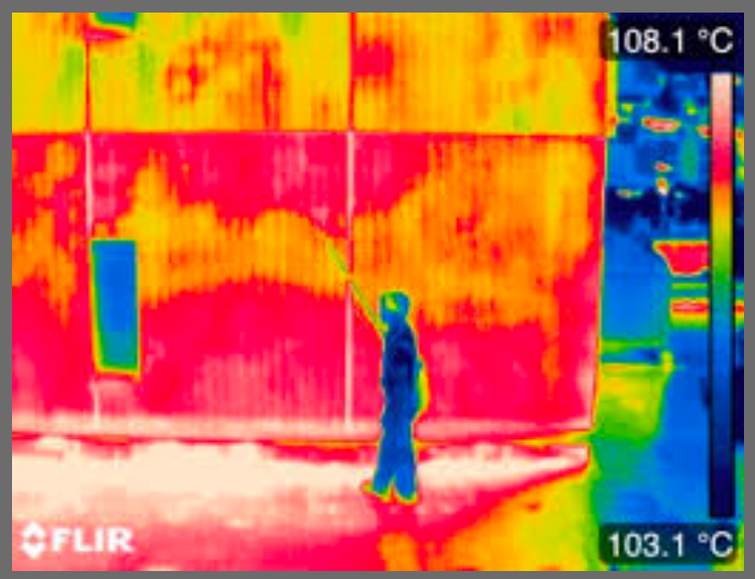
① **打診法**；

テストハンマーなどで壁面を打撃し、その音の差異を耳で 聞き分けて、健全であるか、もしくは浮きがあるかを判別 する

https://www.youtube.com/watch?v=jYEABaSKrxE

② **赤外線装置法**；

外壁の表面温度の差を赤外線装置を用いて測定し、浮き部 分を検出する



打診法による診断の課題

課題について

・検査員の経験・技能により結果のばらつきが 生じることもある

・長時間作業を行っていると、判断力が低下す る場合もある

（適宜、休憩を挟んで実施）

・仮設足場等を設置する必要があるため、費用 がかかる

つまり打診法による外壁の診断は、人に依存していて
能力や疲れによって、誤診断する可能性が高くなっていく。

機械学習を用いた異常検知で異常音の検知できるか！

機械学習で異常検知をする利点

・経験、技能による判定の違いがなくなる

・打音検査で人の耳でもわかる範囲以上の
異常音の違いを見抜ければ早期発見という面で役立つ

・高所など危険な場所での作業はロボットなどに任せることもできる

＜注意＞
今回は、データセットとなる打音検査の音源を手に入れることができなかったので、DCASE2020「Unsupervised Detection of Anomalous Sounds for Machine Condition Monitoring」の異常音検知コンペのデータを使って異音検知できるか試します。

異音検知の機械学習流れ

データ準備
＊WAVデータでない場合は、WAV

DCASE 2020のデータを利用
※正常音のみの学習

雑音除去（余裕があれば）

データの変換
＊メルスペクトル変換など行い
　音データを可視化できるようにする

Librosaを使いデータの変換。
変換については、後述

データ水増し

音声データにホワイトノイズ、シフト、
ストレッチを加えデータの水増し

モデル選定・学習

モデルはオートエンコーダが主流。
評価指標はAUC。PAUC利用

推定

WAVデータのへの変換（データ前処理）

MP3などの音声ファイルの場合、WAV変換を行う。
Pythonモジュールの「Pydub」を利用し変換が行える。

```
import pydub
sound = pydub.AudioSegment.from_mp3('input.mp3')
sound.export('output.wav', format='wav')
```

※エラーが出る場合はffmpegをダウンロードすると解決できる可能性あり

今回のデータセットの音について

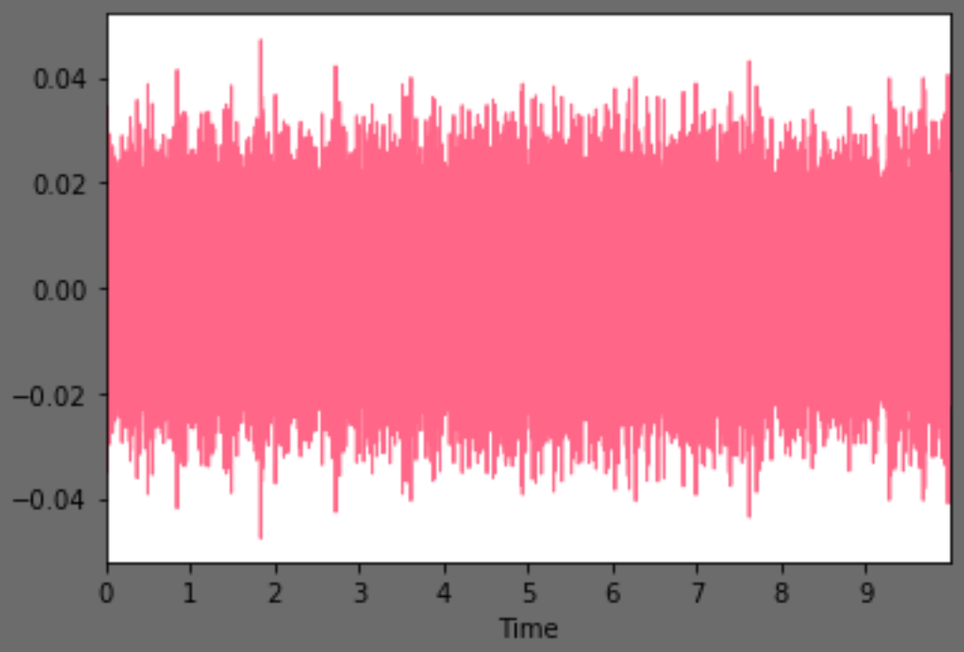
DCASE 2020のFanの音について分析

traindata : 3675　,　testdata : 1875
全て１０秒間のデータで環境音なども含まれているモノラル音。
一般的な音源と同様に44100HZのサンプリング周波数。

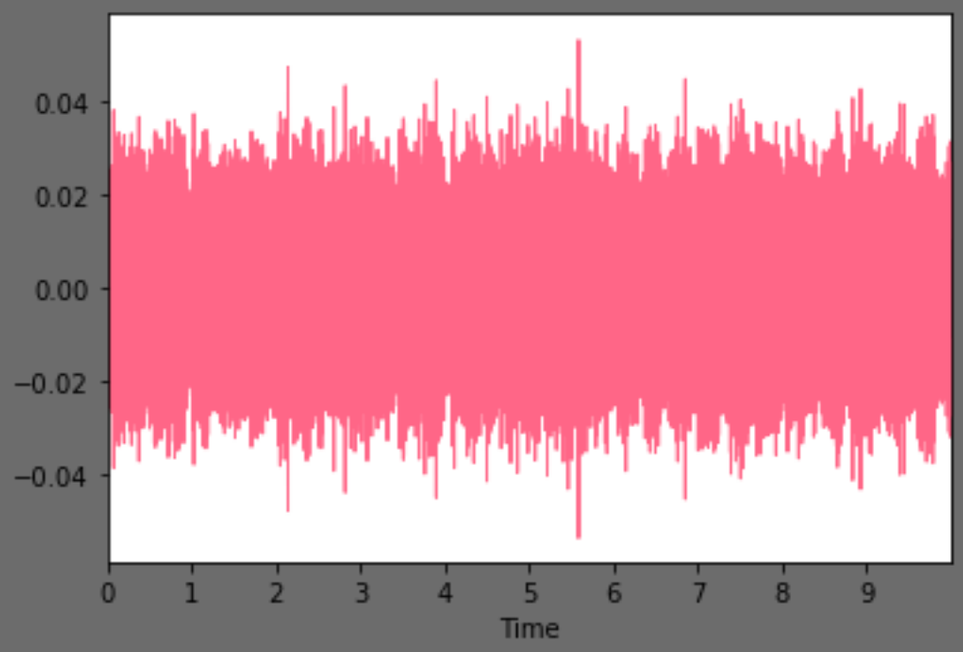
パラメーター数を考えると

44100(hz) * **10**(sec) * **1**(channel) = **441000**個

正常音



異常音



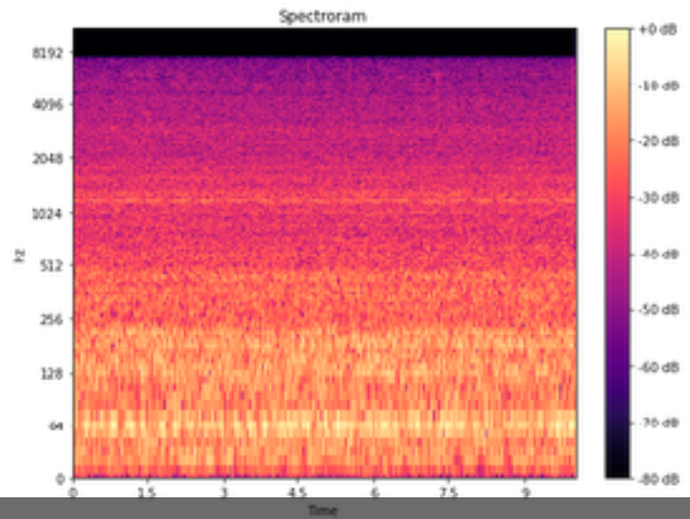
波形データを見ても多少の違いがあるが分かりづらい

データ変換（前処理）

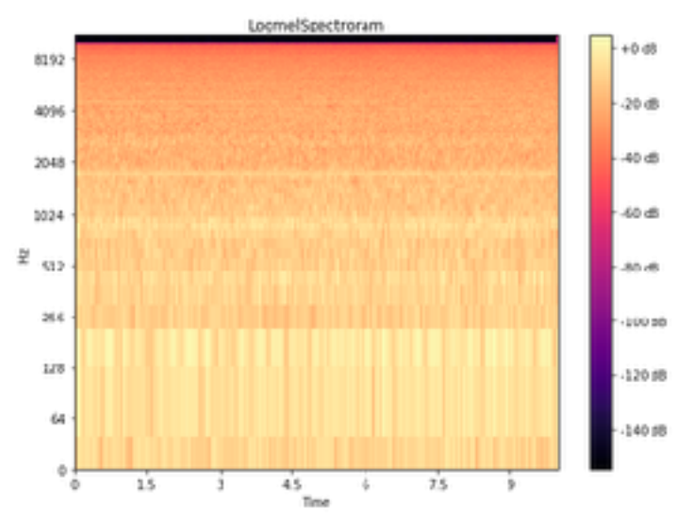
WAVデータを前処理（フーリエ変換）
近年音声解析ではログメルスペクトログラムを使うことがスタンダード。
しかし、今回は異常音検知のため、スペクトログラムも検証。

※ログメルだと人間が認識できる音の範囲にデータが圧縮される。
　今回の異常音検知では人の認識できない音にも着目する必要があるため
　スペクトログラムにするがデータが重くなってしまうのが、学習の際の難点。

```
import librosa
y, sr = file_load(file_name)
S = numpy.abs(librosa.stft(y,
      n_fft=1024, hop_length=128))
log_stft = librosa.power_to_db(S)
```




```
import librosa
y, sr = file_load(file_name)
MS = librosa.feature.melspectrogram(y, sr,
       n_fft=1024, hop_length=128)
LMS= 20.0 / power *
        np.log10(MS+sys.float_info.epsilon)
```



データ水増し（前処理）

データ数がない場合は、水増し必要になる。
Kaggleノートブックで公開されていた「TensorFlow Speech Recognition Challenge」の下記の方法で対応可能。
（今回はコンペの３０００件のトレーニングデータセットがあるため、水増しは行わない）

```
# data augmentation: add white noise
def add_white_noise(self, rate=0.002):
    return self.x + rate*np.random.randn(len(self.x))
# data augmentation: shift sound in timeframe
def shift_sound(self, rate=2):
    return np.roll(self.x, int(len(self.x)//rate))
# data augmentation: stretch sound
def stretch_sound(self, rate=1.1):
    input_length = len(self.x)
    x = librosa.effects.time_stretch(self.x, rate)
    if len(self.x)>input_length:
        return self.x[:input_length]
    else:
        return np.pad(self.x, (0, max(0, input_length - len(x))), "constant")
```

モデルの選定

異常検知の分野ではAutoEncoder主流だが、画像分野の異常検知では
MetricLearningやAnoGANなども利用されている。
今回は、AutoEncoder（畳み込み）と時間があればMetricLearning
AnoGANも利用したい

＜モデルの簡易説明＞

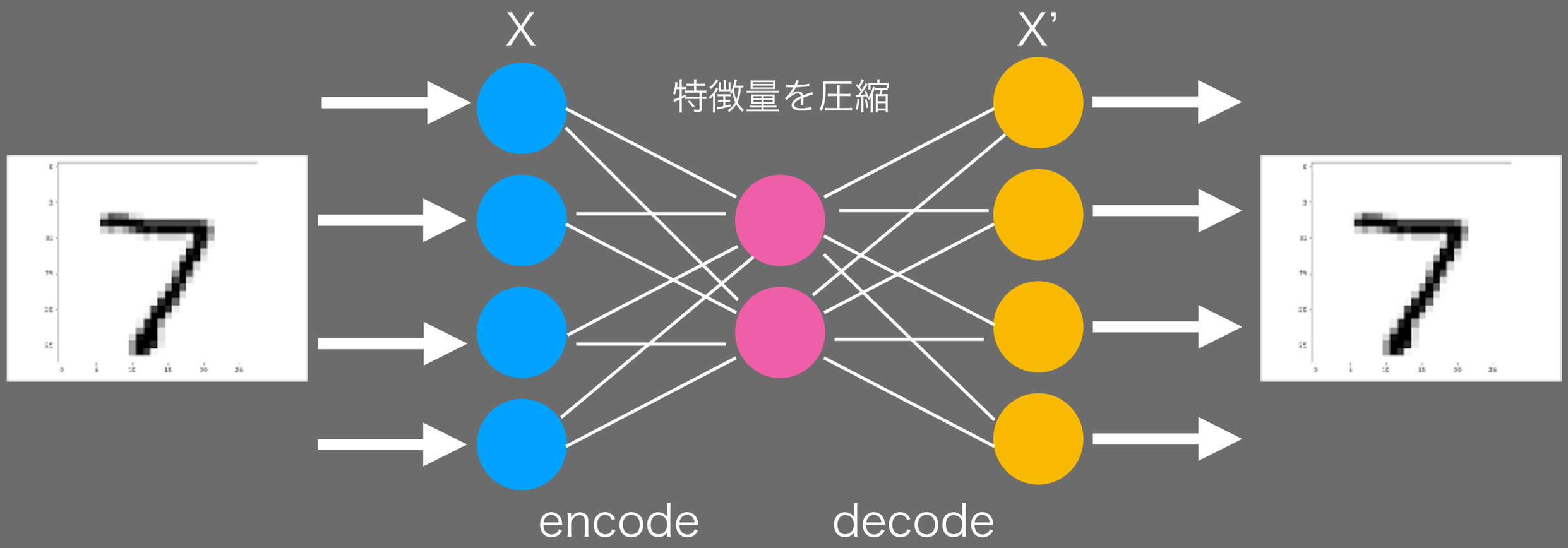
AutoEncoder：
入力と出力が同じになるようにニューラルネットワークを学習させるような手法

MetricLearning：
データ間の関係性を考慮した特徴量空間を学習する手法

AnoGAN：
GANの入力を適切に変化させ、検査する画像に出来るだけ近い画像を生成させ
異常と判断する手法

今回利用した一般的なAutoEncoder

オートエンコーダでは、圧縮していく過程を**エンコーダ**と呼び、復元する過程を**デコーダ**と呼ぶ。エンコーダは入力を低次元に表現することができ、デコーダは低次元から復元する能力を持ちます。正常データを学習することで、入力値に異常がある場合、入力値と出力値の間に誤差が生じます。



今回利用した一般的なAutoEncoder

```
inputLayer = Input(shape=(inputDim,))
h = Dense(128)(inputLayer)
h = BatchNormalization()(h)
h = Activation(‘relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(8)(h)
h = BatchNormalization()(h)
h = Activation(‘relu’)(h)
```

```
h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(128)(h)
h = BatchNormalization()(h)
h = Activation('relu')(h)

h = Dense(inputDim)(h)
```

目的関数、評価指標

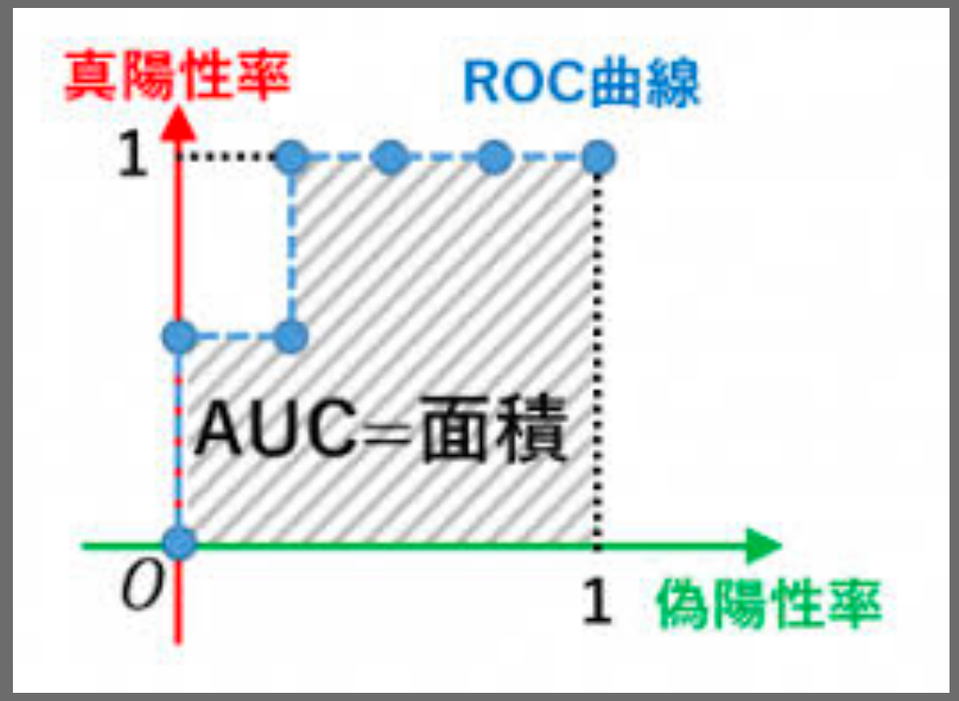
目的関数：mean_squared_error

→ 生成された音データと正常音の比較し誤差を小さくしていくため利用

評価指標：AUC、p_AUC

＜AUC＞

AUCはROC曲線の下側に面積のこと。ROC曲線は横軸にFalse Positive Rate、
縦軸にTrue Positive Rateを取ったグラフのこと。
AUCは0から1までの値をとり、値が1に近いほど判別能が高いことを示す。
判別能がランダムであるとき、AUC = 0.5となる。



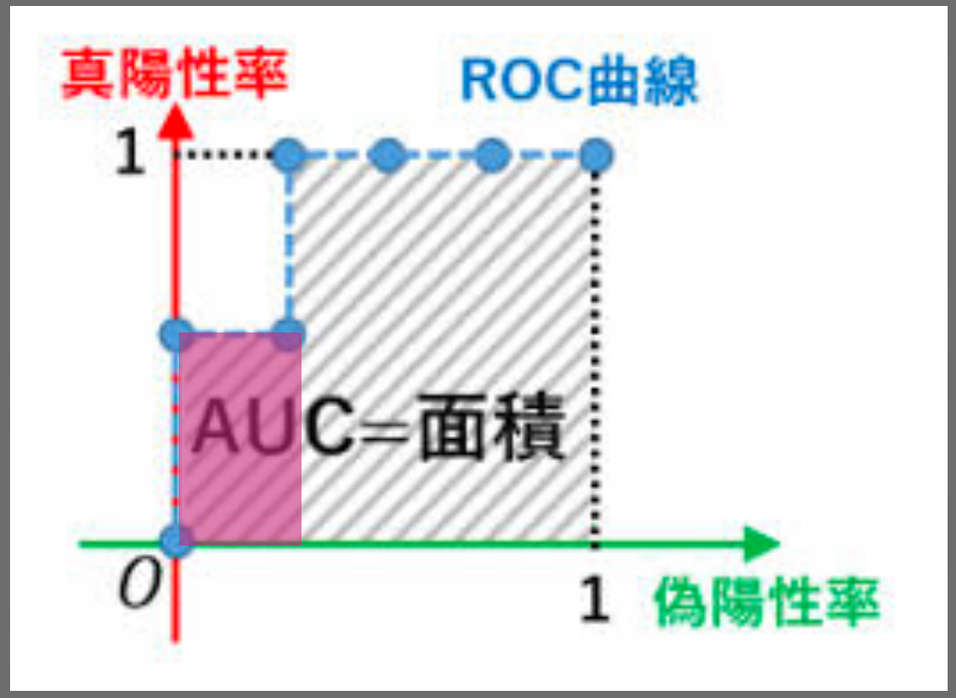
$$AUC = \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} I\left(f(x^+) > f(x^-)\right), I(x) = \begin{cases} 1 & x = True \\ 0 & x = False \end{cases}$$

目的関数、評価指標

＜pAUC＞

AUCのx軸であるFalse Positive Rateに制限をつけたもの。

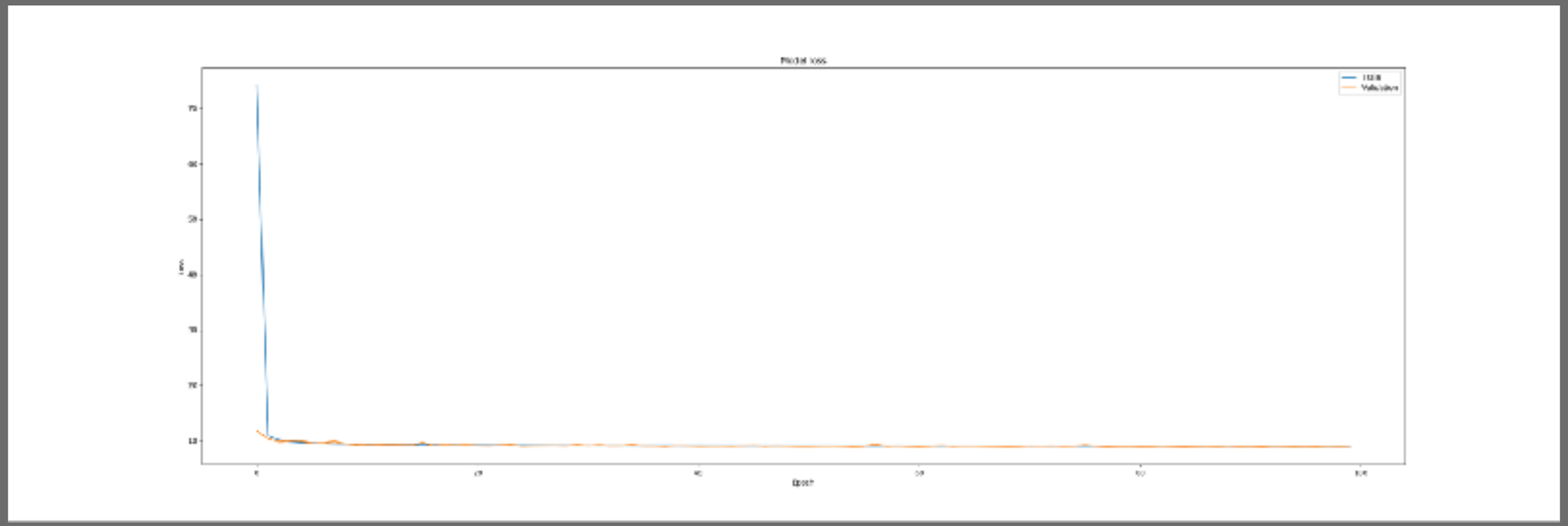
下図のように赤く塗った箇所のFalse Postive Rateが低い部分のTrue Positive Rateを高くすれば、より効率よくAUCを最大化することができるというモチベーション。



$$\widehat{pAUC}(\theta) = \frac{1}{n^+n^-(\beta-\alpha)}\sum_{i=1}^{n^+}[(j_\alpha - \alpha n^-)s(x_i^+, x_{(j_\alpha)}^-;\theta) + \sum_{j=j_\alpha+1}^{j_\beta} s(x_i^+, x_{(j)}^-;\theta) + (\beta n^- - j_\beta)s(x_i^+, x_{(j_\beta+1)}^-;\theta)]$$

結果

＜ベースライン＞
ログメル×オートエンコーダ

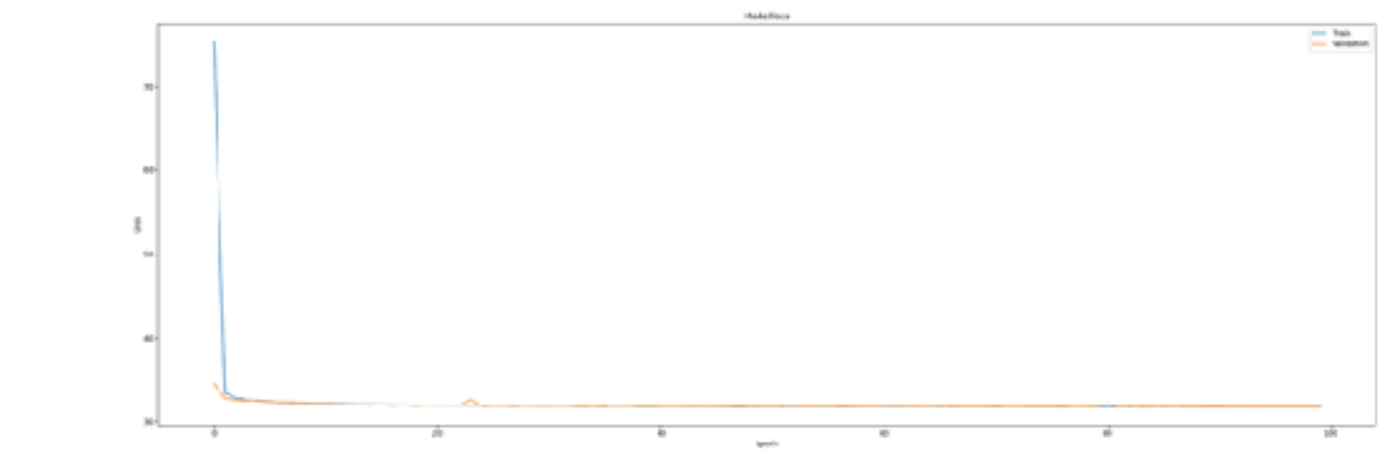


result

fan		
id	AUC	pAUC
0	0.554717444717445	0.494504073451442
2	0.734401114206128	0.54420172995162
4	0.642270114942529	0.526013309134906
6	0.750858725761773	0.526607377168683
Average	0.670561849906969	0.522831622426663

結果

スペクトログラムxオートエンコーダー



```
Epoch 1/100
1022017/1022017 [==============================]
- 98s 96us/step - loss: 75.2880 - val_loss: 34.5054
Epoch 2/100
1022017/1022017 [==============================]
- 86s 84us/step - loss: 33.3605 - val_loss: 32.7994
Epoch 3/100
1022017/1022017 [==============================]
- 86s 84us/step - loss: 32.8233 - val_loss: 32.5687
Epoch 4/100
1022017/1022017 [==============================]
- 87s 85us/step - loss: 32.6214 - val_loss: 32.5677
Epoch 5/100
1022017/1022017 [==============================]
```

⋮

```
Epoch 96/100
1022017/1022017 [==============================]
- 86s 84us/step - loss: 31.8078 - val_loss: 31.7937
Epoch 97/100
1022017/1022017 [==============================]
- 85s 84us/step - loss: 31.8072 - val_loss: 31.7909
Epoch 98/100
1022017/1022017 [==============================]
- 87s 85us/step - loss: 31.8055 - val_loss: 31.8086
Epoch 99/100
1022017/1022017 [==============================]
- 86s 84us/step - loss: 31.8048 - val_loss: 31.7661
Epoch 100/100
1022017/1022017 [==============================]
- 86s 84us/step - loss: 31.8035 - val_loss: 31.7799
2020-04-29 07:39:10,681 - INFO - save_model -> ./model/
model_fan.hdf5
============== END TRAINING ==============4
```

result

fan		
id	AUC	pAUC
0	0.548894348894349	0.495409284882969
2	0.734094707520891	0.544641548160094
4	0.561925287356322	0.517392619479734
6	0.653684210526316	0.512611167808719
Average	0.624649638574469	0.517513655082879

結果

＜ベースライン＞
ログメル×畳み込みオートエンコーダー

```
Epoch 1/100
1022017/1022017 [==============================]
- 682s 667us/step - loss: 994.3393 - val_loss: 1001.8195
Epoch 2/100
 587776/1022017 [==========>.....] - ETA:
4:36 - loss: 985.2998channel 3: open failed: connect failed:
Connection refused
channel 4: open failed: connect failed: Connection refused
1022017/1022017 [==============================]
- 676s 662us/step - loss: 984.4968 - val_loss: 997.2944
Epoch 3/100
1022017/1022017 [==============================]
- 676s 661us/step - loss: 981.8330 - val_loss: 995.8619
Epoch 4/100
1022017/1022017 [==============================]
- 676s 661us/step - loss: 980.8569 - val_loss: 995.2435
Epoch 5/100
1022017/1022017 [==============================]
- 676s 661us/step - loss: 980.4633 - val_loss: 995.0158
Epoch 6/100
1022017/1022017 [==============================]
- 675s 660us/step - loss: 980.2995 - val_loss: 994.9168
Epoch 7/100
1022017/1022017 [==============================]
- 674s 659us/step - loss: 980.2321 - val_loss: 994.8756
Epoch 8/100
1022017/1022017 [==============================]
- 671s 657us/step - loss: 980.2060 - val_loss: 994.8580
Epoch 9/100
1022017/1022017 [==============================]
- 670s 656us/step - loss: 980.1907 - val_loss: 994.8508
Epoch 10/100
 272896/1022017 [=======>......................] - ETA: 7:52 -
loss: 980.1603channel 3: open failed: connect failed:
```

検証したがあまり良い結果が得られなかった。
深層学習の場合、どれだけ層を深くするか検討が必要そう。
あまりに深くなると特徴量がのこならくなってしまうのかもしれない。

検証結果から見える課題と今後のできそうなこと

＜課題＞

・正常音と異常音の差となる特徴量をキャッチすることがオートエンコーダでの学習には必須。今回のスペクトログラム変換については特徴量が多く再現性の悪い結果となった。単に特徴量が多いことが良いことでもない。

・音データの形状などちゃんと理解し、
どこで音を切り取ったほうが良いかなどの検証が足りなかった。

・打音検診のデータをどう入手するか。
自身で収集が難しい場合、団体や企業にアプローチも必要と感じた。

検証結果から見える課題と今後のできそうなこと

＜今後できそうなこと＞

・音声のノイズ削除をしっかり行ってから学習。
（https://shingi.jst.go.jp/var/rev0/0000/6114/2017_4u3_5.pdf）

・音声データから時系列の波形を作成し時系列データの特徴量を作成

・音声は時系列データなのでLSTMを使った分析で検証。

・もう少し違いを見つけやすいものから試行して、
結果の出た方法で試すなど。

この３〜４ヶ月振り返って

・スクラッチを経験することで、各機械学習手法がどのように
動いているのが理解することができた。

・生徒同士お互いの学習共有など、アドバイスされたり
したりすることで自分の理解を深められてよかった。

・オフラインだったので、つまづいた際にメンターに
直接聞けることで自分一人で悩まず学習進められてよかった。

以上です